Generic Named Entity Extraction

José Luis Jara-Valencia

This thesis is submitted in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy.

University of York
Department of Computer Science

July 2005

Abstract

This thesis proposes and evaluates different ways of performing generic named entity recognition, that is the construction of a system capable of recognising names in free text which is not specific to any particular domain or task.

The starting point is an implementation of a well known baseline system which is based on maximum entropy models that utilise lexically-oriented features to recognised names in text. Although this system achieves good levels of performance, both maximum entropy models and lexically-oriented features have their limitations. Three alternative ways in which this system can be extended to overcome these limitations are then studied:

- more linguistically-oriented features are extracted from a generic lexical source, namely WordNet®, and then added to the pool of features of the maximum entropy model
- the maximum entropy model is bias towards training samples that are similar to the piece of text being analysed
- a bootstrapping procedure is introduced to allow maximum entropy models to collect new, valuable information from unlabelled text

Results in this thesis indicate that the maximum entropy model is a very strong approach that accomplishes levels of performance that are very hard to improve on. However, these results also suggest that these extensions of the baseline system could yield improvements, though some difficulties must be addressed and more research is needed to obtain more assertive conclusions.

This thesis has nonetheless provided important contributions: a novel approach to estimate the complexity of a named entity extraction task, a method for selecting the features to be used by the maximum entropy model from a large pool of features and a novel procedure to bootstrap maximum entropy models.

Contents

List of Figures

List of Tables

List of Algorithms

Declaration

All material contained in this thesis is the author's own.

José L. Jara V.

Chapter 1

Introduction

1.1 Information extraction

Information Extraction (IE) is a relatively new discipline within the wider field of Natural Language Processing (NLP). In general, information extraction can be understood as any process that selects, extracts and combines data from text in natural language to produce structured information.

This discipline has emerged because there was a confluence of the necessity for automatically processing information that exists only in natural language form, and the ability to have a rough understanding of texts by using the current NLP technology (Grishman 1997).

Although some IE systems are starting their commercial life, the technology is not completely mature. There is a universal consensus that IE will be of great significance to companies of all kinds, especially those that make intensive use of information such as government institutions and finance enterprises (Cardie 1997, Grishman 1997).

Information extraction is quite intuitive for humans. Consider the following example text from the Management Succession Domain, as used in (MUC 1995):

> Topologix Inc. announced that Donald E. Martella, formerly vice president, operation, was named president and chief executive officer of this maker of parallel processing subsystems. He succeeds Jack Harper, a company founder who was named chairman.

The task for this domain is to identify succession events contained within the text and represent them in structures that contain four pieces of information: the person who is taking the new position, the person who is leaving the position, the position title, and the organisation where the succession is happening. For the example text above, adapted from Soderland, Fisher and Lehnert (1997), the resulting *answer template* would be:

	Event 1	Event 2	Event 3
Person_In	Donald E. Martella		Jack Harper
Person_Out	Jack Harper	Donald E. Martella	
Position	president and chief executive officer	vice president, operations	chairman
Organisation	Topologix Inc.	Topologix Inc.	Topologix Inc.

However, this task is very complex from the NLP point of view. Firstly, any IE system must apply a set of pre-processing tools to the text so that sentences, words, subjects, verbs, objects and ultimately clauses can be identified. Then, it must recognise names of people, organisations and positions in the text. Then it must recognise relations among these named entities suggested by expressions like formerly and was named. Finally, further processing is required to trim away odd words, to relate events reported in different sentences and to instantiate generic references, which would involve a high level of *coreference resolution* which is beyond the state-of-the-art (Cardie 1997).

The most important events in IE history were the ARPA-sponsored Message Understanding Conferences (MUCs) held between 1987 and 1997 (see for example (Lehnert, Cardie, Fisher, McCarthy, Riloff and Soderland 1994, McCarthy 1996, Wilks 1997)). These conferences introduced several technological challenges and a rigorous evaluation for the participants, providing the right environment for the development and evolution of a wide spectrum of approaches to information extraction, from traditional NLP — i.e. full syntactic, semantic and discourse analyses— to keyword matching with little or no linguistic analysis (Cowie and Lehnert 1996).

By the latest MUCs, there was a clear generic architecture and virtually all major participant systems shared —in some form— the same modules (Hobbs 1993, Cardie 1997). This convergence happened because researchers faced the challenges imposed by the MUCs in similar ways: exploiting the power of shallow parsing — rather than insisting on a full syntactic analysis, using shallow knowledge such as gazetteers and small hierarchy lexicons, using the key answers for deriving more shallow knowledge and using the target corpora for tuning some modules in the system (Cowie and Lehnert 1996, Cardie 1997, Wilks 1997).

This last two steps above have resulted in a shift of information extraction technology towards *empirical methods* —also called corpus-based methods— which address the current problems in IE systems — and NLP in general: accuracy, portability and knowledge acquisition (Cardie 1997). However, the performance of these systems remains poor. For example, in the last MUC the best corpus-based system could extract just over half the events reported in the test corpus (MUC 1998).

1.2 Named entity extraction

As mentioned above, named entity extraction (NEE) —the main problem with which this thesis is concerned— is a subtask of IE and much simpler than the general extraction procedure. In few words, NEE can be defined as the identification of expressions in free text which are "unique identifiers" of entities and their classification as instances of a finite set of categories. The first subtask is called named entity recognition (NER) and the second one named entity classification (NEC). However, it is very common in the literature to use named entity recognition for the whole process instead of named entity extraction. An attempt at keeping this distinction in this document will be made.

There is a general consensus that named entity recognition is not only relevant for information extraction, but also an important subtask for many other natural language engineering applications, such as information retrieval, question/answer and machine translation systems.

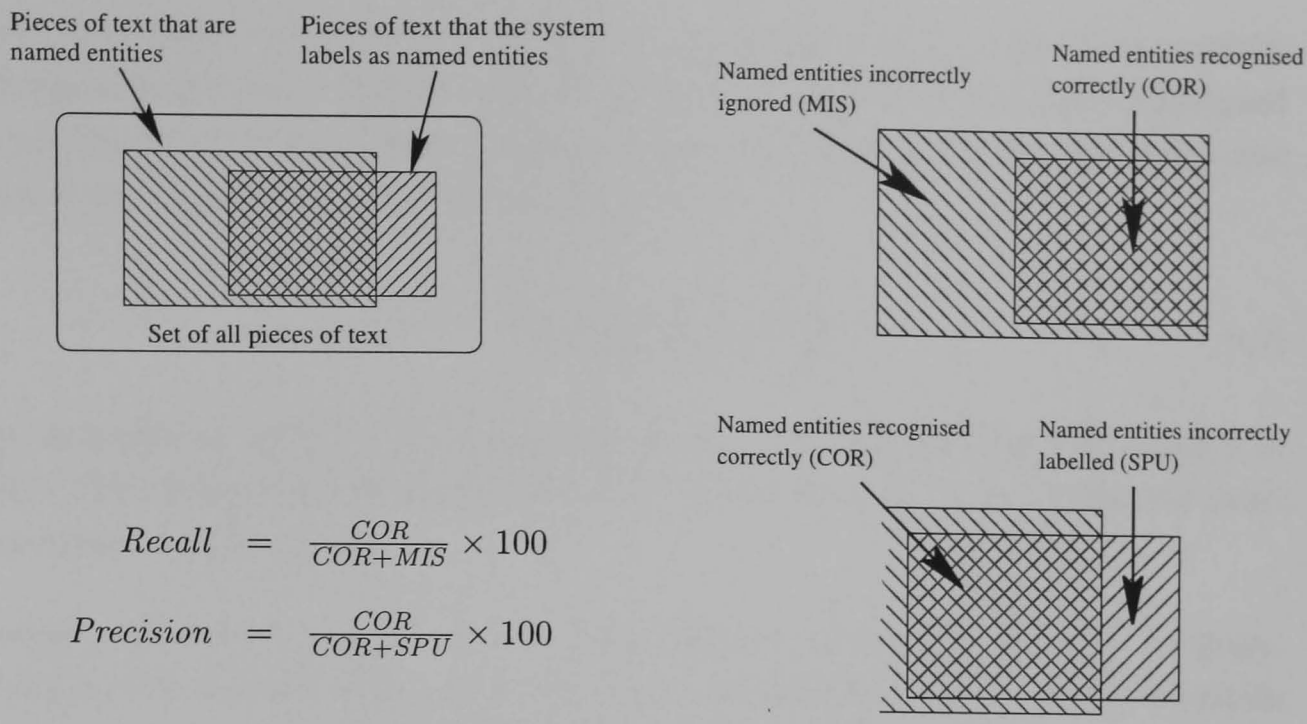
The latest Message Understanding Conferences (MUC) included a track for named entity recognition, which has become a sort of implicit universal definition for this task. The input for an NEE system is a text in free form such as the following:

> Llennel Evangelista, a spokesman for Intelsat, a global satellite consortium based in Washington, said the accident occurred at 2 p.m. EST Wednesday, or early Thursday morning at the Xichang launch site in Sichuan Province in southwestern China.

The system is expected to produce a new version of the input text in which named entities are marked with SGML tags according to their class. For the input above, the following is the required output for the MUC competitions, which considered seven categories: organisations, people, location, dates, time, money and percentages.

> <PERSON>**Llennel Evangelista**</PERSON>, a spokesman for <ORGANISATION>**Intelsat**</ORGANISATION>, a global satellite consortium based in <LOCATION>**Washington**</LOCATION>, said the accident occurred at <TIME>**2 p.m. EST**</TIME> <DATE>**Wednesday**</DATE>, or <TIME>**early Thursday morning**</TIME> at the <LOCATION>**Xichang**</LOCATION> launch site in <LOCATION>**Sichuan Province**</LOCATION> in southwestern <LOCATION>**China**</LOCATION>.

It is clear that this output text is much easier to process by a part-of-speech tagger or a parser, and by higher-level NLP tasks. For example it is clear that for an Internet retrieval engine or a question/answer machine, the latter text would help in a more accurate and faster answer to the query *where is Sichuan?* than the former text.



$$Recall = \frac{COR}{COR+MIS} \times 100$$

$$Precision = \frac{COR}{COR+SPU} \times 100$$

Figure 1.1: Definition of the evaluation metrics: recall and precision, in terms of correct named entities (COR), missed named entities (MIS) and spurious named entities (SPU).

MUC competitions also gave a universally accepted method for evaluating the performance of systems which attempt named entity extraction. Considering the *accuracy* of the system, that is the percentage of pieces of text labelled correctly, is not an appropriate measure. For example, suppose that there are 10,000 phrases in a text of which 100 are named entity expressions. Suppose now that a named entity extractor marks 80 instances of which 60 are correct. Then, the system's accuracy would be 99.4%, which intuitively seems wrong.

Therefore, named entity extractors are evaluated in terms of *recall* and *precision*. Recall is the ratio of the number of named entities that the system recognises correctly to the total number of named entities in the text. Precision is the proportion of correct named entities out of the pieces of text that the system labels as named entities. In the example, the system's recall is 60% (60 out of 100) and the system's precision is 75% (60 out of 80). Clearly these measures are better indicators for the performance of the system.

Figure 1.1 presents a graphical definition and the mathematical definitions of these metrics, which are based on the number of named entities correctly identified by the system (COR), the number of named entities that the system misses (MIS) and the number of spurious named entities in the system's output (SPU) (Chinchor 1998b).

Although every system tries maximising both recall and precision, soon enough it was evident that this was hard because a system that tries to maximise precision tends to mark only entities which it knows for certain and therefore it misses those for which it is not very sure. The vice versa effect is observed when the systems tries to maximise

recall. As it is not very obvious which is more important, MUC competitions opted for defining a third index called *F-score* or *F-measure* —borrowed from metrics designed by van Rijsbergen (1979) for information retrieval— which is the weighted harmonic mean of recall and precision, following the formula

$$F(\beta) = \frac{(\beta^2 + 1) \cdot Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} \tag{1.1}$$

The parameter β controls the relative importance given to precision with respect to recall. Throughout this thesis, the value $\beta = 1$ will be used, which assigns the same importance to both indices.

Recently, this named entity extraction task has been extended in the ACE Program, which has been defined as "a program to develop technology to extract and characterise meaning from human language" (Sheffield NLP Group 2005). This program includes Entity Detection and Tracking (EDT) tasks that are similar to the MUC NE task, but in which text sources are of varying types and quality. In addition, more semantic and more fine-grain information of named entities is required to fill the answer keys. These characteristics make the ACE tasks more challenging than the MUC task and are contrived to lead the development of extraction technology to support automatic processing of text data (LDC 2005).

1.3 Difficulties in NEE

As in any NLP task, most difficulties in named entity extraction are generated by the intrinsic ambiguity of natural languages and the amount of knowledge required to solve this ambiguity.

Firstly, NLP tasks earlier in the pipeline need to solve some of this ambiguity and determine —for example— sentence boundaries, token boundaries and their part-of-speech tag. This will inevitably introduce errors in the input to the NEE system.

Secondly, proper names present the same structural ambiguities as common nouns (Wacholder, Ravin and Choi 1997): compare *Midwest Center for Computer Research* (a single name) versus *Carnegie Hall for Irwin Berlin* (two names); *Victoria and Albert Museum* (a single name) versus *IBM and Bell Laboratories* (two names); *Donoghue's Money Fund Report* (a single name) versus *Israel's Shimon Peres* (two names).

Thirdly, proper names also display semantic ambiguity, as a word —or sequence of words— can be a name for entities of different nature. This is closely related to word-sense disambiguation, another NLP task which could benefit from NEE. In the example

text above, the word *Washington* might be seen with (at least) four senses: George Washington (the person), U.S.S. Washington (the ship), Washington D.C. (the city) and the State of Washington. This semantic ambiguity is quite common among proper names because places are named after famous people and organisations are named after their owners or locations (Wacholder et al. 1997).

Fourthly, proper names share ambiguity with common nouns: the word *china* refers to the high-quality ceramic ware whereas the word *China* refers to the People's Republic in Asia. This type of ambiguity is usually disambiguated through capitalisation in English, but this is not always the case. Consider the following beginnings of sentences: *New Coke drinkers* and *New Sears employees*. The word *New* belongs to the name in the first case, but it does not in the second case (Wacholder et al. 1997), a fact that can only be determined by world knowledge.

Finally, there is a further level of ambiguity introduced by what humans and particular applications may recognise as a name. For example, the MUC application required systems to make the distinction between organisations, people, locations, etc. and artifacts. Thus, in this case the sentences *Good news for Boeing shareholders* and *They will buy a new Boeing* are of different nature: the former contains an organisation entity whereas the later refers to an artifact. However, this differentiation may not be required by another application. In addition, the phrase *early Thursday morning* can be marked as a named entity of class time, as in the sample text above. A different person/application can perfectly consider this phrase as containing two named entities: the date *Thursday* and the time *morning*. Similarly, someone may consider the words *Sichuan Province* as the lexical form of a named entity of class location —again as in the text above— while another person may only consider for this the word *Sichuan*. In fact, experiments conducted during the MUC-7 competition have estimated that human annotators disagree in about 3% of the entities they recognised (MUC 1998).

When an NEE system is able to manage all this ambiguity reasonably well for a particular application, it normally handles a large amount of knowledge in the form of lexicons, gazetteers, grammars, patterns, ontologies, etc. This generates the second important drawback of this technology: named entity extractors show *poor portability*.

Named entities vary significantly in type and form across domains and the knowledge collected for one of them might become much less useful for another domain. Therefore, this knowledge requires to be adapted for each application, a process which is generally time-consuming and error prone. Moreover, this adaptation might sometimes be impossible due to the lack of experts.

1.4 Main Goal

The main goal of this thesis —following the discussion above— is to explore methods to capture the ambiguity of named entity extraction tasks, so that the extraction can be done with a relatively good level of performance, but at the same time maintaining a reasonable degree of portability.

There are several issues that these methods must deal with. First, it is indispensable to find an appropriate approach to capture and represent the essence of the extraction task, so that a person does not have to expend large amounts of time doing this. Secondly, it is necessary to provide this approach with the linguistic and world knowledge required for the interpretation of text, a difficult assignment that is specially not solved for general domains. Finally, some mechanism for tuning this knowledge to the specific task and domain, with little or no human support, must be found. This should include the management of exceptions, that is pieces of text which are often considered named entities by the task but in some occasions they are not considered so (or vice versa), as with the above example for the word *Boeing* on the MUC-7 task.

These methods should make possible the implementation of an NEE system that is able to perform *generic named entity extraction* which is not designed for any particular task or domain.

A system with these characteristics could be applied to new domains more quickly and without needing the many hours of work by experts, on both linguistics and the target domain, that traditional approaches normally require.

In chapter 2, these ideas are developed further by analysing techniques reported in the literature on how they could contribute towards more portable NEE systems. Several bases for the research presented here are drawn from this review and three hypotheses about how generic named entity extraction should be approached are proposed.

1.5 Document structure

Chapter 2 presents a review of previous work on named entity extraction. This review is used to introduce the bases for this research, as well as the hypotheses which will be assessed. In the second part of the chapter, the main approach used in this thesis, namely the maximum entropy framework, is explained in detail.

Following the review of previous NEE systems, two baseline systems are selected and evaluated in chapter 3. This chapter also expounds a detailed analysis of the named entities contained in the corpora used in this work.

Chapter 4 assesses the hypothesis that more linguistic information can be helpful in extracting named entities. This is done by extending one of the baseline systems, which is allowed to use domain-independent features derived from a shallow parser and a general lexical resource.

Chapter 5 discusses the use of a memory-based approach to avoid overlooking exceptions in the recognition of named entities and managing the lack of abundant training data.

In chapter 6, a study of how bootstrapping techniques may help the adaptation of the proposed named entity extractor to new domains is presented.

Finally, several conclusions drawn from this work are presented in chapter 7. This chapter also proposes future lines of research that might enrich the knowledge collected by this thesis about the extraction of named entities.

In addition to the main text, this document also includes a set of appendices which provide more detailed information on the analyses and implementations discussed in the chapters above. All these appendices are correspondingly cited in the text.

Chapter 2

Previous work and hypotheses

This chapter discusses previous approaches to solving named entity extraction. This review is used to introduce the bases that guide this research as well as the hypotheses that this thesis aims to assess.

2.1 Initial approaches

The initial attempts made to solve named entity extraction followed the typical approaches in natural language processing at the time, that is building systems based mainly on regular expression and dictionaries (also called gazetteers). In the following paragraphs, three examples which in one way or another characterise the evolution of these types of approaches are examined.

Fisher, Soderland, McCarthy, Feng and Lehnert (1995) presented a system at MUC-6 competition, in which input texts were submitted serially to four *string specialists* that recognised money/dates/percentages, organisations, people and location entities respectively. These specialists were hand-coded pattern matching routines applied serially in the order given above and each component could claim strings in a non-negotiable manner. The organisation, people and location specialists relied on dictionaries for the recognition of these named entities.

Fisher et al. (1995) concluded —after the MUC formal evaluation— that the organisation dictionary used in the competition was weak, and as a consequence the recall obtained on this kind of entity was poor, as well as affecting the performance on person and location names because missed organisations were normally claimed by one of the specialist applied later.

The LaSIE system (Gaizauskas, Wakao, Humphreys, Cunningham and Wilks 1995) and its descendant VIE (Humphreys, Gaizauskas, Cunningham and Azzam 1998) also rely

on hand-coded rules, list of words and gazetteers. Nonetheless, they follow a cleverer approach. For instance, in addition to a gazetteer of 2,600 organisation names, they also use lists of *trigger words* such as company designators (e.g. Co., Ltd., PLC), words that usually occur at the beginning or end of organisation names (e.g. Federal, International) and words which can be use alone as nouns to refer to organisations (e.g. Association, Agency, Ministry, etc.).

Using these resources in combination as a cascade set of finite-state recognisers, LaSIE and VIE have better chances of detecting the organisations mentioned in free text. Similar resources were used for the other named entity types.

The first people to seriously question the utility of gazetteers were Mikheev, Grover and Moens (1998). They observed that some NEE systems did not degrade much when their gazetteers were significantly reduced in size, and that by adding a few especially selected names, a dramatic improvement could be obtained (Mikheev, Moens and Grover 1999, Krupka and Hausman 1998).

Mikheev, Grover and Moens (1999) discussed the problems of using NEE systems relying almost exclusively on looking up proper nouns in gazetteers, which may lead to this behaviour:

- the availability of large and general gazetteers —specially for different languages— is very limited, a fact which has been described as a bottleneck in the development of NE extractors (Cucchiarelli, Luzi and Velardi 1998)
- even if they were available, they would have to be very large —it is estimated that there are 1.5 million surnames just in the U.S.— and searching in them might be infeasible
- it is not easy to keep gazetteers up-to-date; for example a list of all companies in the European Union today would be enormous, and obsolete tomorrow as companies are being created all the time
- named entities occur in variations; for example *The Royal Bank of Scotland plc*, *The Royal Bank*, *The Royal plc* and simply *The Royal*, all refer to the same organisation and gazetteers should contain all these variations
- gazetteers do not solve the problem of overlaps between lists; for example, the word *Washington* could easily be found in a list of person names, organisation names or location names
- language ambiguity and the inclusion of common words in names make the task even more difficult; this is specially true when conjunctions are involved; for example, it is very hard to recognise the organisation name in the sentence *Daily and Partners lost their court case*

Algorithm 2.1: The LTG named entity extraction algorithm. Adapted from Mikheev et al. (1998).

Input: a document C
Output: a new version of the document C' in which recognised named entities are annotated

1: Apply *sure-fire rules* based on reliable phrasal and contextual designators
(For example the rule Mr Xxxx+ is a person)
2: Apply a probabilistic partial match of the identified entities
(This allows the system to recognise variations in the name of entities; for example, if Mr John Adams has been recognised as a possible person's name, then Mr Adams and J. Adams will be also marked as possible instances of person's names)
3: Apply the rules again but using more relaxed contextual constraints by using a grammar of names
(This step also includes conjunctions resolution and the recognition of known named entities from gazetteers)
4: Apply a second probabilistic partial match supported by a maximum entropy model
5: Apply partial matching and check against the maximum entropy model possible named entities in titles
(This is done at the end because headlines provide little guidance for recognising names, as they are normally in capital letters and contain almost no contextual clues)

Mikheev et al. (1998) presented the LTG system which makes use of both *internal* (phrasal) and *external* (contextual) evidence for recognising an entity and its class. This terminology was introduced by McDonald (1996) and has been followed in most named entity extraction approaches since then (Wakao, Gaizauskas and Wilks 1996, Wacholder et al. 1997, Cucchiarelli et al. 1998, Zhou and Su 2002, to mention some). The idea is very simple: consider the string *Adam Kluver*; it can be seen that this string has an internal phrase structure (e.g. both words are capitalised and *Adam* is a common first name) which suggests that this is a person name. However, somewhere in the text is likely to exist some contextual information that should make clear what type of named entity it is. For example, the string could be found in the phrase *Mr. Adam Kluver* which would confirm that this string is a person name, or it could be seen in the phrase *Adam Kluver Ltd.* which would indicate that this string is actually an organisation name. Following this idea, the LTG system only makes the decision of the class for a given entity when clear contextual information that supports such a decision is found.

In addition, the LTG system assumes that —for example— once the string *Adam Kluver* is identified as an organisation name, then any other less clear occurrence of this string will refer to the same entity. If this string is also used for referring to an entity of another type, then the LTG system assumes that clear contextual material would be found to determine this new meaning.

Algorithm 2.1 presents the LTG extraction algorithm. This system reached an F-score over 93% in the MUC-7 evaluation, the highest performance registered for the competition (Mikheev et al. 1998). Moreover, Mikheev, Moens and Grover (1999) conducted a further evaluation of LTG which showed that even without using gazetteers, the algorithm obtains levels of accuracy comparable to many other NEE systems competing

in MUC-7.

Although the LTG system performs remarkably well, the rules and grammars it uses were hand-coded for the MUC-7 application. Therefore, these resources need to be re-coded to apply this system on a new domain, which seriously compromises the portability of the system.

2.2 Machine learning approaches

The portability limitation mentioned in section 2.1 has attracted the attention of researchers towards machine learning techniques which can learn how to use both internal and contextual information for identifying named entities.

The Message Understanding Conferences provided the appropriate environment for the development of this type of NEE system. More recently, the shared tasks of the Conferences on Computational Natural Language Learning (CoNLLs) have also created a competitive environment which has boosted the research in this area.

The latest MUCs defined the standard way in which named entity extractors should be developed. They release three corpora for each task: a training corpus which must be used to develop the system, a dryrun test corpus so that systems can be tested during the development phase, and a blind test corpus on which systems were formally evaluated (MUC 1995, MUC 1998).

The MUC task involved the discrimination of seven named entity types —namely person, organisation and location names; date and time expressions; and money and percentage expressions— from artifacts and normal text (Chinchor 1998a).

In MUC-7, a further problem was introduced: the training and dryrun texts were selected from an aircraft accident domain, whereas the formal test corpus was a collection of articles reporting the launching of aircrafts. This aimed to encourage the development of NEE systems that could be easily adapted across different domains. Many machine learning approaches were presented for these competitions, some of which are described in the following paragraphs.

For example, Aberdeen, Burger, Day, Hirschman, Robinson and Vilain (1995) presented the Alembic system, which uses a maximum error-reduction learning algorithm to construct a list of transformation rules (based on Brill (1995)); this rules obtained an F-score of 85%[1].

[1]Although performance measures are given for these systems, it must be kept in mind that such performance might be estimated on different testing corpora and by different scoring programs. In addition, each system normally prepared its own extra training material as well as their own preprocessing programs. Therefore, comparing systems based on these measures is unrealistic and they must be considered as referential only.

Cowie (1995) developed a system called AutoLearn, which constructs decision trees using the ID3 algorithm (Quinlan 1983) for detecting the start and end points of named entities; its performance was not very impressive: F-score 64%.

Bennett, Aone and Lovell (1997) presented RoboTag, which also uses decision trees to classify words as being potential start/end of a named entity. However, this system uses C4.5 (Quinlan 1993) for the induction and external lexical resources such as gazetteers; RoboTag performed better: F-score 83.6%. Sekine (1998) presented a variation of this approach for a Japanese task, which reached an F-score of 85%. Paliouras, Karkaletsis, Petasis and Spyropoulos (2000) also evaluated C4.5 and external resources on the MUC corpora —though they were not in the competition— and obtained a performance around F-score 83%.

Bikel, Miller, Schwartz and Weischedel (1997) developed Nymble, a system that employs a hidden Markov model and a bigram language model for the task; this system performed remarkably well for its simplicity: its F-score was 93%. This approach will be discussed in more detail in the following chapter.

Borthwick, Sterling, Agichtein and Grishman (1998) use maximum entropy models for predicting named entities in a system they named MENE. This system scored an F-measure of 92%. This system will also be studied in more detail in the next chapter.

The CoNNL shared tasks introduced new challenges to the named entity extraction task, as the organisers were interested in language independent NEE systems that could use additional non-annotated training data (Tjong Kim Sang 2002b, Tjong Kim Sang and De Meulder 2003). Named entity classes were reduced to four types: person, location, organisation and miscellaneous names.

A wide range of methods and meta-methods from machine learning and natural language processing were presented, including support vector machines, transformation-based lists, learners cascade, boosting, Markov models and hidden Markov models, maximum entropy models, character n-grams and tries, stacking, decision trees, voting, Winnow and SNoW, etc. plus some combinations of these methods. Examples of systems presented in these conferences are discussed in the following paragraphs.

Carreras, Màrquez and Padró (2002) presented an NEE system in which recognition (NER) and classification (NEC) were performed sequentially and independently. Both modules used binary AdaBoost classifiers to combine fixed-length decision trees. This system scored best in two different languages: F-measure 81.39% for Spanish and 77.05% for Dutch. They also reported that the pipeline scheme caused propagation of errors and that additional knowledge sources —such as gazetteers and a list of trigger words—

yield an improvement of just 2% in the NEC system, and were of no utility in a second step to obtain the final labels.

Cucerzan and Yarowsky (2002) developed a system which used both word internal and contextual clues as relatively independent evidence sources that drive a bootstrapping algorithm from initial seed names. Internal clues refer to morphological structure such as prefixes and suffixes, which is automatically learnt during the bootstrapping process. Contextual clues refer to patterns (such as Mr., in, mayor of on the left) which are crucial for names that do not follow a typical morphological pattern, are of foreign origin or polysemous. Both types of information are modelled as four smoothed tries: two for context (left and right) and two for internal morphological patterns (prefix and suffix tries). This system scored third and fifth on Spanish (77.15%) and Dutch (72.31%) respectively.

Florian, Ittycheriah, Jing and Zhang (2003) presented an NEE system which combines four classifiers based on different machine learning approaches: the first system used Robust Risk Minimisation (RRM), which is based on Zhang, Damerau and Johnson (2002); the second one relies on a maximum entropy model, similar to MENE; the third classifier utilises transformation-based learning; and the fourth system is a hidden Markov model classifier, similar to Nymble. They reported different performance levels with different ways of combining these four classifiers, which show that an RRM approach yielded better results. In the formal test, this system obtained the highest overall performance on both English and German with F-scores 88.76% and 72.41% respectively, which outperformed the best individual classifier by 17-21% for the English task and less significantly for the German task.

Zhang and Johnson (2003) also presented a combination of classifiers in which the basic units are characters and character n-grams, instead of words and word phrases. The first model is a character-level hidden Markov model and the second one is a maximum entropy model. This system was ranked third on English and second on German, obtaining 86.31% and 71.90% respectively. They reported that when n-grams are not used, their system shows a 25% error increment.

2.3 Bases and hypotheses

Following the discussion above, the first basis in this thesis is that a portable named entity extractor must not use manually-built rules for the task. Hand-coded rules are time-consuming and although there may be attempts at making the modification and addition of rules as simple as possible, this facilitation is normally oriented to experts at both linguistics and the architecture of the system, or in the best case to experts in the extraction task, which automatically restricts the portability of such a system.

Therefore, this thesis will focus on machine learning techniques which should allow the acquisition of rules and knowledge with little or no human intervention.

By a simple examination of the machine learning methods employed by the systems described in section 2.2, it is easy to see that statistical learning techniques are extensively used and quite successful. For instance, Nymble and MENE both utilise statistical methods and both report a performance over 90% for the MUC task.

This is not exclusive to named entity extraction tasks. Most NLP problems can be seen as random processes for which it is necessary to find the probabilistic distributions that model their behaviour. Many successful applications of these methods has been reported, accomplishing levels of performance that are very hard to improve on (Mikheev 1998).

These observations lead to the second basis for this research: modelling NLP problems, of which named entity extraction is just an instance, as statistical classification problems has proved to be a successful approach.

Among many other statistical methods, both hidden Markov models and maximum entropy models are popular choices. For example, out of the sixteen systems competing at the CoNLL-2003 shared task, three systems used maximum entropy models, two utilised hidden Markov models, and two other systems employed a combination of these techniques (Daelemans and Osborne 2003).

However, maximum entropy models present some advantages over hidden Markov models (Mikheev 1998):

1. Hidden Markov models —and generative statistical methods in general— assume that the different pieces of contextual information are independent, and the model engineer must take care to avoid the inclusion of overlapping features. These simplifications are not necessary for maximum entropy models which can deal with both overlapping feature segments and overlapping feature functionality.

2. The above property allows maximum entropy models to use more information and from different sources, resulting in more complex models which have improved the performance of classifiers in a number of applications (for examples, see Rosenfeld (1996) and Ratnaparkhi (1996)).

3. In many cases, hidden Markov models can be moved from one domain into another by re-computing the probabilities associated to the features included in the model. However, if these domains present significant differences, the knowledge engineer must build a completely new model. On the other hand, maximum entropy models normally need to re-estimate their weights to capture a shift in the domain, but

less participation of an expert is needed if the available pool of features is general enough.

Since this work started in 2000, there have been two important conferences on named entity extraction, namely the shared tasks of CoNLL-2002 and CoNLL-2003, which have provided more evidence in favour of maximum entropy models.

In the CoNLL-2002, Malouf (2002) applied three different statistical approaches to the shared NEE task: a baseline statistical model, a hidden Markov model and a maximum entropy model. He reports that even using the same information, the maximum entropy approach widely outperforms the other two models. Moreover, when taking advantage of a maximum entropy model's ability to use more complex information and extra features were added, it obtained a 67% increase in overall performance with respect to the hidden Markov model.

Results of the CoNLL-2003 are even more relevant: the top two systems in the German task and the top three systems in the English task utilised maximum entropy models, in isolation or in combination with other approaches, which would confirm that "Maximum Entropy Models seem to be a good choice for this kind of task" (Tjong Kim Sang and De Meulder 2003).

These observations lead to the third basis of this thesis: maximum entropy models perform well on named entity extraction tasks and they also provide natural ways of introducing new knowledge to guide the extraction procedure, which makes them a good approach for generic named entity extraction.

NEE systems based on maximum entropy models have used a wide variety of features extracted from a *context window* of words around a *focus word*, whose named entity class must be determined. Among the most common features are:

- lexical features, which are the strings —normally in a case insensitive mode— of the tokens under consideration (Borthwick 1999, Malouf 2002, Bender, Och and Ney 2003, Chieu and Ng 2003, Curran and Clark 2003b, Florian et al. 2003, Klein, Smarr, Nguyen and Manning 2003)
- orthographic features, which provide information about the form of these tokens, such as the type of lexeme (e.g. number, word, symbol) and capitalisation properties (e.g. lowercase, uppercase, capitalised) (Borthwick 1999, Malouf 2002, Bender et al. 2003, Chieu and Ng 2003, Curran and Clark 2003b, Florian et al. 2003)

- ▷ section features, which help discriminate between sections in the document (e.g. headlines, text) in which the context window is located (Borthwick 1999, Malouf 2002, Chieu and Ng 2003)
- ▷ dictionary features, which indicate whether a word is contained in external dictionaries of names (gazetteers) (Borthwick 1999, Malouf 2002, Bender et al. 2003, Chieu and Ng 2003, Curran and Clark 2003b, Florian et al. 2003, Klein et al. 2003)
- ▷ PoS features, which give information on the part-of-speech that a word is playing in the sentence (Curran and Clark 2003b, Florian et al. 2003, Klein et al. 2003)
- ▷ morphological features, which supply information about prefixes and suffixes of tokens (Bender et al. 2003, Florian et al. 2003, Klein et al. 2003)

All these types of features are lexically oriented, and besides Curran and Clark (2003b) and Florian et al. (2003), who included chunk features, there have been no attempts to include linguistically richer information into maximum entropy models for named entity extraction. The reason for this omission seems to be that getting rich linguistic features —such as sentence structure, phrasal heads and semantic relations— can be highly expensive and domain dependent, and it is not clear whether such information could be useful. Nonetheless, it would be an important contribution to verify these assumptions.

On the one hand, a good parser of general natural language text is still a matter of research, but obtaining shallow phrase structure —such as noun and verb chunks, and their head words— is not very difficult nowadays and a few systems are starting to be available online. On the other hand, getting semantic and discourse-level information is much harder. Acceptable solutions for structural attachment, sense ambiguity, coreference and semantic relationships are very difficult to obtain, and only established research groups have been able to attempt to solve these problems for information extraction, after several years of dedicated work (Fisher et al. 1995, Wakao et al. 1996). Thus, building these kinds of tools is realistically beyond the scope of this thesis.

The NLP Group at the University of Sheffield is probably the research body which has paid most attention to these problems, resulting in the definition of a general architecture for text engineering (Cunningham 2000). Among their work, Gaizauskas and Humphreys (1997) evaluated the use of semantic networks for information extraction. Basically, they use a semantic network to have a *world model* prior to the processing of the text. This network consists of an ontology and an associated attribute knowledge base, which are manually built from the definition of the task. When a text is processed, this semantic network is populated with the classes and instances mentioned in the text, thus specialising the world model for that particular text. This specialised model allows Gaizauskas and Humphreys (1997) to perform an analysis at discourse level.

In the case of named entity extraction, the discourse model described above is utilised to resolve coreferences of names and to classified ambiguous names by looking into the semantic types of the arguments in certain relations (Wakao et al. 1996). For example, the existence of the ambiguous name *Ford* can be classified by identifying that it refers to the unambiguous name *Ford Motor Co.*, also found in the discourse model; and by identifying that the word *stock* is semantically related to organisations, the ambiguous name *Erickson* in the phrase *Erickson stocks* can be classified as an organisation name. Note however, that only NEC is performed and no new named entities are recognised at this stage.

Gaizauskas and Humphreys's (1997) approach is dominated by the view that more accurate information extraction systems cannot be obtained without attempting a deeper understanding of the text being processed. This thesis widely shares this view. However, getting this understanding from manually built resources —as Gaizauskas and Humphreys (1997) have done— compromises portability and, therefore, this thesis will avoid resorting to this solution.

Nevertheless, the idea of introducing ontologies to obtain certain amounts of semantic information should not be discarded. Guarino (1997) clearly identifies a fundamental role for ontological aspects for information extraction: the semantic matching between the terms used to define the task and those appearing in the text. Thus, named entity extraction can be seen as identifying semantic matches between pieces of text and entity classes. Therefore, this task requires that the meanings of both the classes and the names occurring in the text are clear in order to determine whether they match or not. Unfortunately, the ontologies currently in use are normally built ad-hoc (Guarino 1997) for both the domain and the task.

However, there exists a general purpose ontology which could be used: WordNet® (Miller 1995). WordNet is a lexical reference system which organises English nouns, verbs, adjectives and adverbs into synonym sets, each representing one underlying lexical concept and their relations (Fellbaum 1998). This general resource might provide an NEE system with "semantic" understanding of the text being processed as well as more clues for identifying unseen named entities, and without affecting the portability of the system as the knowledge it contains is not specific to any particular application, but for the English language in general.

In fact, Gaizauskas and Humphreys (1997) made use of WordNet in an attempt to produce a more general world model which could help their system to resolve more coreferences. Although they conducted a very small experiment, they found that the availability of more semantic classes in the extended ontology had little effect on the number of anaphoras that are correctly identified. In this attempt, they also concluded that because WordNet uses a different synset entry for each possible sense of the lexemes

in the database, the problem of word sense disambiguation must be addressed. Their solution was to manually select only one sense per word, based on a small set of training documents.

Thus, in order to use WordNet as a source of semantic information for named entity extraction, a better solution —with respect to the portability of the system— should ideally be found. Although the construction of a general purpose word sense disambiguator is generating much research today (Kilgarriff and Rosenzweig 2000), it is still a medium-term goal and, therefore, beyond the scope of this thesis.

Nonetheless, it can be argued here than using WordNet ontology, even without performing any disambiguation, can be useful for the NEE system proposed here. This is so because a *trigger word* —as proposed in Wakao et al. (1996)— identified in the training documents can be extended to examples which are not seen in the corpus. For example, suppose that the word *chairman* is frequently seen in collocation with a person name. WordNet would inform the system that it should also consider its synonym *director*. In this way, the maximum entropy model would give to a noun phrase which starts with the word *director* a higher probability of containing a person name, even if the pattern *director <person name>* was not seen during training, just because it hits the same WordNet synset as the word *chairman*. This intuition is explained in more detail in section 4.5.

All these last observations lead to the first hypothesis of this thesis:

Hypothesis 1

> General, domain independent linguistic knowledge —such as the semantic information provided by WordNet— is useful for extracting named entities.

The introduction of shallow parsing and general ontologies should not affect the portability of the system. Moreover, projects like EuroWordNet (Vossen 1998), which aims to develop WordNet-like semantic networks for several European languages, might provide the necessary resources to allow the approach to move to other languages.

The second hypothesis of this thesis follows from the observation that statistical learning methods, such as maximum entropy models, heavily rely on the frequencies of the events being learnt. In fact, this is a characteristic of all machine learning methods that build general hypotheses: they try to capture the general and tend to overlook infrequent events.

Daelemans, van de Bosch and Savrel (1999) showed that not considering infrequent training examples —exceptions in their terms— can negatively affect the performance of learners when a natural language task is involved. More specifically, they provide empirical evidence that by keeping training exceptions, a memory-based learner improves its performance up to a level which allows this approach to outperform a decision tree learner.

This conclusion might be perfectly valid for named entity extraction. For example, the token *Clinton* is seen in the MUC-7 training corpus 52 times: 51 times as a person name and once as a location name. Thus, the probability that the token *Clinton* is reporting a person name is 0.98, which will likely dominate the decisions that the maximum entropy model makes for this example. This cannot be considered a mistake since statistical methods assume that the same distribution observed in the training data will be found in the decoding data.

There have been some attempts at overcoming this problem. A popular choice has been applying *Boosting* (Schapire 1990). Boosting is a general method to produce very accurate classifiers by combining rough and moderately inaccurate classifiers. Perhaps the most used version of this technique is the AdaBoost algorithm —short for Adaptive Boosting— which was introduced by Freund and Schapire (1996) with a strong theoretical framework based on PAC-learning. AdaBoost calls a weak learning algorithm repeatedly —though they may be different learning methods— in a series of rounds, in which a distribution of weights is defined over the set of training examples. Initially, every instance has the same weight, but on each round, the algorithm increases the weights of misclassified examples, so that the weak classifier will try harder on these examples on the next round. Using the distribution of weights for the current round, the algorithm obtains a weak hypothesis, its error and its global importance in inverse proportion to that error. The final hypothesis is the weighted majority vote of all weak hypotheses obtained. In this way, AdaBoost is able to identify exceptions —outliers in Freund and Schapire's (1996) terms— which are mislabelled or inherently ambiguous and hard to classify.

This method has been successfully applied to several NLP tasks, such as part-of-speech tagging/PP attachment (Abney, Schapire and Singer 1999) and text categorisation (Escudero, Màrquez and Rigau 2000), and more recently to the CoNLL-2002 named entity extraction task in Carreras et al. (2002) and Wu, Ngai, Carpuat, Larsen and Yang (2002). These latter works are particularly successful: Carreras et al.'s (2002) system obtained the best scores in both Spanish and Dutch and Wu et al.'s (2002) was fourth for Spanish and second for Dutch.

Carreras et al. (2002) and Wu et al. (2002) both used very simple weak learners that obtained (very) shallow decision trees. It is not clear whether boosting would have

obtained the same results when a statistical method is used as the weak learner. For these kinds of methods, a common practice is to perform *boosting by re-sampling* (rather than boosting by re-weighting, as described above), in which a fixed number of training examples are chosen with replacement according to the distribution defined by their weights. But Freund and Schapire (1996) found that one of the requirements for obtaining a significant improvement in performance by boosting is that the weak learner must be sensitive to changes in the training examples, so that the hypotheses generated for the different training sets at each round are significantly different. This condition might be difficult to meet with maximum entropy models and it would need a careful re-sampling of the training examples with the explicit intention of producing different constraints from one round to the next. In fact, Park and Zhang (2002) showed that the effect of boosting is not significant when applied on a maximum entropy model for shallow parsing.

The approach proposed here is based on the combination of memory-based methods and maximum entropy models, so that the advantages of both types of techniques can be united.

Memory-based reasoning solves new problems by *adapting* solutions that were used to solve old problems (Burkhard 1998), but they do not learn a general hypothesis to be applied later on. Instead, they use a retrieval engine which utilises the concept of similarity to search among the past cases to obtain the most similar ones to a new case, or *query*, that needs to be solved (i.e. classified, in this task). This way of *reasoning* has many advantages: adaptation is not limited to any specific framework and therefore cases and solutions are not restricted in structure; the similarity measures are not fixed and frequencies can be considered but may not be the only factors; because adaptation for a query can normally be performed from few cases retrieved, huge amounts of training data are not essential; and the set of cases kept in memory can vary dynamically allowing the deletion of useless cases and the insertion of new cases.

The idea is basically not to build a general maximum entropy model to be applied to all decoding examples. Instead, a retrieve engine will be applied to obtain a set of training examples that are similar to each decoding example. Only then will the maximum entropy framework be utilised to adapt each set of retrieved training information in order to decide a classification for the query text.

The intuition behind this idea is that by *biasing* the maximum entropy model in favour of examples that are similar to the piece of text that needs to be classified, the model will be able to capture exceptions and the contexts in which unseen named entities appear in a better way. Chapter 5 gives more details of this expected effect. This discussion leads to the second hypothesis to be assessed in this thesis.

Hypothesis 2

> Biasing the maximum entropy model towards the training examples that are similar to the text under processing results in an increase in the performance of the model on exceptions and unseen named entities

The third hypothesis is related to the fact that for natural language applications, the annotation of training examples has been reported as a difficult, error-prone and time-consuming task (Cardie 1997). This is an important problem, even for NEE systems based on machine learning techniques that do not use hand-coded rules or gazetteers, as they normally require a large number of labelled examples to obtain reasonable levels of accuracy, limiting their portability across domains and languages.

There have been some attempts at overcoming this problem by obtaining new training material from *unlabelled* text, mainly motivated by the work of Yarowsky (1995) and Blum and Mitchell (1997), which showed that supervision could be significantly reduced by exploiting the natural redundancy in textual data. They introduced two different ways of performing *semi-supervised learning*, which the literature has generally called *bootstrapping* and *co-training*, though a theoretical connection between these approaches has been shown very recently (Abney 2004).

Semi-supervised learning works basically as an iterative process to *estimate* annotations for unlabelled data, whose final objective is to provide annotated training data to improve a learner. Yarowsky (1995) used his bootstrapping algorithm to solve word sense disambiguation, obtaining better performance from an initial small set of *seed* collocations than a completely-supervised learning approach. Riloff and Jones (1999) used a variation to extract a semantic lexicon (i.e. named entities) and extraction patterns for an IE task, just from a few seed words. At about the same time, Collins and Singer (1999) utilised co-training to induce rules for named entity classification from just seven seed rules.

However, the above approaches have in common that they employ rule learners. It is not clear whether bootstrapping would improve the performance of maximum entropy models, though Blum and Mitchell (1997) reported experiments with a statistical method, namely the naïve Bayes classifier, in which the semi-supervised version outperforms the supervised one.

Nevertheless, it would be an important contribution to explore ways of doing semi-supervised learning with maximum entropy models, which would increase the portability of an NEE system that uses this machine learning approach. This interest underlies the third hypothesis to be assessed in this thesis:

Hypothesis 3

> Bootstrapping techniques can help an NEE system based on maximum entropy models to be more portable by obtaining information which is valuable for the task from unlabelled text

However, recent studies by Clark, Curran and Osborne (2003) and Cui and Guthrie (2004) on bootstrapping maximum entropy models are indicating that they pose some challenges for semi-supervised learning. These studies will be discussed in detail in chapter 6.

The rest of this document is dedicated to the the evaluation of the hypotheses discussed in this section.

2.4 Maximum Entropy Models

2.4.1 NLP and classification

Many problems in natural language processing —such as part-of-speech tagging (Brill 1995), word-sense disambiguation (Gale, Church and Yarowsky 1992), propositional phrase attachment (Aberdeen et al. 1995) and chunking (Cardie, Daelemans, Nédellec and Tjong Kim Sang 2000)— have been formulated as classification problems and solved with a variety of methods.

In particular, NLP problems can be modelled as a statistical classification task in which the probability of a class y —from a set of classes Y— occurring with context x —from a space of contexts X— is estimated. Classes and contexts depend on the particular NLP problem being solved. For example, calculating the probability of a word w surrounded by words w_{-1}, w_{+1} of being a noun, a verb or none, i.e. $P(noun| < w_{-1}, w, w_{+1} >)$, $P(verb| < w_{-1}, w, w_{+1} >)$ and $P(none| < w_{-1}, w, w_{+1} >)$, a coarse part-of-speech tagging can be obtained. In this example, $\{noun, verb, none\}$ are the classes and $< w_{-1}, w, w_{+1} >$ for each word w constitute the contexts. In this way, it is expected that $P(noun| < the, model, is >)$ would be higher than both $P(verb| < the, model, is >)$ and $P(none| < the, model, is >)$. Similarly, $P(verb| < we, model, the >)$ should be higher than both $P(noun| < we, model, the >)$ and $P(none| < we, model, the >)$, and that the context $< model, the, data >$ would yield to a higher probability with the class *none* than for the other two classes.

There are two major problems with this kind of formulation. Firstly, a relatively large collection of annotated texts that might provide information about the occurrence of a class $y \in Y$ with contexts $x \in X$ is needed. Normally, this information is not enough for accurately estimating $p(y|x)$ for all possible pairs (y, x) due to sparseness (a good example of this problem can be found in Godfrey, Holliman and McDaniel (1999)). Secondly, the different pieces of information considered within each context are generally from different sources and some might be irrelevant, overlapping or probabilistically dependent. These difficulties are to some extent overcome by Maximum Entropy Models (MEMs).

In an MEM, problem-specific knowledge is represented as binary features[2] which test the presence of pieces of information in a context. In this way, virtually any kind of knowledge —even from different nature— can be introduced into the model (Ratnaparkhi 1998).

Once the set of features to be included by the model is decided, a general purpose iterative algorithm can be used to estimate the parameters of the model (see section 2.4.3.1). Therefore, modellers need only focus their efforts on determining what set of features to use and not how to use it (Ratnaparkhi 1998).

Features of an MEM do not need to be statistically independent or not overlapping. Borthwick (1999) shows that:

- ▷ an MEM that includes two features $f_1(x, y)$ and $f_2(x, y)$ so that $f_1(x, y) = 1 \Leftrightarrow f_2(x, y) = 1$, is equivalent to an MEM that includes just one of them; and
- ▷ an MEM that includes features $f_1(x, y)$, $f_2(x, y)$ and $f_3(x, y)$ so that two of them partition the other, i.e. $f_1(x, y) = 1 \Rightarrow \overline{f_2}(x, y) = 1$ and $f_2(x, y) = 1 \Rightarrow \overline{f_1}(x, y) = 1$ and $f_3(x, y) = 1 \Rightarrow f_1(x, y) = 1 \vee f_2(x, y) = 1$, is equivalent to an MEM that includes just two of these features

Moreover, if the set of features do not overlap then there is no need for an iterative algorithm and the probabilities can be estimated by a simple ratio of counts (Ratnaparkhi 1998). Thus the true value of MEMs is obtained when features that do not form a partition of the space of classes and contexts need to be combined robustly. This is also important because determining the larger set of partitioning features from the set of features that are subsets of a given feature f is $\mathcal{NP}$-Complete (Borthwick 1999). Furthermore, an MEM can account not only for overlapping segments in the set of training samples, but also for overlapping functionality among features which might not even be represented in terms of overlapping features (Mikheev 1998).

[2]Features do not need to be binary-valued, but using binary features only makes the estimation of the parameters easier (Della Pietra, Della Pietra and Lafferty 1997).

All these characteristics of MEMs significantly reduce the work of constructing sound probabilistic models for any task. Successful examples are many, for instance the work of Ratnaparkhi (1998), who applied the maximum entropy framework to sentence boundary detection, part-of-speech tagging, prepositional phrase attachment, natural language parsing and text categorisation. In all five problems, he obtained performance at or near the state of the art.

2.4.2 The Maximum Entropy Framework

The maximum entropy modelling framework is introduced here, adapted from (Della Pietra, Della Pietra and Lafferty 1995) and (Berger, Della Pietra and Della Pietra 1996). More details can be obtain in these two excellent publications.

Consider a random process which produces an output value y from a finite set of classes Y. The objective is to construct a stochastic model that accurately represents this process by estimating $p(y|x)$, that is the conditional probability that given a context x, the process will output y. The model provides a conditional probability distribution $\mathbf{p}(\mathbf{y}|\mathbf{x})$, in which the placeholders $\mathbf{x}$ and $\mathbf{y}$ are instantiated to specific contexts in X and classes in Y respectively. Let $\mathcal{P}$ be the set of all conditional probabilities. Then $\mathbf{p}(\mathbf{y}|\mathbf{x})$ is just one member of $\mathcal{P}$.

In order to build this model, a number of samples $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ are collected, which provides the model with information of the behaviour of the random process. These observations are normally called *training samples*.

The set of training samples is summarised in terms of its empirical distribution $\tilde{\mathbf{p}}$, defined by

$$\tilde{p}(x, y) \equiv \frac{count(x, y)}{N}$$

where the function $count(x, y)$ counts the number of times that the pair (x, y) occurs in the set of training samples.

The task can now be seen as building a statistical model of the random process which generated the training sample $\tilde{\mathbf{p}}(\mathbf{x}, \mathbf{y})$.

The concept of *features function* needs to be introduced here. A feature function or *feature* for short, is a binary-valued indicator which expresses a particular statistic of the set of training samples. Following the example in 2.4.1, the following indicators could be introduced:

$$f_1(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} = \textit{noun} \text{ and } w_{+1} \text{ is a form of the verb 'to be' in } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} = verb \text{ and } w_{-1} \text{ is a pronoun in } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

The training sample $(< the, model, is >, noun)$ will fire f_1, that is $f_1(< the, model, is >, noun) = 1$, but not f_2. Similarly, the sample $(< we, model, the >, verb)$ will fire f_2, but not f_1. The interest is focused on the expected value of a feature $\mathbf{f_i}$ with respect to the empirical distribution $\tilde{\mathbf{p}}(\mathbf{x},\mathbf{y})$. This value is given by equation 2.1.

$$\tilde{p}[\mathbf{f_i}] \equiv \sum_{x,y} \tilde{p}(x,y) f_i(x,y) \tag{2.1}$$

When a particular statistic is considered relevant for describing the random process, it is included into the model by constraining the expected value that the model associates to the corresponding feature. This expected value is calculated as

$$p[\mathbf{f_i}] \equiv \sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y) \tag{2.2}$$

where $p(y|x)$ corresponds to the conditional probability of obtaining y given a context x estimated by the model and $\tilde{p}(x)$ is the empirical probability of seeing context x in the training samples.

The obvious constraint is that the model should agree with the set of training samples on how often the output of the random process exhibits a given feature $\mathbf{f_i}$. This is done by requiring $p[\mathbf{f_i}] = \tilde{p}[\mathbf{f_i}]$ which yields to the more explicit equation

$$\sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y) = \sum_{x,y} \tilde{p}(x,y) f_i(x,y)$$

which is known as a *constraint equation* or simply a *constraint.*

Thus statistical phenomena on the training samples that are considered important can be represented —through $\tilde{p}[\mathbf{f_i}]$— and the model for the random process is requested to exhibit these phenomena by imposing constraints. Suppose that n features will be included in the model, then a model whose distribution $\mathbf{p}$ is in the subset $\mathcal{C} \subset \mathcal{P}$, defined by equation 2.3, must be found.

$$\mathcal{C} \equiv \{\mathbf{p} \in \mathcal{P} \,|\, p[\mathbf{f_i}] = \tilde{p}[\mathbf{f_i}] \text{ for } i \in \{1, 2, \ldots n\}\} \tag{2.3}$$

That is, the space of conditional probability distribution whose expected values for the n features agree with the empirical statistics. Unfortunately, these constraints do not determine a unique distribution $\mathbf{p}$. Moreover, $\mathcal{C}$ allows infinite models.

Thus, a criterion to select a probabilistic distribution from the space $\mathcal{C}$ is needed. For this, the maximum entropy principle is used. This principle states that the model whose distribution is *most uniform* should be selected, so that "it agrees with everything that is known but carefully avoids assuming everything that is not known" (Jaynes 1991).

For finding such a model, the *entropy* is used as a measure of the uniformity of a conditional distribution $\mathbf{p}(\mathbf{y}|\mathbf{x})$, given by

$$H(\mathbf{p}) \equiv -\sum_{x,y} \tilde{p}(x)p(y|x)\log p(y|x) \tag{2.4}$$

If a random process has m possible outputs —i.e. $|Y| = m$— then the entropy of a conditional distribution for that model can vary from zero —a model with no uncertainty— and $\log m$ —the uniform distribution over $y_1, y_2, \ldots, y_m$.

Based on this definition, the principle of maximum entropy can be restated as follows:

> From the set $\mathcal{C}$ of allowed probability distributions, select the model $\mathbf{p}_* \in \mathcal{C}$ with the maximum entropy $H(\mathbf{p})$:
>
> $$\mathbf{p}_* = \underset{\mathbf{p}\in\mathcal{C}}{\operatorname{argmax}}\, H(\mathbf{p}) \tag{2.5}$$

It has been shown that there is a unique, well-defined model which is the solution to equation 2.5 (Della Pietra et al. 1995, Berger et al. 1996). Furthermore, such a model has the exponential form

$$\mathbf{p}_*(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_\lambda(\mathbf{x})} e^{\sum_i \lambda_i f_i(\mathbf{x},\mathbf{y})} \tag{2.6}$$

$$Z_\lambda(\mathbf{x}) = \sum_y e^{\sum_i \lambda_i f_i(\mathbf{x},y)} \tag{2.7}$$

in which each feature is associated with a parameter λ_i and $Z_\lambda(\mathbf{x})$ is a normalising constant, determined by the requirement that the conditional probabilities for a given context x add up to one over the classes $y \in Y$. The reader is invited to see the details of the derivation of this parametric form —with the constrained optimisation method of Lagrange multipliers— in Della Pietra et al. (1995). Sometimes the equivalent parametric from

$$\mathbf{p}_*(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_i \alpha_i^{f_i(\mathbf{x},\mathbf{y})} \tag{2.8}$$

$$Z(\mathbf{x}) = \sum_y \prod_i \alpha_i^{f_i(\mathbf{x},y)} \tag{2.9}$$

is used (Ratnaparkhi 1998, Borthwick 1999), where each parameter $\alpha_i \equiv e^{\lambda_i}$ and $Z(\mathbf{x})$ is the appropriately modified normalising constant.

Interestingly, these exponential forms have been obtained from different points of view, namely Information Theory and Constrained Optimisation Theory on probability distributions. Moreover, both approaches have shown that there exists a unique model $\mathbf{q}_*$ which solves

$$\mathbf{q}_* = \underset{\mathbf{q}\in\mathcal{C}}{\operatorname{argmax}}\, L(\mathbf{q}) \tag{2.10}$$

where $L(\mathbf{q})$ corresponds to the log-likelihood of a conditional probability distribution $\mathbf{q}$ over the set of training samples

$$L(\mathbf{q}) = \sum_{x,y} \tilde{p}(x,y) \log q(y|x) \tag{2.11}$$

In general, the maximum likelihood and the maximum entropy frameworks are two different methods for statistical modelling. However, both analyses have found that in this case $\mathbf{q}_*$ also solves equation 2.5, and therefore $\mathbf{q}_* = \mathbf{p}_*$.

The fact that a model obtained under the maximum entropy approach is the same as the model obtained by —the more traditional technique of— maximising the probability of the training samples, is a strong argument in favour of the validity of the method (Ratnaparkhi 1998, Borthwick 1999).

2.4.3 Learning Maximum Entropy Models

Any statistical modelling problem requires a two step process:

1. finding the appropriate set of facts to describe the random process to model, and
2. incorporate these facts into the model.

The first step is related to determining the set of features that will describe the process best.

The second step is partially solved by the discussion in the previous section. It is just necessary to calculate the expected values of the selected features according to the training samples and find a model which satisfies the constraints that these values impose. What is not said is how this model can be found.

In this section, some ways in which these two steps can be addressed are discussed.

2.4.3.1 Parameter Estimation

There are two main algorithms for estimating the parameter of a model with the exponential forms shown above (Darroch and Ratcliff 1972, Della Pietra et al. 1995): the Generalised Iterative Scaling Algorithm (GIS), and the Improved Iterative Scaling (IIS). Actually IIS can be seen as an optimisation of GIS.

The GIS algorithm requires that the features sum to some constant K for any training sample, that is

Algorithm 2.2: Generalised Iterative Scaling. Adapted from Ratnaparkhi (1998).

Input: a set of non-negative feature functions $F = \{f_1, f_2, \ldots, f_j, \ldots, f_n\}$, their empirical expected values $\{\tilde{p}[f_j]\}_n$, the empirical distribution $\tilde{p}(x, y)$ and the normalisation constant C

Output: the maximum entropy model p_*

procedure GIS($F, \{\tilde{p}[f_j]\}_n, \tilde{p}(x), C$)
  Initialise parameters: $\alpha_j^{(0)} \leftarrow 1$
  $i \leftarrow 1$
  repeat
    Define the current distribution $p^{(i)}$:
      $p^{(i)}(y|x) \leftarrow \frac{1}{Z(x)} \prod_{j=1}^{n} \left(\alpha_j^{(i)}\right)^{f_j(x,y)}$
    Calculate the expected values of each feature f_j from $p^{(i)}$:
      $p[f_j]^{(i)} \leftarrow \sum_x \tilde{p}(x) \sum_y p(y|x)^{(i)} f_j(x, y)$
    Update the parameters for the next iteration:
      $\alpha_j^{(i+1)} \leftarrow \alpha_j^{(i)} \left(\frac{\tilde{p}[f_j]}{p[f_j]^{(i)}}\right)^{\frac{1}{C}}$
    $j \leftarrow j + 1$
  until $p^{(i)}$ has converged
  $p_* \leftarrow p^{(i)}(y|x)$
end procedure

$$\sum_i f_i(x, y) = K \tag{2.12}$$

If this condition is not met, a correction feature is introduced into the model so that the constraint 2.12 is satisfied. In theory, a correction constant for all (x, y) pairs should be derived from the space of possible events $X \times Y$. However, this is not practical and correction constants are estimated from the training samples as

$$k = \max_{x \in X, y \in Y} \sum_i f_i(x, y)$$

which is accurate enough in practice when the set of training samples is large. In this way, the following *correction feature* $\mathbf{f_l}$ is added for each sample

$$f_l(x, y) = K - \sum_i f_i(x, y)$$

Algorithm 2.2 describe the GIS procedure. Darroch and Ratcliff (1972) showed that the model built by this algorithm converges to $\mathbf{p}_*$. This version of the algorithm is for estimating the α_i parameters of models with the exponential form of equation 2.8 (Borthwick 1999).

The key step in each iteration of the GIS algorithm is the calculation of the expectations for the set of features. Suppose there are N training samples, m possible classes and l feature functions, then the running time at each iteration is $O(Nml)$ (Ratnaparkhi 1998).

The Improve Iterative Scaling algorithm follows the same basic steps as the GIS algorithm. The improvement is related to the way in which the expected value of a feature function is obtained.

Algorithm 2.3: Improved Iterative Scaling. Adapted from Berger et al. (1996).

Input: a set of non-negative feature functions $F = \{f_1, f_2, \ldots, f_j, \ldots, f_n\}$, and the empirical distribution $\tilde{p}(x, y)$
Output: the maximum entropy model p_*

procedure IIS($F, \tilde{p}(x, y)$)
  Initialise parameters: $\lambda_j^{(0)} \leftarrow 0$
  $i \leftarrow 1$
  repeat
    Define the current distribution $p^{(i)}$:
      $p^{(i)}(y|x) \leftarrow \frac{1}{Z_\lambda(x)} e^{\sum_{j=1}^{n} \lambda_j^{(i)} f_j(x,y)}$
    Find $\Delta\lambda_j$ for each feature which is solution to:
      $p[f_j e^{\Delta\lambda_j f^\#}]^{(i)} = \tilde{p}[f_j]$ with $f^\#(x, y) \equiv \sum_{k=1}^{n} f_k(x, y)$
    Update the parameters for the next iteration:
      $\lambda_j^{(i+1)} \leftarrow \lambda_j^{(i)} + \Delta\lambda_j$
    $j \leftarrow j + 1$
  until $p^{(i)}$ has converged
  $p_* \leftarrow p^{(i)}(y|x)$
end procedure

Algorithm 2.3 outlines the procedure as described in Berger et al. (1996), which determines the optimal values of the λ_i parameters of a model with the exponential form of equation 2.6.

The key step of the Improved Iterative Scaling Algorithm is the calculation of each increment $\Delta\lambda_i$. Unlike GIS, this algorithm does not require $S(x, y)$ to be a constant; it can compute these increments numerically by Newton's method or other equivalent techniques (Berger et al. 1996, Borthwick 1999). Della Pietra et al. (1995) showed that $\mathbf{p}(\mathbf{x}, \mathbf{y})^{(j)}$ converges to $\mathbf{p}_*(\mathbf{x}, \mathbf{y})$.

It should be noted that both algorithms terminate when convergence to the maximum entropy model has been reached, that is when the change in the parameters estimated in the iteration is zero or negligible. However, stopping the algorithm after a fixed number of iterations works well generally and it is the most commonly used criterion in practice. For example, Ratnaparkhi (1998) uses 100 iterations in all the four applications of MEMs he presents.

2.4.3.2 Feature selection

As explained previously, building a maximum entropy model involves two steps. The iterative scaling algorithms presented above provide a method for determining the optimal parameters of the model, once the set of feature functions has been defined.

The problem is that maximum entropy framework specifies how constraints should be combined, but it does not stipulate directly what constraints should be included into a model. Moreover —as discussed earlier— features which are overlapping each other

and some that are even not relevant can be included, and the model should be able to deal with them correctly.

Penrose (1979) complained about this deficiency of the maximum entropy framework. Jaynes's (1991) answer was rather strong:

> "Well, we had thought it rather obvious that one should always take into account *all of the relevant information one has*; and find it incredible that anyone could have supposed differently."

Risking Mr. Jaynes's rage, it might be said that determining what information is *relevant* is not so obvious to everyone, specially if that person is not an expert on the random process being modelled —but a simple computer science student, say— and such an expert is not at hand to be consulted.

The view expressed in Jaynes (1991) is that the modeller has all the responsibility for providing the framework with the features that describe the random process best. However, computer scientists do not usually like this kind of answer and are always looking for ways in which computers can *help* to solve any task with as little human intervention as possible.

Help from a computer becomes more relevant when the target process is complex, since it is not uncommon to find problems with thousands or even millions of possible features —such NLP problems— from which only a small fraction are expected to be crucial for modelling the process (Berger et al. 1996, Blum and Langley 1997). In these circumstances, feature selection is critical as the iterative algorithms of section 2.4.3.1 are computationally costly, and their running times depend on the number of constraints to be considered by the model. Moreover, there are two reasons which make feature selection even more critical for maximum entropy models.

Firstly, the iterative algorithms for estimating parameters do not look for higher-order interaction between features. This is consistent with the maximum entropy principle in that no assumptions should be made other than the constraints to be met. On the other hand, if there exists a special interaction between two features, their combination in a more complex feature function should result in a more accurate model (Mikheev 1998). For example, a model for finding organisation names can combine the weights of the features "this word is capitalised" and "the next word is plc" which are estimated separately, but it can also use the complex feature that results from the *conjunction* of these two features, i.e the feature "this word is capitalised and the next word is plc", whose weight is estimated from the training examples that fire both original features simultaneously.

Mikheev (1998) states that if the conjoined features are not independent —as in the above example— then the resulting complex feature should produce a better prediction. However, as things are, this higher-order feature needs to be included explicitly as a constraint by the modeller from the beginning.

Secondly, although an MEM can handle some irrelevant features by assigning the appropriate near-zero weights (Rosenfeld 1996, Mikheev 1998), the introduction of many irrelevant features can degrade the predictions of the model. Similarly, MEMs cope well with overlapping features, but a high degree of overlap requires more iterations in the iterative scaling algorithms (Borthwick 1999, Ristad 1998).

The two step task of deciding which features to use in describing a concept and then deciding how to combine them, is not a property of maximum entropy models only but present in most machine learning methods (Blum and Langley 1997). In all these techniques —though they may significantly differ in the approach— there exist induction algorithms which aim to scale well from domains with many irrelevant features.

Thus feature selection defines a two level process in which modellers are responsible for establishing a set of initial features that they think might be useful in describing the target concept, and then a refinement of this set is performed by an inductive algorithm. This combination is very common in practice, including work with maximum entropy models.

A good example of this combination can be found in Borthwick (1999). He presents a basic feature selection method in which an algorithm collects a pool of features, and then a *cutoff* filter is applied to discard all features that are not fired more than three times. Both the features extracted by the algorithm and the cutoff threshold were defined by Borthwick and were probably based on his intuition of the problem being solved. Although this simple method worked well, he recognised the necessity of deviating from it to solve some practical problems. He incremented the cutoff threshold for too frequent features, which resulted in a reduction of the size of the model without a significant loss in accuracy. He also excluded features which were fired by default in many contexts and features which only predicted the default class of the process.

Ratnaparkhi (1998) also uses the frequency-based count cutoff to select the features to be included in the model with success. He set the cutoff threshold to values five and ten in most cases. For one task, Ratnaparkhi set this value to zero. He argues that this value should be used when the initial feature set consists of only specific features whose valuable information might be thrown away when discarded. On the other hand, a threshold greater than zero is useful when the set of features includes generalised features in addition to the specific ones. Thus, most of the disregarded

features will be specific features which are unreliable sources of evidence —due to their low counts among the training samples— and the model will be able to fall back on —more reliable— generalised features for predictions. Ratnaparkhi also states that the features used in solving the NLP problems presented in his thesis are ***knowledge-poor*** —i.e. do not require linguistic expertise— by design, so that a computer is forced to learn as much as possible from the training samples.

Despite the success of frequency cutoff, the modeller has the main responsibility for determining which features are included into a model. Hence, this kind of selection is still difficult to apply when the modeller has little idea of which features might be useful.

Two kinds of features in this process can be distinguished: *atomic features* which cannot be decomposed into simpler features, and *complex features* which are built by making conjunctions of atomic features. The latter ones were also called higher-order features earlier. In the terms of Ratnaparkhi (1998), a specific feature is a feature compounded from many atomic features —and thus fired for a small number of training samples— whereas generalised features involved few or just one atomic feature, being fired more frequently.

Borthwick (1999) uses only atomic features for his model, but he describes an attempt to include complex features. He realised that the cutoff method was not appropriate for performing the selection when higher-order features were involved, and applied a multi-stage process.

He first created a pool of complex features of the form $f_{ij} = f_i \wedge f_j$ that satisfy $\#f_{ij} > 3$, $\#f_i - \#f_{ij} > 3$ and $\#f_j - \#f_{ij} > 3$, where $\#f$ is the number of times a feature f is fired in the training samples. After this filtering by count, he obtained a pool of about 139,000 features. Unfortunately, this was too large to handle by the implementation of the IIS algorithm he was using (Ristad 1998).

Therefore, he defined a second stage in which a selection method proposed by Ristad (1997) is applied. In this method, features are selected by comparing the model's expectations of how often they should occur in the training samples against the empirical expectations. Let d_{f_i} be this difference calculated as

$$d_{f_i} = \left| \sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y) - \sum_{x,y} \tilde{p}(x,y) f_i(x,y) \right|$$

where $\mathbf{p(y|x)}$ is a maximum entropy model which does not include the candidate features. By the restrictions imposed by the framework, if a candidate feature f_i were added to the model, then $d_{f_i} = 0$. Hence, the higher the value of d_{f_i}, the bigger the impact on the model if f_i were included in the model.

Algorithm 2.4: Basic Feature Selection. Adapted from Berger et al. (1996).

Input: a large pool of candidate features F and the empirical distribution $\tilde{p}(x, y)$
Output: a set S of active features and the maximum entropy model p_S that includes these features

procedure BFS($F, \tilde{p}(x, y)$)
  Initialise $S \leftarrow \varnothing$ and $p_S \leftarrow$ the uniform distribution over Y
  repeat
    for each candidate feature $f_j \in F$ **do**
      Compute the model $p_{S \cup \{f_j\}}$ using an iterative scaling algorithm
      Compute the gain in log-likelihood from adding this feature:
        $\Delta\mathcal{L}(S, f_j) \leftarrow \mathcal{L}(p_{S \cup \{f_i\}}) - \mathcal{L}(p_S)$
    end for
    Check the termination condition
    $f_* \leftarrow \operatorname{argmax}_j \Delta\mathcal{L}(S, f_j)$
    $S \leftarrow S \cup f_*$
    Compute the model p_S using an iterative scaling algorithm
  until convergence
end procedure

Although Borthwick (1999) does not explain exactly how he used this number to rule out features, it can be guessed that he set another threshold quantity and discarded any candidate feature which did not pass this threshold. Unfortunately, this method was not sufficient for obtaining the final set of features.

Borthwick (1999) reports that there were features that fired too frequently, causing numerical problems in the implementation of the IIS algorithm. Therefore, he had to add a third stage in which features were eliminated from the pool manually. Unfortunately, the resulting pool of features did not outperform the model obtained with atomic features only.

Berger et al. (1996) suggested a different approach for the feature selection problem. In this, the modeller is only responsible for providing as large a pool of features as possible. Thus it may include both atomic and complex features, and they need not be relevant or useful. This approach is presented in algorithm 2.4. Basically it builds incrementally —with a strategy similar to the induction of decision trees— a set S of features to be considered by the model, from a large pool of features F, by selecting at each step the feature that provides the greatest improvement in log-likelihood of the model with respect to the training samples.

One issue not specified in algorithm 2.4 is the termination condition. Obviously, the algorithm should ideally stop when all useful features are included in the set S. One reasonable stopping criterion would be to subject f_* to an increase of likelihood on held-out training samples. If this feature does not satisfy this condition, then it is discarded and the algorithm stops.

However, the biggest problem in algorithm 2.4 is that at each step the maximum entropy model $\mathbf{p_{S \cup \{f_i\}}}$ must be computed. Despite the efficiency of the iterative scaling

algorithms presented in section 2.4.3.1, this is a computationally costly task that makes the method impractical. For this reason, Della Pietra et al. (1995) and Berger et al. (1996) make the algorithm greedy but more efficient. Instead of computing $\Delta\mathcal{L}(S, f_i)$, the greedy version calculates its approximation $\sim\Delta\mathcal{L}(S, f_i)$ by keeping all the parameters of the model $\mathbf{p_S}$ fixed and determining only the new parameter required for the constraint imposed by f_i. After selecting the candidate feature which maximises this approximated value, it is added to the set S and the parameters of the model are recomputed. This approach estimated good features relatively fast but it does not guarantee to make the best selection at each step because adding a new feature to a model can change *all* its parameters.

Ratnaparkhi (1998) conducted controlled experiments to evaluate the differences in the frequency cutoff method he utilised and the Random Field Induction method explained above. He allowed the induction algorithm to run a fixed number of iteration M, and then selected the set S_i with $i \in \{1, \ldots, M\}$ which yielded the highest log-likelihood on held-out training samples. The aim of these experiment was to assess whether the smaller set of features produced by the inductive algorithm resulted in better accuracy of the final model. This set is clearly smaller because, unlike the frequency cutoff method, non-informative features introduce negligible gains in likelihood and are consequently discarded by the algorithm.

Ratnaparkhi (1998) found that both approaches —i.e. frequency cutoff and incremental induction— obtained models that perform similarly. The main differences between the methods are in the running time and the readability of the resulting set of features. He concludes that if efficiency is the main issue, then frequency cutoff should be used as it is much faster than the inductive algorithm. On the other hand, if the goal is to obtain a readable set of features, then incremental induction should be used as it yields a concise and understandable list of features.

Mikheev (1998) also conducted a study of feature selection and proposed a new method based on the construction of a lattice of higher-order features, which he calls *collocations*. The basic idea is to include collocations —i.e. complex features which are empirically observed— and features which might provide significant generalisations over the observed collocations into this lattice. Only then, important features are selected for the model.

He starts by collecting training samples and representing them as *configuration* of atomic features. Then he applies some sort of goal regression to identify configurations which can safely be removed from the training sample space in order to reduce the dimensionality of the task.

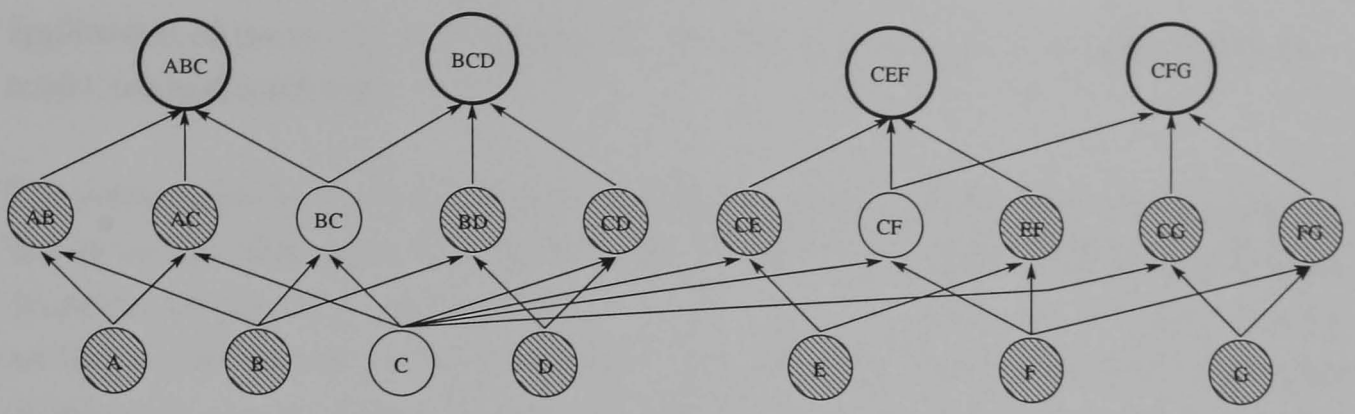


Figure 2.1: Example of a collocation lattice in which thick circles represent reference nodes and filled circles represent possible hidden nodes that are not part of the lattice. Adapted from Mikheev (1998).

The resulting configurations are organised as cliques of features in a lattice. Observed configurations are represented as *reference nodes*. For example, suppose the training samples provide the configurations [ABC], [BCD], [CEF] and [CFG]. Figure 2.1 shows the lattice that these configurations generate. In this figure, configurations are represented as reference nodes denoted by thick circles. Then, nodes which share part of at least other two nodes in the lattice are added to support generalisations over the domain. These kinds of nodes are called *latent* or *hidden nodes*, and are not normally observed on their own but only as part of reference nodes. As shown in figure 2.1, the hidden nodes representing the collocations [BC] and [CF] and the hidden node representing the atomic feature [C] are also considered in the lattice. All other hidden nodes are discarded because they directly support only one node and thus they do not provide any generalisation.

Each node has associated with it two frequency counts: the *configuration frequency counts* (*cf*), which corresponds to the number of times the represented configuration has been observed in the training samples, and the *feature frequency count* (*ff*), which corresponds to the number of times that the represented feature —atomic or complex— has been seen in all observed configurations.

Therefore, in reference nodes these counts normally have the same value, that is $cf(f_i) = ff(f_i)$. Hidden nodes normally have zero configuration frequencies ($cf(f_i) = 0$) but non-zero feature frequencies ($ff(f_i) > 0$). It might be the case that a reference node could also be a hidden node for another higher-order reference node. In this case, its configuration frequency count will not be zero and it might be different from its feature frequency count.

Mikheev (1998) discussed two ways in which features from this lattice —which he calls the *empirical lattice*— can be selected for the model. The first idea is to apply frequency cutoff over all nodes in the lattice. This idea is easily implemented but if too many features are selected, then both the estimation of the model's parameters and the

application of the model to new examples —which is linear in the number of features— might become inefficient.

The second idea is to try determining which features contribute to the frequency distribution on the reference nodes. This is done by creating an *optimised lattice* based on the empirical lattice. The optimised lattice is initially empty and is built incrementally by adding a node at each step, together with the nodes which are the minimal collocations of this node and the nodes already included in the lattice. Thus, the optimised lattice always contains non-overlapping feature cliques. In this way, there is no need to use iterative scaling to account for overlaps among features.

For the example of figure 2.1, suppose the first node to be added to the optimised lattice is the one representing the feature [A]. The configuration frequency in the optimised lattice ($\hat{cf}$) of this node will concentrate all configuration frequencies of itself and the higher-order related nodes, thus $\hat{cf}([A]) = cf([A]) + cf([AB]) + cf([AC]) + cf([ABC])$. Now suppose the feature [B] is selected to be included in the optimised lattice. This will also require the node for [AB] being added and redistributing the frequencies among these three nodes. The resulting frequency counts will be $\hat{cf}([A]) = cf([A]) + cf([AC])$, $\hat{cf}([B]) = cf([B]) + cf([BC])$ and $\hat{cf}([AB]) = cf([AB]) + cf([ABC])$. If at some point the node representing the feature [C] is added to the optimised lattice, then the whole feature clique involving these features will be present and with identical frequency counts to those in the empirical lattice.

In this selection method, the node to be added at each step is the one which makes the greatest increment in log-likelihood of the optimised lattice with respect to the reference nodes in the empirical lattice. For this, the probability of a node is considered to be the probability of the highest related node in the optimised lattice, which could be the node itself. It is also necessary to define a "lattice root" node which is used as default related node for estimating the probabilities of reference nodes in cliques that are not yet considered in the optimised lattice. Finally, Mikheev (1998) also introduces a smoothing scheme which does not affect frequent nodes, but considerably penalises sparse collocations. In this way, this method defines a greedy hill-climbing algorithm with maximum likelihood evaluation function which adds a winning set of non-overlapping features at a time, whose solution can be easily derived and re-calculated from observed frequencies.

It is not clear whether one of these two alternatives —i.e. the empirical lattice or the optimised lattice— is better than the other. Mikheev (1998) points out that this is an empirical matter which depends on the complexity of the task, because the time needed for the feature selection can compensate for the time saved during parameter estimation.

Table 2.1: Comparison of lattice methods for feature selection and Random Field Induction in tasks with different dimensionality. Adapted from Mikheev (1998).

	Dimensionality 9		
	Empirical Lattice	**Optimised Lattice**	**Random Field Induction**
size	195	148	44
time	00:29	00:30	33:26
training accuracy	85.04	85.04	84.85
test accuracy	83.99	83.99	84.27

	Dimensionality 11		
	Empirical Lattice	**Optimised Lattice**	**Random Field Induction**
size	628	271	46
time	03:06	01:59	1:09:26
training accuracy	85.47	85.38	84.97
test accuracy	85.39	85.67	84.55

	Dimensionality 13		
	Empirical Lattice	**Optimised Lattice**	**Random Field Induction**
size	1,530	449	54
time	10:55	08:15	2:23:48
training accuracy	86.04	85.37	85.88
test accuracy	85.67	87.64	85.99

Table 2.1 shows experiments —reported in Mikheev (1998)— with both methods on tasks with different dimensionality, which confirm this observation. What is clearly determined by these experiments is that the Random Field Induction discussed earlier required much more training time with similar levels of performance.

The main advantage of the optimised lattice method is that it provides a much smaller maximum entropy model than just using the empirical lattice. This is a very important advantage when additional optimisation is applied over the features, which normally does require iterative scaling. In Mikheev (1998), an approach to further pruning the number of features considered for the final model is proposed. This method requires a quarter of the time when working on the optimised lattice compared to working on the empirical lattice. In addition, the resulting maximum entropy model after this optimisation consistently showed better performance than both lattice methods and the Random Field Induction approach.

The main problem with Mikheev's (1998) method is that building the feature collocation lattice can be prohibitive for tasks with many dimensions. Nonetheless, this approach can provide a better model and in faster time than the inductive technique for many practical tasks. For tasks with more than 25-30 dimensions, Mikheev suggests Random Field Induction should be used.

2.4.4 Modelling

The previous section has detailed methods for estimating the parameters of a maximum entropy model and methods for selecting the features to be included in such a model. However, all the discussed approaches for selecting the correct set of features —namely frequency cutoff, Random Field Induction, empirical lattice and optimised lattice— start from an initial set of atomic and higher-order features which *must be provided by the modeller.*

This is not the case in other machine learning approaches, which automatically look for the best conjunction of attributes (Blum and Langley 1997), in which the modeller is required only to provide the *initial set of atomic features.*

It could be argued that this is all that is needed for the lattice methods. This is true in general, but it is not clear that the method might be useful when the task being modelled has many dimensions[3]. For example, suppose there are 18 non-valued atomic features for each training sample of an NLP classification task, of which nine correspond to *tokens* in a sequence within a context window[4]. Thus the number of valued atomic features will normally be very large. It will be rather difficult if collocations of 17 valued atomic features —which in the best case do not include one of the nine tokens in the sequence— will support more than one observed configuration. Hence it is likely that none of these hidden nodes should be present in the empirical lattice and thus the only nodes to be considered for the model would be the observed conjunctions of 18 features.

The usefulness of the method cannot be completely disregarded because there are applications of the lattice method to NLP tasks in Mikheev (1998), though everything suggests that in all of them the initial configurations were not so sparse. Moreover, in an application for determining whether a period is marking the end of a sentence, only the 238 *most frequent* atomic features are used, and no explanation for this reduction is given. Furthermore, this set must correspond to 238 valued atomic features —and only around six non-valued atomic features— as they only generate 8,245 nodes in the empirical lattice.

This last point may require further discussion. If the contexts considered for modelling a random process contain six non-valued atomic features, then the number of possible features for each training sample j is given by

$$|\mathcal{F}_j| = \binom{6}{6} + \binom{6}{5} + \binom{6}{4} + \binom{6}{3} + \binom{6}{2} + \binom{6}{1} = 63$$

[3]Dimensions is Mikheev's (1998) term to refer to *non-valued features.* For example, "the part-of-speech of the next word" can be included in the information provided for a set of training examples. This corresponds to one non-valued atomic feature. In the data, however, this feature will be valued: "the part-of-speech of the next word is Noun", "the part-of-speech of the next word is Verb", etc. These instantiations of the abstract feature are *valued atomic features* or just atomic features.

[4]Such a problem will be encountered in later chapters.

that is, one conjunction of six features, six conjunctions of five features, 15 conjunctions of four features, 20 conjunctions of three features, 15 conjunctions of two features and six atomic features. Hence it is impossible that 238 non-value atomic could generate only 8,245 possible feature collocations. Moreover, this number needs to be multiplied by the possible values of each conjunction. For example if all six atomic features have three possible values, then the theoretical number of possible conjunctions is

$$|\mathcal{F}| = 3^6 + 6 \cdot 3^5 + 15 \cdot 3^4 + 20 \cdot 3^3 + 15 \cdot 3^2 + 6 \cdot 3 = 4,095$$

which corresponds to the maximum size of the empirical lattice. In NLP tasks, atomic features are rarely three-valued (e.g. part-of-speech tags) and frequently lexical forms — i.e. tokens— are included. Thus the number of possible conjunctions is normally huge. However, this theoretical number of feature collocations is almost never encountered in practice, and the lattice method can be used even for tasks with moderate dimensionality if common high-order collocations can be found. Nonetheless, the number of collocations is still extremely high and considering all of them also makes the use of Random Field Induction impractical.

Mikheev's (1998) lattice methods can be seen as a filter approach to feature selection, because it filters out irrelevant features before the induction occurs[5]. Filter approaches are common in machine learning and the actual techniques employed vary enormously. They have many advantages, but the most attractive characteristic is that they are normally independent of the induction algorithm that will use their output and thus can be combined with any such method (Blum and Langley 1997).

One approach successfully applied for filtering is the use of decision trees. For example, Kubat and colleagues have used them to filter attributes for a Bayesian classifier and initialise neural networks (Kubat, Flotzinger and Pfurtscheller 1993, Kubat 1998, for example). Cardie (1993) used them to select the features to be included in a k-nearest neighbour retrieval function for solving an NLP problem. She found that this hybrid technique outperforms alternative systems that utilise only decision trees or k-nearest neighbour, and also two other case-based systems that incorporated — potentially expensive— expert knowledge.

More recently, Park and Zhang (2002) presented an approach in which decision trees are used to generate higher-order features for a maximum entropy model to solve text chunking. They showed that a decision tree can easily be represented as feature functions for a maximum entropy modelling. For example, consider the decision tree of figure 2.2, which makes predictions for the part-of-speech tagging task discussed in section 2.4.1. Rules in this tree can be directly denoted with the if-then form used in feature functions:

[5]In this context, the induction step would be the optimisation with iterative scaling or the estimation of the parameters of the model.

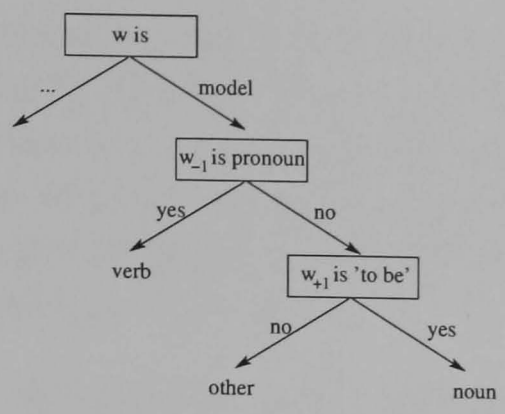


Figure 2.2: Part of a decision tree for part-of-speech tagging. The rules are: if the current word is 'model' then (a) if the previous word is a pronoun, then predict current word is a verb; otherwise (b) if the next word is a form of the verb 'to be', then predict current word is a noun; (c) otherwise predict the current word is not a verb nor a noun.

$$\mathbf{f_1}(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} = \textit{verb} \text{ and } (w_0 \text{ is 'model' and } w_{-1} \text{ is a pronoun}) \text{ in } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f_2}(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} = \textit{noun} \text{ and } (w_0 \text{ is 'model' and } w_{-1} \text{ is not a pronoun and} \\ & w_{+1} \text{ is a form of the verb 'to be') in } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{f_3}(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{y} = \textit{other} \text{ and } (w_0 \text{ is 'model' and } w_{-1} \text{ is not a pronoun and} \\ & w_{+1} \text{ is not a form of the verb 'to be') in } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

Park and Zhang (2002) transcribe each path from the root to leaves of the decision tree as feature functions. Then, they argue that because algorithms for the induction of decision trees (Quinlan 1983, Quinlan 1993) try to partition the sample space into non-overlapping regions, each feature —i.e. each path— can be considered to have the same importance weight. They introduce these features into a maximum entropy model which has to re-weight the complex features in order to optimise their predictive power. This method obtained an improvement of 2.34% in accuracy with respect to the performance of the decision tree. More importantly, the number of errors were reduced by 41.64% from 2,663 to 1,554 on test data. This improvement is important as decision trees are themselves strong classifiers.

However, the main contribution of Park and Zhang's (2002) approach is that it automatically selects features for a maximum entropy model starting from atomic features only. Thus the linguistic knowledge required for modelling the task is considerably reduced.

In the decision tree of figure 2.2, three non-valued atomic features are used: f_1= "the string of the current word is", f_2= "the previous word is a pronoun" and f_3= "the next word is a form of the verb to be". Thus the set of possible features defined by these features is $\{f_1, f_2, f_3, f_1 \wedge f_2, f_1 \wedge f_3, f_2 \wedge f_3, f_1 \wedge f_2 \wedge f_3\}$. Now suppose that w can

take values from —a rather small— lexicon of 3,000 words. Because f_2 and f_3 are binary valued, there are 12,000 possible configurations just of the form $(f_1 \wedge f_2 \wedge f_3)$ and 27,008 possible valued features in total. If all 3,000 words in the lexicon are present in the *training corpus*, from which the training samples are taken, then this potential number of valued features will become the actual size of the feature space for both Random Field Induction and the lattice methods.

A decision tree searches for the most informative combinations of features that split the data. Therefore, the tree of figure 2.2 is indicating than the collocations $f_1(\textit{model}) \wedge f_2$ and $f_1(\textit{model}) \wedge f_2 \wedge f_3$ are valuable in modelling the task. This tree also indicates that there is no need to try more complex combinations with these features. This automatically discards collocations such as $f_1(\textit{model}) \wedge f_2 \wedge f_3 \wedge f_4$, $f_1(\textit{model}) \wedge f_2 \wedge f_5 \wedge f_6$, etc.

Thus, if only conjunctions considered by a decision tree are used as the initial set of features, for both Random Field Induction and lattice methods, a significant reduction in size of the task can be obtained making practical the use of feature selection techniques in tasks with more dimensions. This is an interesting idea which deserves some investigation.

Nonetheless, there is an important limitation for this idea to work: induction algorithms for decision trees are computationally expensive. For example, both Ratnaparkhi (1998) and Borthwick (1999) conducted experiments that aimed to compare the performance of their maximum entropy approaches against known, commercial decision tree induction algorithms. In addition to obtain results which suggest that maximum entropy models outperform decision tree classifiers, neither of them could obtain decision trees for tasks that included lexical features (strings) for moderate-size training corpora.

This last problem could be solved by using shallow decision trees or decision trees with fixed depths, for which relatively efficient induction algorithms have been proposed (Auer, Holte and Maass 1995, Dobkin, Fulton, Gunopulos, Kasif and Salzberg 2000). Obviously, finding sub-optimal conjunctions only is a risk as the number of features in each collocation would be restricted by the depth of these decision trees. Nevertheless, it would provide a set of initial higher-order features which should make practical the induction methods discussed above, starting only from the specification of atomic features.

2.4.5 Limitations

One of the limitations of using maximum entropy models has already been discussed: their computational cost. Although building an MEM is a tractable problem, large tasks

with thousands or millions of training samples and with many dimensions that cannot be pruned without expert knowledge require a considerable amount of both memory and CPU time.

A more important limitation is related to the convergence of the parameters when an exact solution for equations 2.6 and 2.8 does not exist. This is the case with random processes which require a model to predict $p_*(y|x) = 1$ for some pair (x, y). In such cases iterative scaling will always increase the parameter associated with the constraint imposed by this pair.

A related problem is that the maximum entropy framework "gives infinite confidence to contexts that are not ambiguous with respect to the predictions with which they occur, regardless of their frequency" (Ratnaparkhi 1998). This fact produces an undesired effect when parameters interact to make a prediction. Infrequent events tend to be unambiguous and —as explained above— normally get higher parameters in the model. Thus when combined with other evidence which might appear much more frequently in the training samples, parameters for infrequent events will dominate the prediction.

However, these limitations are seldom encountered in NLP tasks (Ratnaparkhi 1998). Moreover, the count cutoff used for feature selection normally discards these problematic infrequent features. Another suggested solution for these problems is the application of smoothing techniques and using *soft* constraints (Lau 1994).

A final observation is that only binary feature functions are considered in most —if not all— applications and implementations of the maximum entropy model. However, this is not a limitation of the framework itself (recall that the iterative scaling algorithm only requires positive initial parameters). This limitation is not a problem, since a feature of the form:

$$\mathbf{f_1(x,y)} = \begin{cases} count(w) & \text{if } \mathbf{y} = verb \text{ and } w \text{ occurs in } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

in which $count(w)$ is the number of times that a word w is found in a document, can be re-written as the following binary feature:

$$\mathbf{f_1(x,y)} = \begin{cases} 1 & \text{if } \mathbf{y} = verb \text{ and } w \text{ occurs frequently in } \mathbf{x} \\ 0 & \text{otherwise} \end{cases}$$

Although they are not exactly the same feature, they provide similar evidence for the prediction. In any case, binary features have proved to be sufficient to capture enough information —at least at word or sentence level— for making accurate predictions (Ratnaparkhi 1998).

2.4.6 Maximum entropy tools

There are a number of tools that implement the maximum entropy framework. The main advantages of using an existing implementation —rather than constructing one from scratch— are that the significant time needed for programming and debugging can be saved and any results obtained are supported by previous work with the same tool.

The most popular tools are The Maximum Entropy Modeling Toolkit (MEMT) (Ristad 1998) and the OpenNLP Maxent package (MaxEnt) (Baldrige and Bierner 2001). MEMT supports the construction and application of maximum entropy models for discrete domains. This tool effectively implements the IIS algorithm for estimating the parameters of the model — in the C language. MaxEnt also supports maximum entropy models for discrete domains, but it implements the GIS algorithm — in the Java language.

It is not clear that one tool is better than the other. Nonetheless, MaxEnt provides more support: expected values calculation and the frequency cutoff algorithm, if that is the feature selection mechanism to be used. In any case, both tools can be used for parameter estimation only after doing the selection in a preprocessing step.

Borthwick (1999) reported some numerical problems with MEMT, which have not been reported for MaxEnt so far. This might be considered as a reason for preferring the latter package. Nevertheless, the main reason for using MaxEnt is that this tool is free, whereas MEMT is nowadays part of PMT, a commercial product from Mnemonic Technology Inc.

2.5 Summary and discussion

In this chapter, a literature review of previous approaches to named entity extraction has been presented. This review has provided three important bases:

- ▷ manually-built extraction rules should be avoided by portable NEE systems and machine learning techniques must be used instead
- ▷ among all machine learning paradigms, statistical methods have been shown to be quite successful when applied on NLP tasks, and
- ▷ maximum entropy models have proved to be a good approach for extracting named entities and they also show benefits for building generic systems

This review and the above bases have also been the foundations on which a number of hypotheses have been proposed with the aim of developing methods that will contribute towards the implementation of a portable, generic NEE system.

The first hypothesis of this thesis is that the inclusion of general, more linguistically-oriented features might help a maximum entropy model to capture useful clues for identifying named entities which cannot be obtained from lexically-oriented features. The main challenge here is obtaining and representing this linguistic knowledge without affecting the portability of the approach. Chapter 4 discusses possible resources for this information and also explains how it can be introduced into a generic named entity extractor.

The second hypothesis propounded here is that a better treatment of exceptionality can be provided to the system by biasing the maximum entropy model to consider further the most similar training examples to the piece of text being analysed. Chapter 5 explains in more detail why this approach could work and surveys the validity of this hypothesis.

The final hypothesis proffered is related to semi-supervised learning. It is evident that the ability of making good use of unlabelled text would increase the portability of any NLP system. However, it is not clear that maximum entropy models are suitable for bootstrapping techniques. In chapter 6, this suitability is analysed and ways of overcoming the difficulties they pose are discussed and tested.

This chapter has also provided a detailed overview of the maximum entropy framework and how it can be applied to the formulation of named entity extraction tasks as classification problems.

Chapter 3

Baseline systems

In this chapter, a new method for analysing target corpora is proposed. This method offers some advantages over previous approaches in the literature such as more detailed information on the performance of an NEE system.

This corpora analysis is then applied on new implementations of two known named entity recognisers which will be used as baseline systems in later chapters.

3.1 Corpora analysis

The experience accumulated over the years on solving NLP tasks indicates that the complexity of a particular application not only depends on how hard the task could be, but also on the complexity posed by the corpus being targeted. Therefore, it is not possible to determine whether an NEE system is obtaining an acceptable level of performance if it is not known how intricate the target texts are.

A common way of getting round this problem is to use the performance of a (very) simple *baseline system* as a minimum bound of effectiveness and against which other approaches can be compared. Thus according to basis three (section 2.3), an NEE system based on maximum entropy models should exhibit a performance that is significantly higher than such a baseline system.

Palmer and Day (1997) conducted an effort to analyse the complexity of the MUC named entity extraction task on several corpora in different languages. In this task there are three types of phrases that report seven classes of named entities (Sundheim 1995): TIMEX (dates and time expressions), NUMEX (money and percentage expressions) and ENAMEX (names of people, locations and organisations).

They found that almost all time and numerical expressions could be captured by a small number of simple patterns, whereas ENAMEX phrases presented a much more challenging job. However, they also discovered that TIMEX and NUMEX phrases accounted for 20-30% of the named entities in the different corpora, thus NEE systems had to be good at recognising names of people, locations and organisations in order to perform well on the task.

A second important result of this analysis is that an NEE system cannot focus on one ENAMEX class, because they significantly varied between languages. For example, a system that optimises the recognition of location names would perform well on the Chinese corpus under analysis, in which this category is a majority, but poorly on the English MUC-6 corpus, in which this category only represents 14.5% of the ENAMEX phrases.

But the most important result reported by Palmer and Day (1997) is that a small number of ENAMEX entities occurred very frequently. In general, they observed that 10% of the named entities represented up to 50% of the ENAMEX phrases occurring in the corpora. Thus, there is an important section of named entities that most likely never occur in any amount of training data.

Following these findings, they conducted an experiment to determine how well a system that memorises ENAMEX entities in the training data could identify named entities in unseen text. The results indicate that the coverage of unseen named entities —as more and more training data is provided— peaks rapidly, leaving a large percentage of phrases uncovered.

Palmer and Day (1997) used these results to estimate a lower bound for the recall of a baseline system which just memorised ENAMEX phrases and utilised simple patterns for TIMEX and NUMEX phrases. They found that this number varied greatly between languages —and possibly across domains— because they presented a different *vocabulary transfer rate*, that corresponds to "the percentage of phrases [labelled as named entities] occurring in the training corpus which also occurred in the test corpus".

Palmer and Day (1997) proposed an estimation for this lower bound of recall that is given in equation 3.1, where N_{NUMEX}, N_{TIMEX} and N_{ENAMEX} are the proportion of NUMEX, TIMEX and ENAMEX expressions in the text respectively — i.e. the percentage of named entity phrases represented by each type; α is the percentage of time and numerical expression that can be captured by simple patterns — which they estimate at 0.95; and T_{ENAMEX} is the vocabulary transfer rate of ENAMEX phrases.

$$((N_{\text{NUMEX}} + N_{\text{TIMEX}}) * \alpha) + (N_{\text{ENAMEX}} * T_{\text{ENAMEX}}) \tag{3.1}$$

Clearly, N_{NUMEX}, N_{TIMEX}, N_{ENAMEX} and T_{ENAMEX} are corpus —or at least language— dependent. As stated above, the first three parameters seem to be relatively constant

across corpora, but the vocabulary transfer rate of ENAMEX phrases differs considerably. For example, the Chinese corpus shows a very high rate of $T_{\text{ENAMEX}} = 0.732$, whereas the French corpus presents a modest one with $T_{\text{ENAMEX}} = 0.236$. Interestingly, the English corpus turned out to be quite difficult: $T_{\text{ENAMEX}} = 0.212$ and $N_{\text{ENAMEX}} = 0.798$, resulting in a lower bound of just 38.4%.

It would be quite interesting to establish whether these numbers are also found in the MUC-7 corpus, which is the one used in this thesis. Some correlation can be expected as both corpora are collections of news articles, though from different sources: the MUC-6 corpus collects Wall Street Journal articles whilst the MUC-7 corpus corresponds to documents from the New York Times.

However, these ideas can be developed further to obtain an indicator of the portability of an NEE system. In effect, Palmer and Day's (1997) study suggests that it is very difficult to provide an extractor with sufficient training examples to cover most of the named entities that will be encountered in the target texts. Therefore if large amounts of manually-labelled training examples are not available, an NEE system which performs well only on seen phrases will not be able to recognise a significant portion of the named entities present in the target texts.

Consequently, it is important that it could be determined whether a named entity extractor is performing well on unseen named entities. The overall recall and precision, or their F-score combination, does not give this information directly. It is necessary then to obtain these indices separately on the basis of how familiar a named entity is.

In section 3.2, a new approach to analysing the complexity of an NEE task is proposed. This approach allows both the estimation of task-independent lower bounds for the recall and the precision of a named entity extractor, and the evaluation of its performance individualised by the familiarity of the named entities to be extracted.

3.2 Analysis of the MUC-7 corpora

This section proposes a new approach to estimate the complexity that the target domain of an NEE task poses. This complexity is reflected as lower bounds for the recall and the precision that a baseline system that memorises the named entities seen during training can obtain. This approach is then applied to the MUC-7 training, dryrun test and formal test corpora.

Although this corpora analysis method is inspired by Palmer and Day's (1997) work, it can also be seen as an extension of the more recent analysis conducted by Whitelaw and Patrick (2003), in which comparisons of different NEE systems were reported separately for named entities that were seen and unseen during training.

Table 3.1: Types of familiarity for named entities in test corpora.

Familiarity type	Description
Unseen	Named entities whose text has not been seen in the training data
Seen	Named entities whose text has been seen in the training data with the same class
Hard	Named entities whose text has been seen in the training but not with the decoding class
Ambiguous	Named entities whose text has been seen in the training data with the same class, but has also been seen with other class(es) or not marked as a named entity

In the approach proposed here decoding named entities, that is named entities in the target texts, are classified into four categories according to the *familiarity* of their (case insensitive) text and their decoding class. Table 3.1 presents the familiarity types for decoding named entities considered in this approach.

Similarly, individual tokens that compose named entities —hereafter NE tokens— are also classified into these categories. This follows the idea of *raw output counting* (Roth and van den Bosch 2002, Daelemans and Osborne 2003), in which each token within a named entity is considered as a whole named entity.

All systems evaluated in this thesis use a fine-grain tokenization scheme, in which most *tokens* are sequences of symbols separated by spaces; however, there are a number of exceptions: *said."*, *tonight's*, *London-based*, etc. It is clear that these sequences need to be divided because they might contain named entities, as the latter ones do for the MUC task. Therefore, the above pieces of text are tokenized as: *said . "*, *tonight 's* and *London - based*.

The token-level analysis provides a different insight into the performance of an NEE system, as it considers partially identified named entities which are normally ignored in the evaluations at phrase level. For instance, suppose that a system produces the following wrong output.

> <ORGANISATION>Western Co.</ORGANISATION> of <LOCATION>North America</LOCATION>.

This output is normally considered just wrong; however, the system has managed a partial recognition of the organisation name. This is the kind of approximation that can

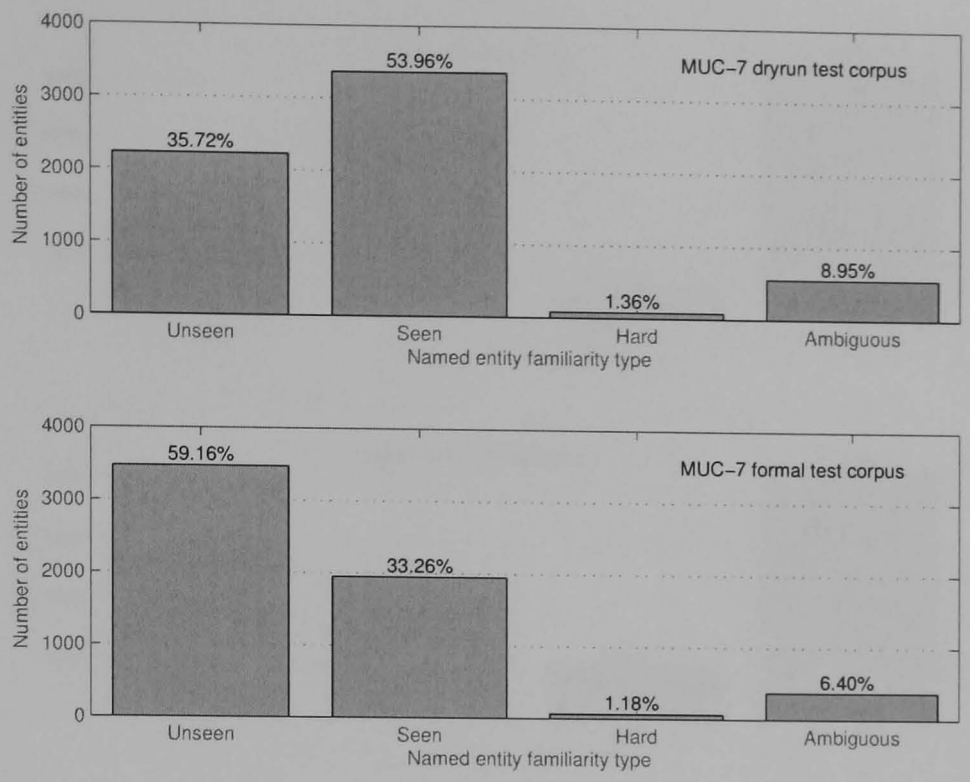


Figure 3.1: Distribution of named entities in the MUC-7 test corpora according to their familiarity.

be captured by counting NE tokens. Moreover, this level of analysis focuses the evaluation on the management of external evidence by the system, as the internal evidence is reduced. For example, it would not be surprising than the phrase *American Airlines* is always seen as an organisation name within a given corpus. However, the tokens *American* and *Airlines* will probably be seen in other contexts as well and a named entity recogniser will have problems in discriminating when these tokens must be extracted from internal evidence only.

Figure 3.1 presents the distribution of named entities in the MUC-7 testing corpora according to their familiarity with respect to the MUC-7 training corpus. In both cases, most named entities are either seen or unseen, constituting around 90% of each corpus. A further analysis of the other 10% suggests that many of them are actually inconsistencies in the annotations, due to disagreement between annotators (e.g. *earlier yesterday* tagged as time and as date) or mistakes (e.g. *Miami* tagged as a date, more than once). This means that the number of actual hard and ambiguous named entities would be even lower in a hypothetical noise-free corpus.

It can be expected that this distribution of named entities will be seen in other domains. This is because an end-user will be interested in the best performance for the system. This includes tuning the named entity extractor to the particular application by providing training data which is *representative* of the target documents. Therefore, many of the named entities included for training will be seen during decoding. In addition, it has been observed that human writers tend to avoid introducing ambiguity in text (Gale et al. 1992, Mikheev, Moens and Grover 1999). Thus ambiguous or contradic-

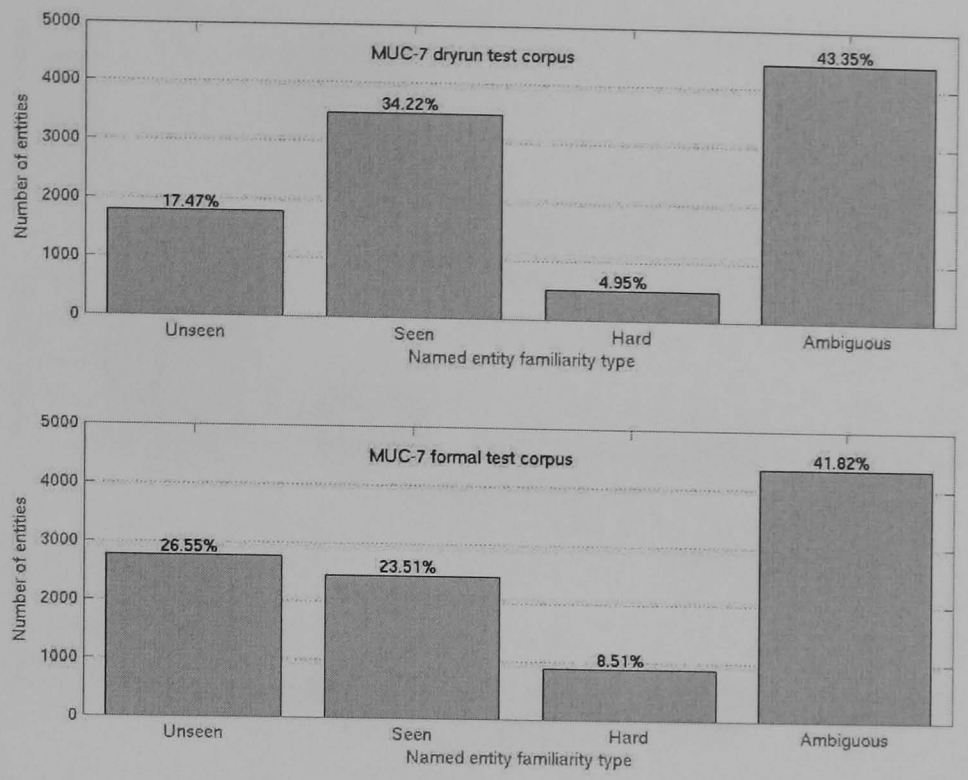


Figure 3.2: Distribution of NE tokens in the MUC-7 test corpora according to their familiarity.

tory named entities will rarely be encountered. On the other hand, training data is by nature limited and target documents will contain named entities which have not been seen during training.

In conclusion, good performance on seen and unseen named entities is a desirable characteristic of NEE systems. This conclusion matches the observation that performance on unseen words is a major factor in the success of an NEE system (Klein et al. 2003, Whitelaw and Patrick 2003).

Figure 3.2 presents the distribution of NE tokens for the MUC-7 test corpora. These results indicate that ambiguous NE tokens are the most frequent and they have to be considered with the unseen and seen familiarities for obtaining good performance at token level. This figure also suggests that an NEE system has to deal with a significant amount of ambiguous tokens to obtain good performance at phrase level.

However, these numbers can be misleading because some named entities appear several times in the text. For example, the named entity phrase *ValuJet* has 241 occurrences in the training corpus: 239 times marked as an organisation name and two times marked as not being part of a name. This named entity phrase is found 244 times in the dryrun test corpus, 243 of them tagged as an organisation name. Thus 244 ambiguous named entity occurrences are being counted just for this phrase. This realisation has encouraged an analysis of named entity phrases —i.e. without considering repetitions— in the corpora, whose results are shown in figure 3.3.

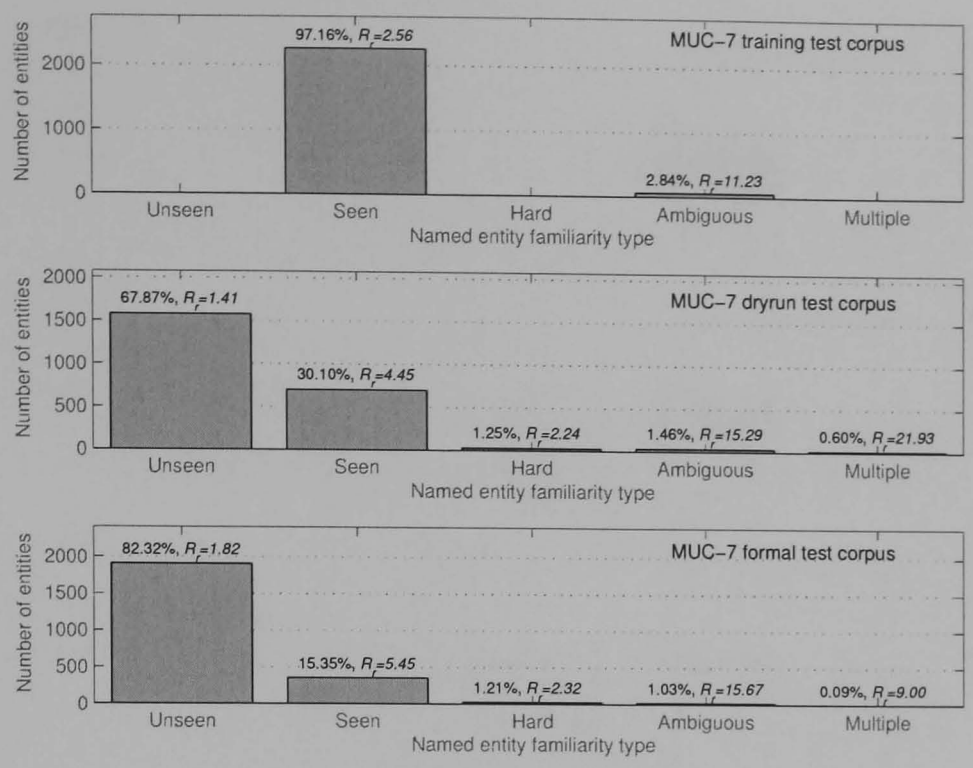


Figure 3.3: Distribution of named entity phrases in the MUC-7 corpora according to their familiarity.

Obviously, in the training corpus there are only seen and ambiguous entities. Only five ambiguous entities have different classes (e.g. *Clinton* is seen once as a location name and 51 times as a person name) and the rest are —beside errors and inconsistencies— one-token named entities which are also frequent common words (e.g. *china*, *march*, *may*, *turkey*, *American*, *brown*, etc.).

Although only 66 out of 2,324 named entities are ambiguous, they exhibit a high *repetition rate* (R_r). Each ambiguous named entity appears on average $R_r = 10.44$ times in the corpus[1], whereas the seen named entities appear just $R_r = 2.57$ times on average.

Figures for the dryrun test corpus are different now: though unseen and seen named entities still count for most of the corpus, unseen named entities are an absolute majority constituting about 65%. The amount of unseen named entities is even higher for the formal test corpus, but this could be explained by the slight change of domain introduced in this collection of articles.

Some named entities present multiple familiarity types because they appear in the corpus with different classes. For example, the text *8 p.m.* is found in the dryrun test documents with the class time and the class date; but in the training corpus this text was seen tagged only as a time named entity. Thus, this phrase is considered seen when the decoding class is time, but hard when the decoding class is date. In general, most multiple familiarity type named entities are introduced by inconsistencies in the annotation within the corpus, as this example has been.

[1]This rate does not consider the cases in which the word is not part of a name. Including these occurrences the frequency ratio would rise to 23.56.

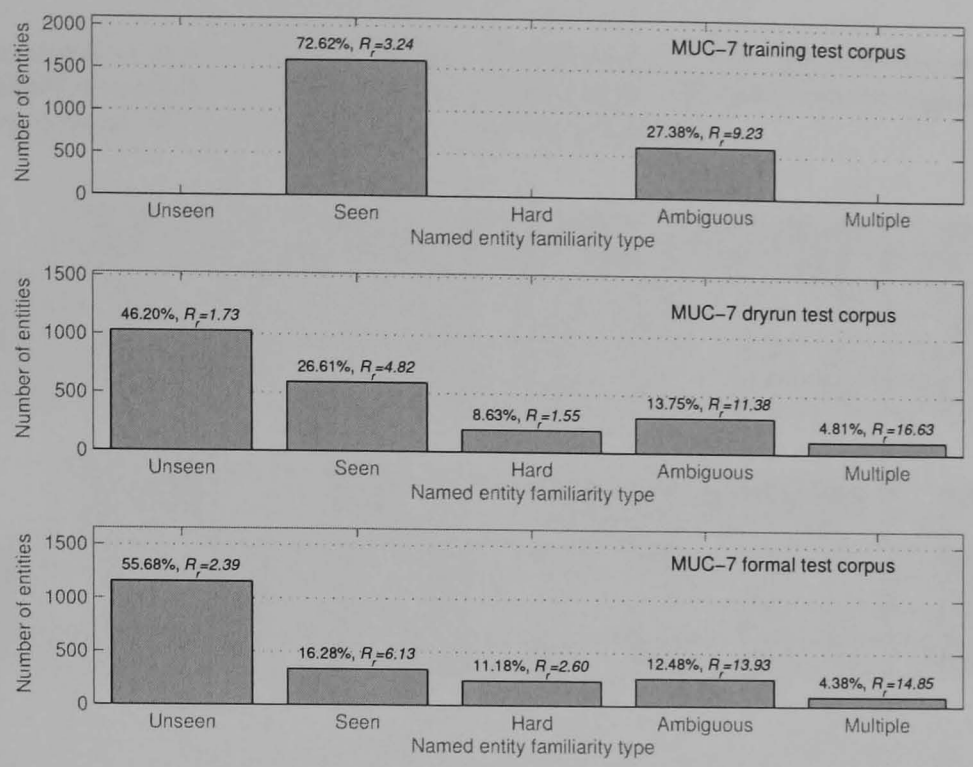


Figure 3.4: Distribution of NE tokens as named entity phrases in the MUC-7 corpora according to their familiarity.

Interestingly, the repetition rate of ambiguous named entity phrases in the dryrun test corpus is consistent with the rate found in the training corpus. That is to say that though only a fraction of named entities are ambiguous, they are encountered much more frequently than seen entities (3:1 approximately). Remembering that many ambiguous named entities are mainly introduced by noise in the annotations, it could be suggested that highly frequent named entities are somehow attracting —if not generating— lapse of concentration from the annotators. Taking care of these named entities during annotation might result in a significant reduction in the noise in training corpora. Although the trend of noise among seen named entities cannot be automatically analysed, they seem to be much less frequent ($R_r = 2.6$ approximately) and mis-annotations in this category result in less damaging noise.

As mentioned above, with the change in domain introduced by the MUC-7 formal test corpus, the number of unseen named entities rises to more than 80% of the cases, but the repetition rate follows the same pattern, that is unseen named entities are less frequent than seen ones, which in turn are less frequent than ambiguous named entities (2:5:16 approximately).

Although unseen events are repeated much less, their absolute numbers make them important for an NEE system in both test corpora. This is again consistent with the findings in Klein et al. (2003) that performance on unseen words is what makes the difference between current NEE approaches.

Figure 3.4 presents the distribution of NE tokens when they are considered as named

Table 3.2: Distribution of named entities in the MUC-7 test corpora according to their phrase type (TIMEX, NUMEX & ENAMEX) and familiarity type (UNSEEN, SEEN, HARD & AMBIGUOUS). Figures are separated for named entity phrases (NEP) and named entity occurrences (NEO).

MUC-7 dryrun test corpus

	UNSEEN		SEEN		HARD		AMBIGUOUS		TOTAL	
	NEP	NEO	NEP	NEO	NEP	NEO	NEP	NEO	NEP	NEO
TIMEX	439	499	190	791	10	11	10	59	649	1,360
NUMEX	62	71	25	37	0	0	0	0	87	108
ENAMEX	1,087	1,657	466	2,536	34	74	27	499	1,614	4,766
TOTAL	1,588	2,227	681	3,364	44	85	37	558	2,350	6,234

MUC-7 formal test corpus

	UNSEEN		SEEN		HARD		AMBIGUOUS		TOTAL	
	NEP	NEO	NEP	NEO	NEP	NEO	NEP	NEO	NEP	NEO
TIMEX	635	812	154	602	14	24	10	41	813	1,479
NUMEX	213	262	26	60	3	4	0	0	242	326
ENAMEX	1,066	2,400	178	1,291	15	41	14	335	1,273	4,067
TOTAL	1,914	3,474	358	1,953	32	69	24	376	2,328	5,872

entity phrases — i.e. without considering different occurrences. It can be observed that the proportions between familiarity types change. Although unseen tokens are still a majority, ambiguous and hard NE tokens are not tiny fractions of the corpora now but represent around a third of all NE tokens. Interestingly, the proportion of seen NE tokens is relatively the same as that for seen named entities with 15-25%.

Despite this change, the trend of the repetition rate remains: unseen and hard tokens are less frequent than seen tokens, which in turn are less frequent than ambiguous ones, though the differences are also reduced slightly (1:3:5 approximately).

All this information collected for the MUC-7 corpora allows an estimate of the performance of a baseline system similar to the one defined by Palmer and Day (1997). Table 3.2 presents the distribution of named entities at phrase level on the MUC-7 test corpora, in which named entities have also been classified according to their familiarity with respect to the MUC-7 training corpus, with and without considering different occurrences. There are actually 2,326 named entity phrases in the dryrun test corpus, which form 2,350 (NE phrase, NE class) pairs that occur 6,234 times. In the formal test corpus, there are 2,323 named entity phrases which form 2,328 (NE phrase, NE class) pairs that occur 5,872 times in the documents.

In terms of Palmer and Day's (1997) estimations, ENAMEX phrases represent 76.45% of the total named entities in the dryrun test corpus, and the remaining 23.55% corresponds to TIMEX and NUMEX phrases. Palmer and Day's (1997) vocabulary transfer rate can be obtained by adding the percentage of seen, hard and ambiguous ENAMEX named entities of table 3.2. Thus, the vocabulary transfer rate for the dryrun test corpus is 65.23%. Note that the proportion of phrases by type in this corpus follows the observations of Palmer and Day (1997). However, the vocabulary transfer rate is much higher here than in the MUC-6 corpus studied by Palmer and Day (1997), which was

estimated in 21.2% only. This confirms that some corpora pose more difficulties than others, even for the same extraction task and in the same language.

Similarly, it can be established that in the formal test corpus, 69.26% of the named entities correspond to ENAMEX phrases and 30.74% to TIMEX and NUMEX phrases, and that it presents an ENAMEX vocabulary transfer rate of 40.99%. Again, the proportion of named entities by type corresponds with the results of Palmer and Day (1997), but the transfer rate is higher.

With these figures, it is possible to determine lower bounds for a baseline system as defined in equation 3.1 for both MUC-7 test corpora:

$$\begin{aligned} 0.2355 * 0.95 + 0.7645 * 0.6523 &= 72.24\% \quad \text{(dryrun)} \\ 0.3074 * 0.95 + 0.6926 * 0.4099 &= 57.59\% \quad \text{(formal)} \end{aligned}$$

It must be noticed that these are bounds for recall only and that it assumes that 95% of TIMEX and NUMEX expression can be captured by using simple regular patterns. However, this approach to measuring complexity is for the extraction task as defined in the MUC conferences only and it could be argued that assuming 95% recall on TIMEX and NUMEX expression is overoptimistic and even unfair to systems that are committed to using as little human intervention as possible.

Fortunately, the information provided in table 3.2 permits better estimates for a hypothetical system that only memorises the named entities which it sees in the training documents. Such a baseline system would extract all seen named entities correctly but, supposing it would abstain from classifying ambiguous named entities, it would get nevertheless hard named entities wrong. Thus, recall can be estimated as:

$$\begin{aligned} \frac{3{,}364}{6{,}234} &= 53.96\% \quad \text{(dryrun)} \\ \frac{1{,}953}{5{,}872} &= 33.26\% \quad \text{(formal)} \end{aligned}$$

On the other hand, precision is limited by the portion of the named entities identified in the decoding text which are classified with the correct class, which corresponds to:

$$\begin{aligned} \frac{3{,}364}{3{,}364+85} &= 97.54\% \quad \text{(dryrun)} \\ \frac{1{,}953}{1{,}953+69} &= 96.59\% \quad \text{(formal)} \end{aligned}$$

A lower bound for the F-score can also be given by applying formula 1.1 with parameter $\beta = 1$: 69.48% for the dryrun test corpus and 49.48% for the formal test corpora respectively. These bounds are quite high for such a simple baseline technique and improving on this performance will probably present a difficult challenge. The main room for improvement seems to be the low recall that the hypothetical system would obtain. However, as explained in previous chapters, increasing recall has always been followed by a drop in precision. The challenge then will be to increase recall in a greater proportion than the corresponding fall in precision.

Although a system that just memorises named entities has been used as reference (for example in the CoNLL conferences, in which all participant systems outperformed such a baseline system), it could be argued that the named entity extraction technology is too developed now to consider such naïve methods.

A more realistic approach would be to compare the performance of a new NEE system with existing systems that are known to perform well. Unfortunately, it is not easy to find good NEE systems freely available.

One alternative would be to use domains for which results are known, but these are limited to very few domains. Moreover, it is difficult to compare results from a new NEE system with previous works because even if the systems are using the same corpora —and therefore the same domain— the pre-processing tools may vary and errors from this stage do affect the overall performance of these systems. In addition, the exact training resources which each system utilises are very difficult to replicate.

In conclusion, the best way of comparing new results against the state-of-the-art of the technology is to implement one or more known systems and use them as baseline systems. In this way, it can be ensured that both training data and early NLP steps are shared and the same.

3.3 Baseline systems

Following the discussion in the previous section, two known good statistical approaches have been selected and implemented as baseline systems. There are two additional reasons for implementing these baseline systems:

- the performance obtained by the approaches proposed in this thesis will be individualised by familiarity type. This information cannot be obtained from published results as they are made only in terms of overall recall and precision
- they can be used to confirm the basis that statistical machine learning methods are successful for named entity extraction tasks

In the following sections, these systems are discussed in more detail.

3.3.1 Nymble

The first baseline system implemented is a version of Nymble (Bikel et al. 1997). This approach has been chosen because:

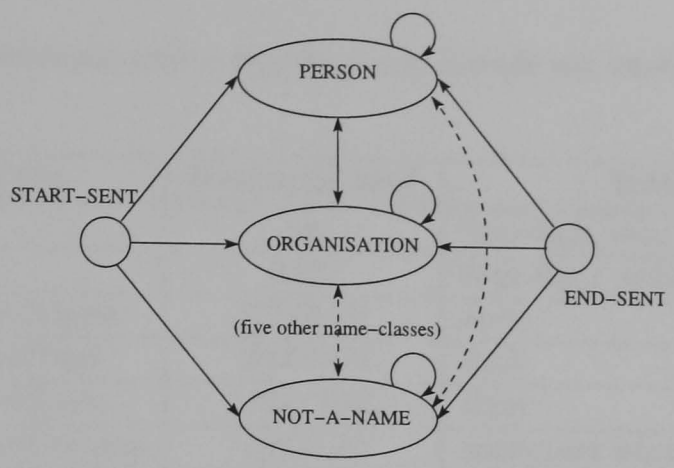


Figure 3.5: The conceptual Hidden Markov Model used in Nymble. Adapted from Bikel et al. (1997).

1. it uses a 100% learning approach (with no external lexical resources)
2. it is extremely simple
3. its performance is very good
4. the approach seems to be well explained in the literature

Nymble uses an ergodic Hidden Markov Model (Rabiner 1989) with eight regions —also called *name-classes*—, one for every type of MUC named entity to recognise (Chinchor 1998a) plus a default not-a-name region for words which are not part of any entity name (figure 3.5). Consequently, it identifies person names, organisation names, location names, dates, time expressions, money expressions and percent expressions. In addition to these, the model includes two special states which represent the beginnings and ends of sentences.

Within each region, Nymble uses a statistical bigram language model in which every state can emit one word. Therefore, each name-class has $|V|$ states and $|V|^2$ transitions, where V is the vocabulary recognised by the system. Unlike normal Hidden Markov Models, the transition of states and emission of symbols follows a three-step procedure:

1. Select a name-class NC conditioned on the previous name-class and the previous word, that is
$$\Pr(NC|NC_{-1}, w_{-1}) \tag{3.2}$$
2. Generate the first word within the current NC conditioned on the current and previous name-classes, that is
$$\Pr(<w,f>_{first}|NC, NC_{-1}) \tag{3.3}$$

Table 3.3: Orthographic features as used in Nymble plus an example and intuition behind them. Adapted from Bikel et al. (1997).

Word-feature	Example text	Intuition
twoDigitNum	98	two-digit year
fourDigitNum	2002	four-digit year
containsDigitAndAlpha	I2534-W	code
containsDigitAndDash	20-01-02	date
containsDigitAndSlash	1/20/2002	date
containsDigitAndComma	5,000.00	monetary amount
containsDigitAndPeriod	1.00	monetary amount, percentage
otherNum	283845	other number
allCaps	IBM	organisation
capPeriod	L.	person name initial
firstWord		less useful capitalisation
initCap	John	possible name
lowerCase	is	possible not a name
other	,	all other words

3. Generate all subsequent words inside the current name class conditioned on the previous word, that is

$$\Pr(<w,f>|<w,f>_{-1}, NC) \tag{3.4}$$

As can be seen from above, a word in Nymble is actually a pair: the lexeme and an orthographic feature, which are denoted $<w,f>$. Nymble uses fourteen disjoint orthographic features (showed in table 3.3) which describe some lexical characteristics of tokens that should help the model to recognise names. Only one feature is assigned to each word, so there is a precedence scheme among the features (which can be seen in this list). Thus a word which begins with a capital letter but is starting a sentence will have associated the *firstWord* feature rather than the *initCap* feature. This precedence is based on the intuitive fact that a capitalised word at the beginning of a sentence will provide less evidence for being a name than in the middle of a sentence.

Nymble also introduces two *magical* words: $<+begin+,other>$ to compute the likelihood for a word being the first word of its class-name, and $<+end+,other>$ to compute the probability for any word being the final word of its class-name.

As with any n-gram language model, it is unrealistic to expect that Nymble will be provided with all possible bigrams from the training data. This is overcome by collecting statistics for unknown words by manipulating the system's vocabulary and backing off to models based on incomplete information. Table 3.4 summarises the backing-off strategy as used by Nymble.

Table 3.4: Nymble's back-off/smoothing scheme. Adapted from Bikel et al. (1997).

Name-class	First words	Subsequent words
$Pr(NC\|NC_{-1}, w_{-1})$	$Pr(< w, f >_{first} \|NC, NC_{-1})$	$Pr(< w, f > \| < w, f >_{-1}, NC)$
$\vdots$	$\vdots$	$\vdots$
$Pr(NC\|NC_{-1})$	$Pr(< w, f > \| < +begin+, other, NC)$	$Pr(< w, f > \|NC)$
$\vdots$	$\vdots$	$\vdots$
$Pr(NC)$	$Pr(< w, f > \|NC)$	$Pr(w\|NC) \cdot Pr(f\|NC)$
$\vdots$	$\vdots$	$\vdots$
$\frac{1}{\text{number of name-classes}}$	$Pr(w\|NC) \cdot Pr(f\|NC)$	$\frac{1}{\|V\|} \cdot \frac{1}{\text{number of word features}}$
	$\vdots$	
	$\frac{1}{\|V\|} \cdot \frac{1}{\text{number of word features}}$	

In addition, Nymble uses the back-off models for smoothing the top-level model by assigning the appropriate weight to each model and its immediate back-off model.

Appendix A presents a detailed analysis of Nymble and explains how it has been implemented. Because this version includes only the main features of the original system and has not been tuned to any specific task, it has been named siNymble (simple implementation of Nymble). This appendix also includes a walk-through example with siNymble.

3.3.2 MENE

The other baseline named entity extractor chosen is the MENE system (Borthwick 1999). This system utilises maximum entropy statistical modelling (Berger et al. 1996, Della Pietra et al. 1997, Ratnaparkhi 1998) to capture relevant features in free text, which are used later to predict occurrences of named entities. MENE was one of the systems which participated in the latest MUC conference (MUC 1998) with good results.

The training information provided to MENE is a pool of features of different nature:

- ▷ lexical features (i.e. the lexemes) of surrounding words
- ▷ orthographic features, such as the type of lexeme (number, word, symbol) and capitalisation properties
- ▷ section features, which discriminate between sections in the document (e.g. headlines, text)
- ▷ dictionary features, which indicate whether a lexeme is contained in external dictionaries of first names, corporate names, etc.

- ▷ reference resolution features, which relate different —including partial— occurrences of a sequence of lexemes

Thus, each token in a training document has associated —at least some of— these features and a tag which indicates its named entity feature. If c is one of the named entity classes to identify, then any token could be associated with the tags c_start, $c\_continue$, c_end, c_unique or not_ne indicating that the token is starting, continuing or ending a named entity of class c, the token constitutes a one-word named entity of class c or the token is not linked to any named entity. According to the MUC-7 definition of the named entity task (Chinchor 1998a), 29 tags are used to represent the options for the seven target named entity classes and the not-a-named-entity case.

During decoding, MENE assigns each token with the probabilities of being associated with one of these 29 tags. Taking the highest probability could result in invalid sequences of tags. Therefore, Borthwick (1999) uses a Viterbi search to avoid incompatible assignments and obtaining instead the most probable valid sequence of tags.

The version of MENE presented here, named LexMENE, uses only the first three types of features, that is lexical features, orthographic features and section features. Borthwick (1999) found that lexical and orthographic features, as well as the coreference resolution features, make the most important contributions to the performance of the approach. The other kinds of features, namely section and dictionary features, did not change the accuracy of the system significantly.

MENE's orthographic features are similar to Nymble's, presented in table 3.3. There are some differences, which can be summarised as

- ▷ the set of features is slightly different:
  - ▷ the feature *onlyDigits* is added for numbers such as *5*, *321*, *2000*, etc.
  - ▷ the feature *internalCapitalisation* is added for tokens like *EasyJet*, *McCarthy*, etc.
  - ▷ the feature *otherNum* is replaced by the feature *validNumber*
  - ▷ features *containsDigitAndDash*, *containsDigitAndSlash*, *capPeriod*, *firstWord* and *other* are not used
- ▷ MENE allows more than one orthographic feature to be fired for a given token

Consequently, LexMENE follows these features as well as the approach for unknown words used by MENE: in both phases of the algorithm —namely training and decoding— words that appear less than three times in the corpora are replaced by the token *UNK*.

The differences between MENE and LexMENE can be summarised with the following points:

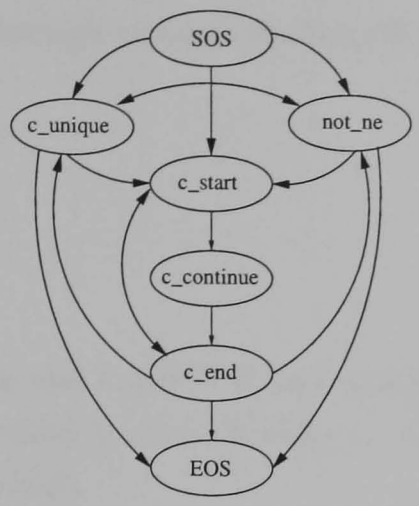


Figure 3.6: A schematic view of the the Viterbi search for a given named entity class c.

- ▷ MENE's dictionary features are not used by LexMENE
- ▷ the highly desirable coreference features of MENE are not present in LexMENE; this is because a reasonably good coreference resolution system is not cheap and it would require a considerable investment of time and resources
- ▷ Borthwick (1999) used the Maximum Entropy Modelling Toolkit (Ristad 1998) for his system, whereas LexMENE uses the opennlp.maxent package (Baldrige and Bierner 2001). The former is implemented in C++ and the latter is implemented in Java. There should be no significant impact from this variation, though the Java package implements the Generalised Scaling Algorithm (GIS) (Darroch and Ratcliff 1972, see algorithm 2.2) that is a special case of Improved Iterative Scaling (Della Pietra et al. 1997, see algorithm 2.3), which is also the algorithm implemented in the C++ toolkit used by MENE

Borthwick (1999) did an evaluation of MENE using only lexical, orthographic and section features, obtaining F-score 91.71% for the dryrun test and F-score 83.38% for the formal test corpora respectively. However, these figures were obtained with 350 training documents (321,000 tokens) of which LexMENE only has a hundred (which are translated into 85,837 tokens). Consequently, significantly lower scores can be expected for LexMENE.

As mentioned before, Borthwick (1999) identified the need for a Viterbi search among the probabilities for named entity tags to avoid invalid sequences of named entity tags. This search follows the model presented in figure 3.6 and determines the best sequence of tags for each sentence of a document. Bubbles represent named entity tags and the special start-of-sentence (SOS) and end-of-sentence (EOS) states. Edges represent valid transitions with uniform probability from the source state. The probabilities for each named entity tag are used as emission probabilities.

Appendix B presents a walk-through example for LexMENE to help in putting all these procedures into context.

3.4 Evaluation

In this section, both siNymble and LexMENE are evaluated and their performance is compared with the results reported by their developers. All experiments use the corpora released for the MUC-7 conference.

Although some comparison in terms of recall and precision is included in this section for the baseline systems, comparisons will normally be made in terms of their combining F-score throughout this thesis, so that simple, but enlightening figures for each type of named entity can be obtained.

3.4.1 The scoring program

Unlike Bikel et al. (1997) and Borthwick (1999), who utilised the MUC scoring program (Douthat 1998), an adaptation of the scoring program from the CoNLL conferences is used in this thesis. This is an important difference because the CoNLL scoring software is less generous than the MUC scorer: the latter allows alternative classes and alternative strings. Thus some named entities that are considered wrong by the scorer used here would be counted as correct by the MUC software. For example, the named entity *Kennedy Space Center* can be classified as both an organisation or a location name under the MUC perspective, but only the category organisation is accepted by the CoNLL scorer.

The selection of the CoNLL scorer is based entirely on practical reasons. This scorer, being a script in Perl rather than a program of several modules in C as the MUC scorer, is much easier to modify and is platform independent. Consequently, the CoNLL scorer script was adapted so that it could provide the performance of an NEE system detailed by the different familiarity type of the named entities, in addition to the existing information by class.

3.4.2 Results for siNymble

SiNymble has been trained on the MUC-7 training corpus. Three versions of siNymble have been tested resulting from slightly different interpretations of the description given in Bikel et al. (1997).

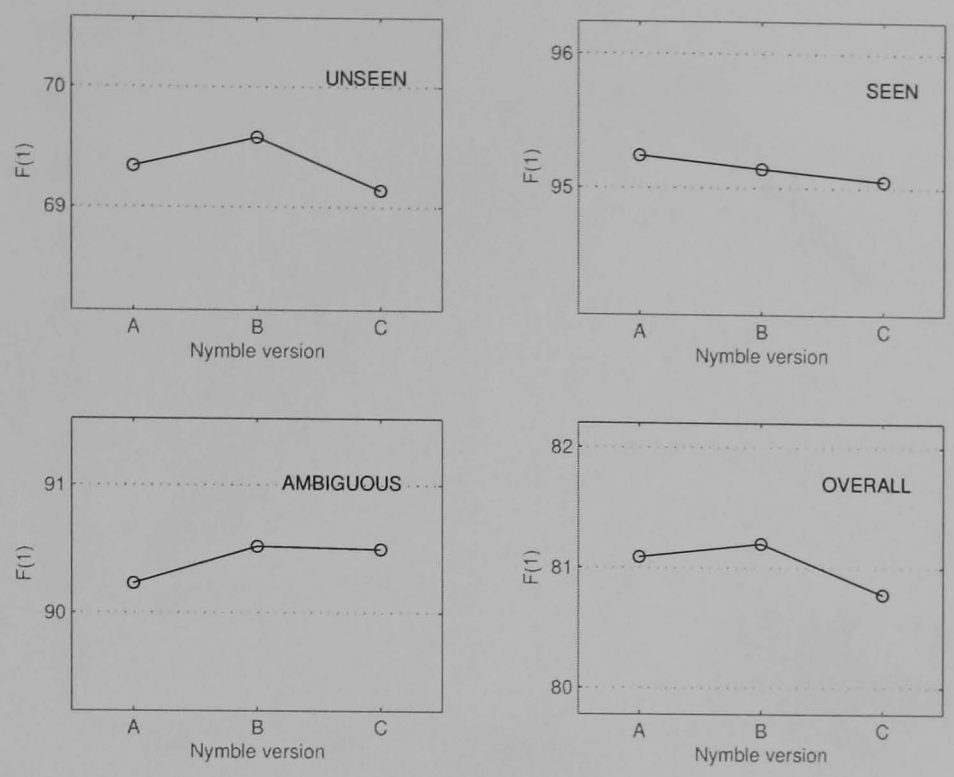


Figure 3.7: Experiments with three versions of siNymble. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

Version A follows the precedence of orthographic features —binary lexical features in MENE's terms— strictly as presented in table 3.3. Therefore, tokens with feature *initCap*, *lowerCase* or *other* are changed to the feature *firstWord* when they are starting a sentence.

Version B only makes this change when the token starting the sentence fires the orthographic feature *initCap*. In addition and for this purpose only, a sentence is not considered started until a token which does not fire the feature *other* is found. For example, in the text *(Figure 1...* the first token of the sentence is *Figure* —rather than the parenthesis— and its orthographic feature will be changed from *initCap* to *firstWord*.

Finally a version C of the approach has also been evaluated. This version arises from the way in which siNymble's implementation manages words and tokens. SiNymble separates multi-token words so that named entities occurring within this type of word can be identified. Consider the word *Atlanta-based*, which naturally contains the location Atlanta. The implementation used here presents these words as three different tokens: *Atlanta - based*; consequently they will fire the orthographic features *initCap*, *other* and *lowerCase* respectively. In version C, this does not happen and the orthographic feature fired by the word as a whole is given to each token, resulting in the features *initCap*, *initCap* and *initCap* being assigned instead.

Figure 3.7 presents the performance scored by each version. Although there are no significant differences, version B obtained the best (overall) results, which is not surprising as the modification intuitively makes sense.

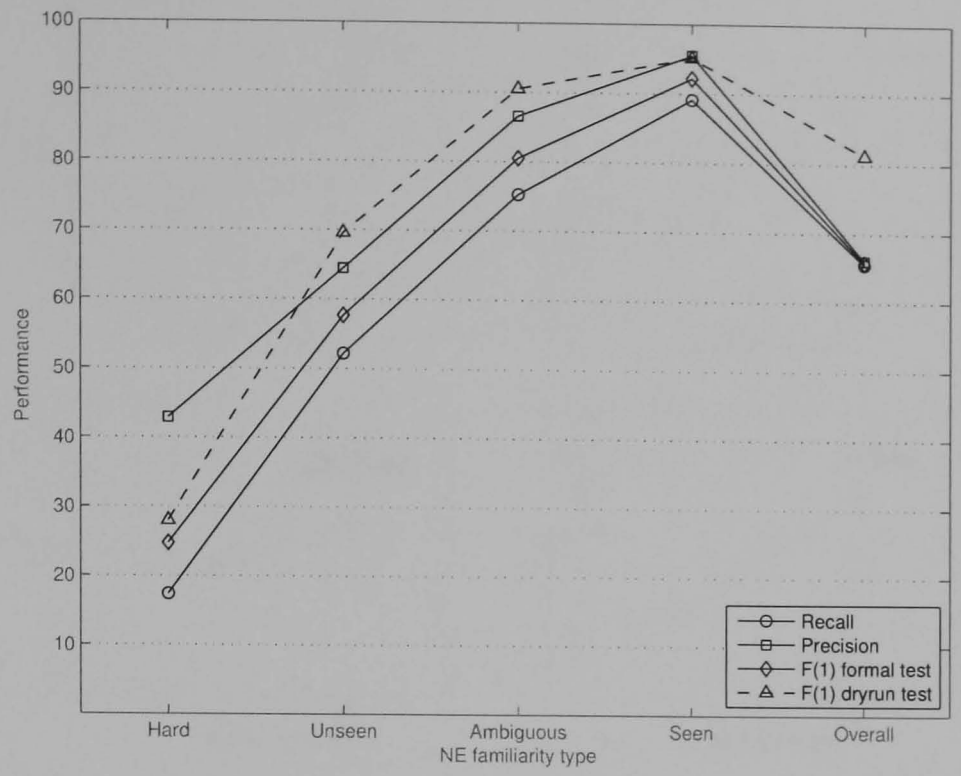


Figure 3.8: Experiments with siNymble (version B). Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

This performance is inferior to that reported by Nymble's creators. However, much of the difference is due to the change in the scorer program used. A similar version of siNymble-B gets an F-score over 85% with the MUC scorer, which is consistent with the results presented in Bikel et al. (1997) for the same amount of training material.

Now siNymble can be evaluated on the MUC-7 formal test corpus by fixing the version to B. Figure 3.8 presents the performance of siNymble version B on this test corpus. It can be seen that recall is always much worse than the precision obtained by the system, and that the overall performance is much worse on this corpus than on the dryrun test with a drop in F-score of about 15%.

The change in domain introduced by the formal test corpus does not degrade the performance of siNymble much on hard and seen named entities —about 3% F-score— but a significant decrease is observed for unseen and ambiguous named entities, with F-scores 12% and 10% lower respectively. Because unseen named entities are far more numerous than ambiguous ones, they are the main factor in the poor performance of this baseline system for this corpus.

Analysing the mistakes that siNymble makes, the same conclusion reported by Bikel, Schwartz and Weischedel (1999) can be reached. Consider the following sentence

> The Turkish company, Birgen Air, was using the plane ...

SiNymble recognised *Birgen Air* as a location rather than an organisation as it is marked in the corresponding key document. The reason is that the word *Birgen* is unknown and

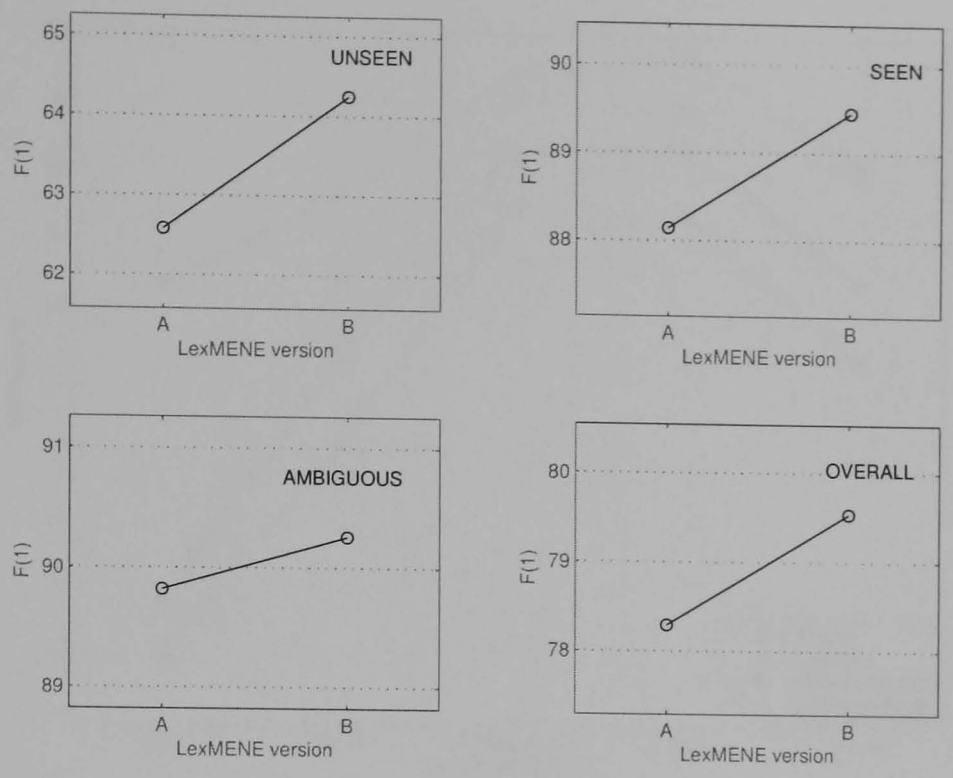


Figure 3.9: Experiments with two versions of LexMENE. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

the word *Air* has been seen within many airport names, which are normally tagged as locations. The problem seems to be that because siNymble uses bigrams for modelling the language, it is incapable of detecting the context word *company* —as the immediate previous token is a comma— which suggests that the entity is actually an organisation. Therefore, one way of improving this kind of approach would be to allow larger contexts to be considered.

3.4.3 Results for LexMENE

LexMENE has also been trained on the MUC-7 training corpus for this evaluation. As with siNymble, there is more than one possible implementation due to different interpretations of Borthwick's (1999) intentions.

Version A strictly follows the description presented in 3.3.2. Nonetheless, MENE does not use the *firstWord* feature to distinguish the possibly irrelevant capitalisation of a token starting a sentence, "believing that MENE could make these judgements from the surrounding lexical context" (Borthwick 1999). However, with version A there is little lexical context for determining that a token is starting a sentence, namely the absence of features for surrounding tokens. This leads to version B, in which lexical features explicitly indicate the non-existence of previous —or following— words by taken the value *NONE*.

Figure 3.9 presents the performance of these two versions of LexMENE according to the familiarity of the named entities in the MUC-7 dryrun test corpus. Version B

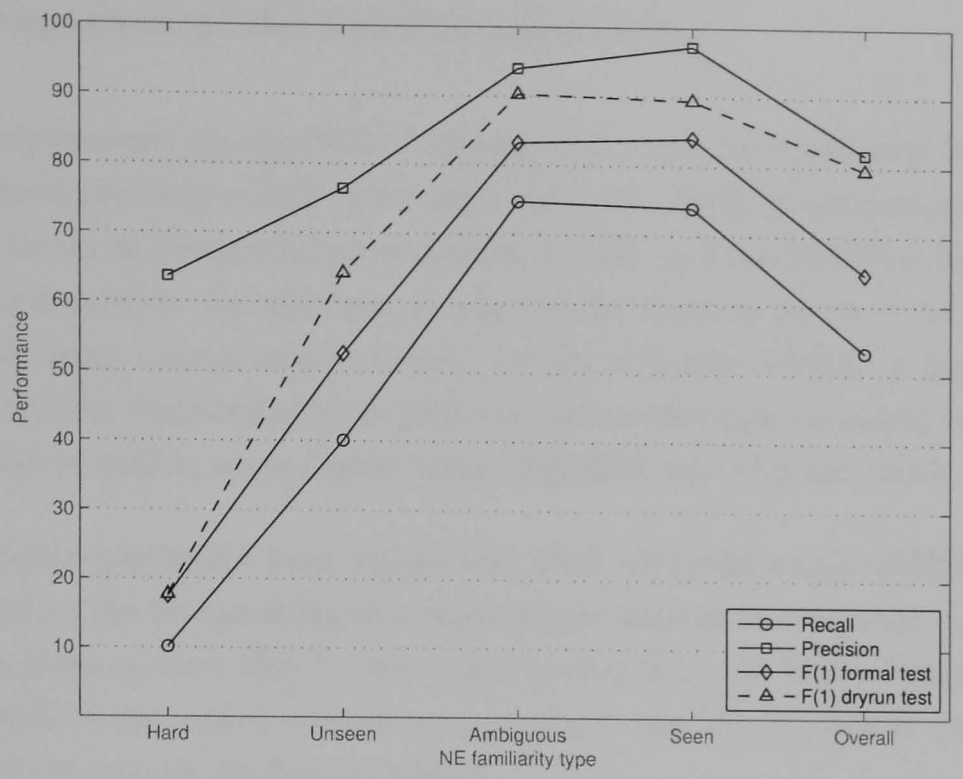


Figure 3.10: Experiments with LexMENE (version B). Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

outperforms version A in every category (except on hard named entities, not shown in this figure, in which they get the same results).

Once more, these figures are lower than the results reported by Borthwick (1999). Again this difference can be explained by the change in the scorer program and the amount of extra training material used in the original experiments. This confirms the idea that comparing NEE systems is a tricky task unless exactly the same resources and pre-processing are used.

Figure 3.10 presents the evaluation of LexMENE (version B) on the MUC-7 formal test corpus. Interestingly LexMENE shows a similar behaviour to siNymble, that is a drop in performance of about 15% with respect to the results on the dryrun test documents. However, it is even more evident in this figure that recall is much worse than precision and that it is this variable that is negatively affecting the F-score line.

Again the change in domain on the formal test corpus does not degrade results on hard named entities, with a drop of less than 1% of the F-score. However, the performance on seen named entities shows a more important drop than in siNymble of about 5%. Nonetheless, the most significant decrease is observed for unseen and ambiguous named entities too, with F-scores 12% and 7% lower respectively. Therefore, unseen named entities are again responsible for most of the decline in performance of this system for the formal test corpus.

3.4.4 Comparison of the baseline approaches

The above experiments on the MUC-7 formal test corpus indicate that both baseline systems perform similarly overall. However, this is the result of compensatory abilities: siNymble is better at recognising seen named entities and LexMENE is better on ambiguous named entities. In addition, neither of the systems seems to be particularly good at recognising unseen name entities, though siNymble obtains a higher F-score. LexMENE exhibits diparatedisparate performance on this type of named entity: it gets relatively good precision, much higher than siNymble, but very low recall.

Moreover, these experiments have shown that both siNymble and LexMENE are negatively affected by the introduction of unseen named entities in the target corpus. Their performance drops around 15% F-score when moved from the MUC-7 dryrun test corpus, where 36% of the named entities are unseen, to the MUC-7 formal test corpus, in which 59% of the named entities are unseen.

Nonetheless, these results confirm that statistical approaches, such as the hidden Markov model used by siNymble and the maximum entropy model —in combination with a Viterbi algorithm for the final labelling— used by LexMENE, are powerful tools for predicting named entities in free text. Both systems largely outperform the hypothetical baseline system: 7-8% higher in the dryrun test and 10-12% higher in the formal test respectively.

However, LexMENE has an important advantage over siNymble. In section 3.4.2, it was suggested that siNymble's bigrams can fail in capturing complex names and contexts and, consequently, broader patterns might help it to recognise more unseen named entities. But adding this information might not be trivial due to the generative nature of the approach and the sparseness of the training data.

In fact, Bikel et al. (1997) recognise the running time speed of the system as a key factor in the success of their approach, because it provided "a rapid code-compile-train-test cycle" that allowed them to perform "numerous experiments" that were "key to improving performance". Adding new information to siNymble would require a similar process of searching the right model for the task, which has evident disadvantages for the portability of the system.

It would be much easier to extend LexMENE by making use of the ability of maximum entropy models to manage information from different sources which might even be overlapping or irrelevant, making this approach an ideal candidate for evaluating the contribution of adding more linguistically oriented knowledge as well as the introduction of other machine learning techniques. These ideas validate the third basis of this thesis (see section 2.3).

3.5 Summary and discussion

In this chapter a new approach for estimating the complexity of a named entity extraction task has been proposed. This approach classifies named entities according to their familiarity into seen, unseen, hard and ambiguous. Knowing the amount of each of these types of named entities occurring in a given corpus allows the estimation of lower bounds for the recall and the precision of a baseline system that memorises named entities during training. This classification is also useful to obtain more detailed information on the performance of an NEE system.

Two simple implementations of statistical approaches to named entity extraction were also presented and evaluated following the familiarity classification of named entities. They have shown that lexical and orthographic features provide useful information for solving the MUC extraction task as they broadly outperform the hypothetical baseline system on both test corpora.

Finally, a comparison of both baseline systems has determined that they perform similarly overall but that the approach based on the maximum entropy framework has advantages related to the portability of the system.

Chapter 4

More linguistically informed MENE

In this chapter a system named MOLI MENE (More Linguistically Informed MENE) is presented. This system is an extension of the LexMENE system discussed in chapter 3, which uses the lexical characteristics that LexMENE utilises, but also includes information drawn from a general lexical reference resource, namely WordNet®, and the syntactic structure of phrases.

4.1 MOLI MENE

Several NEE systems have shown that maximum entropy models are a good choice for identifying named entities: the top three systems for English and the top two systems for German in the latest CoNLL used the maximum entropy framework (Tjong Kim Sang and De Meulder 2003). Therefore, it makes sense to extend LexMENE —one of the baseline systems— by introducing external lexical resources and syntactic information that might provide new, useful features to the maximum entropy model for the extraction task. This extension will be referred to as the "More Linguistically Informed Maximum Entropy for Named Entities", or MOLI MENE for short.

As a starting point, it is reasonable that MOLI MENE —being an extension of LexMENE— would include the same of features of this later system as its basic features, that is lexical features, orthographic features and the zone feature. However, it has been argued that orthographic features are domain-dependent (Mikheev, Grover and Moens 1999), an argument that can be extended to the zone feature.

But it could be argued that this is not exactly the case. It is true that orthographic features —such as those presented in table 3.3— help LexMENE to identify named

Table 4.1: The set of orthographic features in MOLI MENE. It also shows which of these features also exist in LexMENE.

Description	MENE	Example Text	Intuition
2-digit number	yes	96, 01	two-digit years
4-digit number	yes	1999, 2001	four-digit years
only digits	yes	5, 502, 1999	numbers
letters & numbers	yes	F14	codes
number with comma	yes	2,000	money
number with period	yes	4.5, 45.21	money; percentage
number with dash	no	01-03-96	dates
number with slash	no	1/4, 01/03/96	fractions, dates
valid number	yes	-3.5, .12, 30	any number
all capitals	yes	IBM	organisations
initial capital	yes	Jones, Intel	part of a name
mixed capitalisation	yes	AirJetter	organisations
uncapitalised	yes	the, cat, is	not part of a name
symbol	no	%, $, '	not a word
mixed characters	no	's, 'nt	contractions
abbreviation	no	St., Mr., Sen.	abbreviations

entities for the task defined in the MUC competitions. However, they might also help with recognising other types of named entities which present similar patterns. For example, a feature that indicates that a token is composed of numbers separated by dashes would be useful to identify dates in the MUC extraction tasks, but this would also help with recognising product codes in another application. Moreover, if no dates or product codes are part of the task, orthographic features for identifying these types of tokens might help in discriminating text which is unlikely to be relevant for extraction.

Therefore, if a set of orthographic features is sufficient to capture useful internal evidence (McDonald 1996, discussed in section 2.1) of target named entities —or non-named entities— it would be useful for many real-world domains. The same reasoning is valid for the zone feature which —not restricted to a fixed set of sections in the domain's documents— can alert the NEE extractor of changes in the writing style, i.e. external evidence in McDonald's (1996) terms.

Considering these arguments, both orthographic features and the zone feature, in addition to the lexical features, are considered the basic, lexically-oriented features for MOLI MENE. However, the set of orthographic features has been extended to provide MOLI MENE with more generic information on the form of the tokens in an attempt at increasing its portability. Table 4.1 summarises the orthographic features employed in MOLI MENE and which of them are also utilised in LexMENE.

The second modification in the new approach is related to the sizes of the context windows used by the basic features. Both baseline systems, namely siNymble and LexMENE, use a context window that is fixed in size. This restriction does not apply in MOLI MENE, and context windows of different sizes could be used in different applications to collect features for the maximum entropy model. Moreover, each type of feature that MOLI MENE utilises —basic or not— can define its own context window independently[1]. It is even possible for every type of feature to consider contextual information from windows of different lengths on the left and on the right of a focus token.

As a consequence of this flexibility of MOLI MENE, it is necessary to determine empirically a good set of values for these parameters for a given extraction task. This search could be done following traditional approaches for parameter selection, which generally involve running the system repeatedly on a training subset with different parameters and systematically evaluating their contribution on another subset of examples or using cross-validation, leave-one-out, etc. Section 4.2 presents an example of this process for the MUC-7 task.

4.2 Parameter setting

Although it is not possible to guarantee that a given set of parameters that work well for a particular set of features will work as well with other sets of features, it is unlikely that differences in performance will be substantial. In addition, the number of parameters that need to be set for each experiment is considerable and trying even some combinations can be very time consuming.

Therefore a set of initial experiments are conducted to determine a good set of parameters which will be kept fixed in all later experiments. This might prevent MOLI MENE from getting the best possible results, but it will be possible to outline the effect of different sets of features under the same parameters.

In each experiment, MOLI MENE trains its maximum entropy model with all features generated for a given set of parameters for all training examples in the MUC-7 training corpus. The evaluation of each model is conducted on all examples of the MUC-7 dryrun and formal test corpora. Results are reported separately for unseen, seen and ambiguous named entities, as well as the overall performance on all familiarity types.

[1] This flexibility applies to features that make use of a context window.

The experiments reported in this section aim to assess a number of options used by the two baseline NEE systems, in order to determine a set of parameters for MOLI MENE that yields good performance on the MUC-7 extraction task. More specifically, the experiments should answer:

1. which of the two different alternatives for using orthographic features provide more accurate predictions — i.e. considering all the features fired or just the one with higher precedence
2. which of the two different annotation formats allows the system to obtain better performance — i.e the BIO or the FMLU notation
3. how appropriate the sizes of the context windows used by the baseline systems are
4. how many iterations of the GIS algorithm utilised for building MOLI MENE's maximum entropy model are required

For the first experiment, MOLI MENE includes the same features as LexMENE, that is, for each focus token to be classified it considers the section of the document in which the token is found (document zone feature), the strings —or lexical features— of the tokens occurring within a fixed-length context window (hereafter the lexical window) and all —i.e. without precedence— orthographic features of table 4.1 fired by tokens within a fixed-length context window (hereafter the orthographic window). This configuration of MOLI MENE will be referred to as version 1 (V1).

The experiment starts by evaluating the sizes of the context windows used in the baseline systems. On the one hand, siNymble uses a lexical window of size [1,0], that is the focus token and one token on the left. On the other hand, LexMENE considers a lexical window of size [2,2], that is, a window of five tokens: the focus token and two tokens on either side.

Thus, it makes sense to evaluate MOLI MENE V1 with lexical window sizes [1,0], [1,1], [2,1] and [2,2], as a transition from siNymble's to LexMENE's parameters. Figure 4.1 presents the results obtained in these initial experiments.

The best performance is achieved with parameters [1,1], which obtain a significant improvement on seen named entities — though it shows a negative effect on ambiguous named entities— with respect to the size used by siNymble. This is an interesting result which indicates that although siNymble makes mistakes because of its lack of

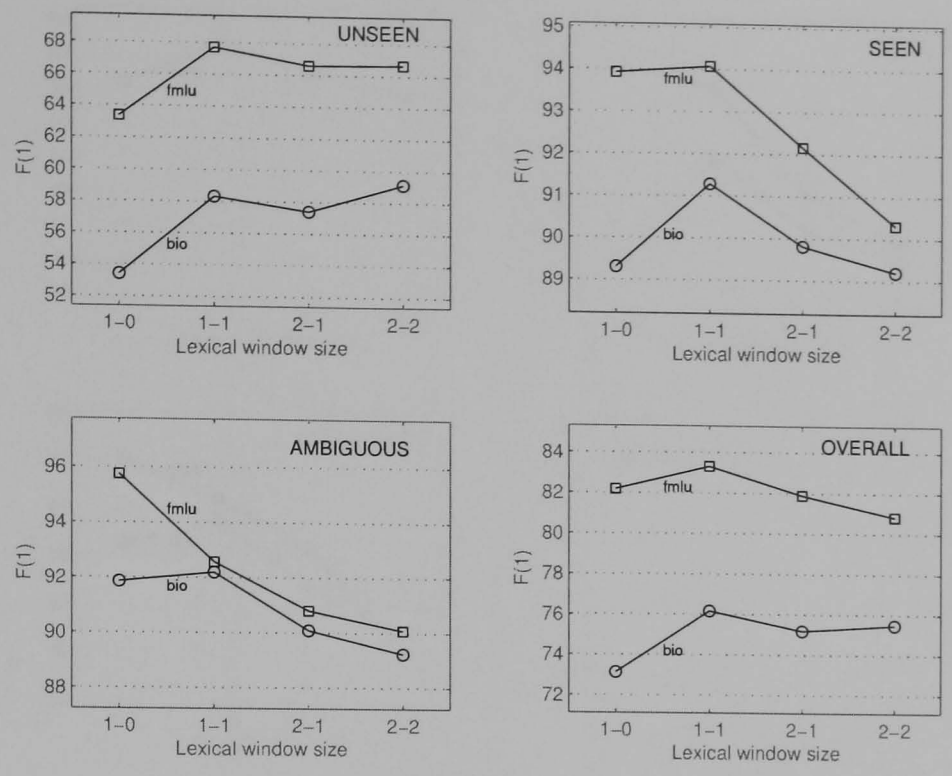


Figure 4.1: Experiments with MOLI MENE V1: size of the lexical window changing from [1,0] to [2,2] with data represented in BIO and FMLU notations. Other parameters are set as in LexMENE. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

context information (see section 3.4.2), adding lexical features from a larger context window might not provide the information needed for increasing its performance, unless significant changes are introduced into its hidden Markov model.

Named Entity recognition can be seen —and modelled— as a *chunking task* (Tjong Kim Sang 2002b, Tjong Kim Sang and De Meulder 2003). Chunking tasks can have many representations which are known to affect the performance of classifiers. In particular, siNymble utilises a representation called *BIO notation* (sometimes called BIO1 to differentiate it from other variations) in which words contained within a chunk are tagged 'I' and words outside any chunk are tagged 'O'. When there are two consecutive chunks of the same class, the first word of the second chunk is tagged 'B'. The following is an example text in BIO notation.

> Mr./O Jason/I-person Jones/I-person is/O currently/O in/O London/I-location England/B-location until/O June/I-date ,/I-date 25/I-date ./O

In contrast, LexMENE employs the *FMLU notation*, which is a more fine-grain representation in which a word is labelled 'F' if it is the first word of a multi-word chunk, 'M' if it is a word in the middle of a multi-word chunk, 'L' if it is the last word of a multi-word chunk and 'U' if it corresponds to a one-word chunk. The following is the example text in FMLU notation.

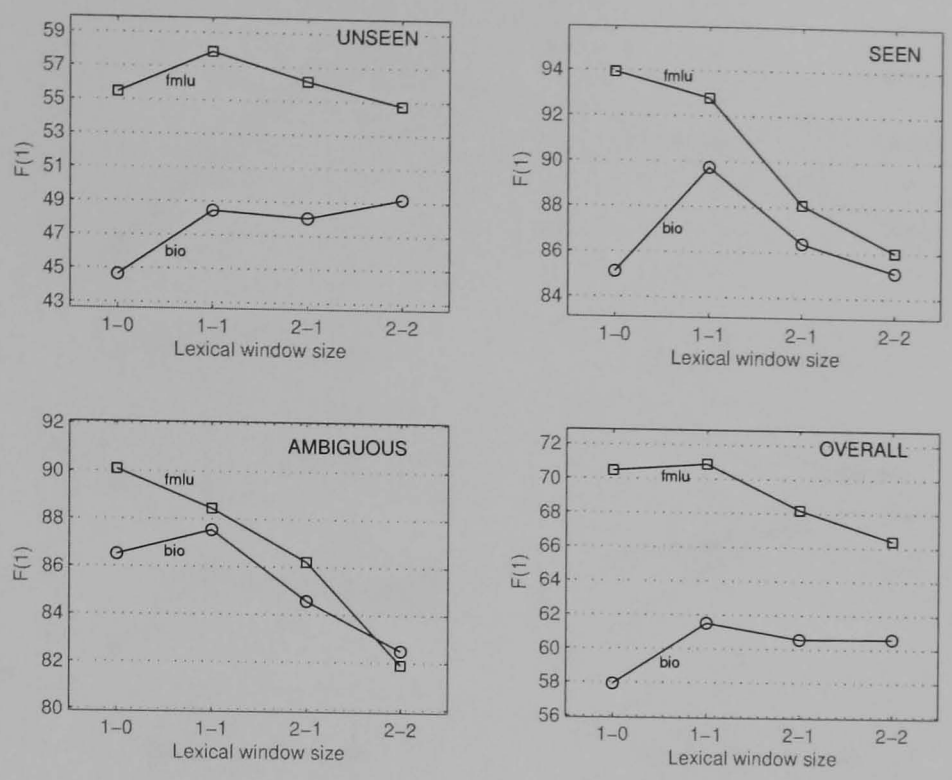


Figure 4.2: Experiments with MOLI MENE V1: size of the lexical window changing from [1,0] to [2,2] with data represented in BIO and FMLU notations. Other parameters are set as in LexMENE. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

Mr./O Jason/F-person Jones/L-person is/O currently/O in/O London/U-location England/U-location until/O June/F-date ,/M-date 25/L-date ./O

Figure 4.1 reveals that the FMLU notation consistently yields better results than the BIO notation.

Consequently, all future experiments will use FMLU notation and a lexical window of size one token to the left and one token to the right.

It would be interesting to evaluate the reliability of the decisions taken regarding the setting of MOLI MENE's parameters based only on experiments with the MUC-7 dryrun test corpus. This would be the normal situation in a real-world application. However, a small set of the target decoding documents, namely the MUC-7 formal test corpus, is also available in this case and this evaluation can be conducted here.

Figure 4.2 presents the same experiment but on the MUC-7 formal test corpus. Interestingly, these results closely follow the trend observed for the dryrun test set. This is specially significant considering that these corpora have been collected from different domains and present different amounts of seen and unseen named entities, as seen in section 3.1.

The next step then is to determine the best size for the orthographic window. This experiment is very similar to the one described above, and MOLI MENE is run with

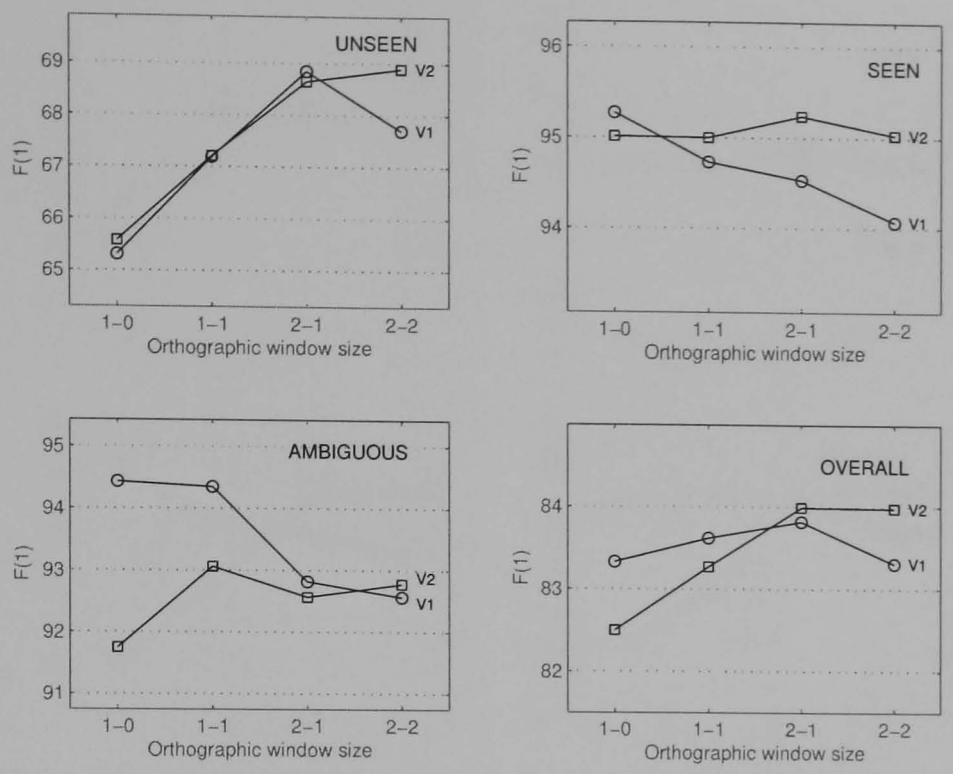


Figure 4.3: Experiments with MOLI MENE V1 and MOLI MENE V2: size of the orthograpic window changing from [1,0] to [2,2] with data represented in FMLU notation. Lexical window is set to [1,1] and other parameters are set as in LexMENE. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

the size of the orthographic window varying from [1,0] —as used by siNymble— and moving progressively towards a window of size [2,2] — as used by LexMENE.

In addition to the size of the orthographic window, there is another parameter considered in this experiment which differentiates the baseline systems: siNymble uses only the top orthographic feature —according to the precedence it defines for this kind of feature— of each token in the context window, whereas LexMENE utilises all orthographic features fired by these tokens.

Therefore, a second version of MOLI MENE has been prepared, namely MOLI MENE V2, which follows siNymble's strategy for this type of feature. Figure 4.3 shows the results obtained for this experiment.

It can be seen from this figure that LexMENE makes a bad decision setting the size of the orthographic window at [2,2], since the value [2,1] consistently gets better results with the configuration used by MOLI MENE V1. SiNymble may also be losing performance by fixing this window to size [1,0]. Increasing the orthographic window to sizes [2,1] or [2,2] seems to be better options, as these values help the classification of unseen named entities.

From these observations, henceforth experiments will employ the top orthographic feature —i.e. using the orthographic feature with higher precedence only— of tokens within a context window of size [2,2].

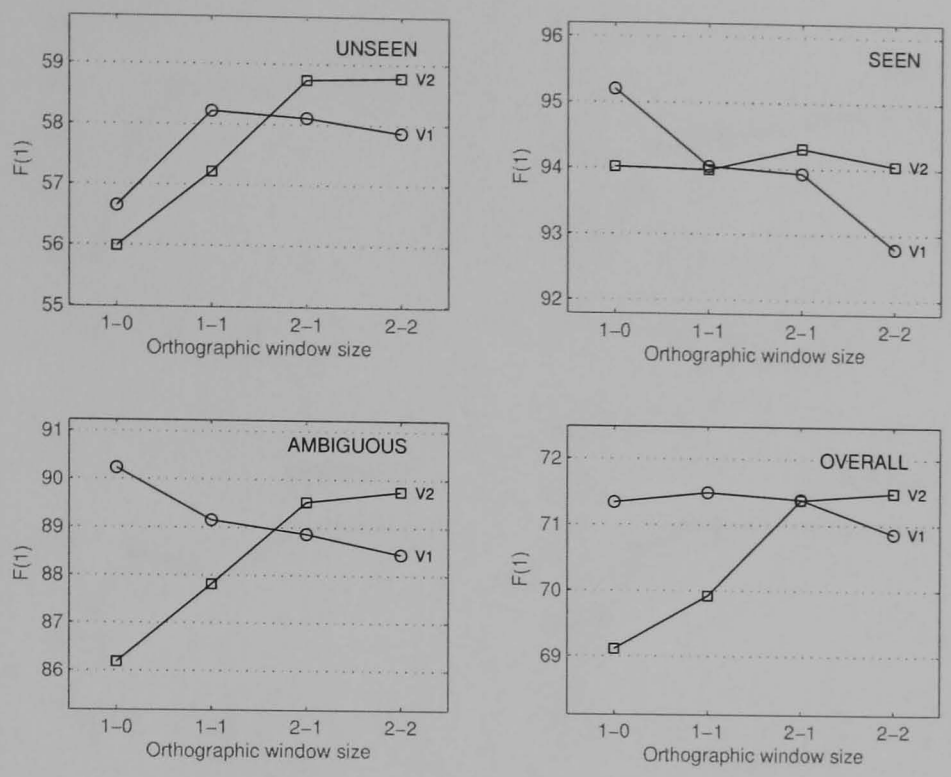


Figure 4.4: Experiments with MOLI MENE V1 and MOLI MENE V2: size of the orthographic window changing from [1,0] to [2,2] with data represented in FMLU notation. Lexical window is set to [1,1] and other parameters are set as in LexMENE. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

As previously, this experiment has been repeated with the MUC-7 formal test corpus. Figure 4.4 shows the results on this set of documents.

Once more the parameters fixed by looking only at the results obtained with the dryrun test corpus, are good parameters for the formal test corpus too. Moreover, both experiments present essentially the same trends.

The next experiment aims to determine good parameters for the GIS algorithm used to train the maximum entropy model used by MOLI MENE. In other NEE systems that employ maximum entropy models, the cutoff parameter is normally set to 3 or 4 with good results (Roth and van den Bosch 2002, Daelemans and Osborne 2003). Consequently, this parameter can be knowledgeably fixed to value 3, saving a potentially large number of experiments.

In this experiment, nine different models have been trained by allowing the GIS algorithm to run from 50 to 800 iterations. Figure 4.5 presents the performance of MOLI MENE V2 with each number of iterations tested.

Interesting observations can be obtained from these results: the more GIS iterations are permitted, the better the performance of the system on seen named entities; however, the performance of the system on unseen named entities stabilises at around 200 iterations. The same stabilisation is observed in term of the overall performance. This

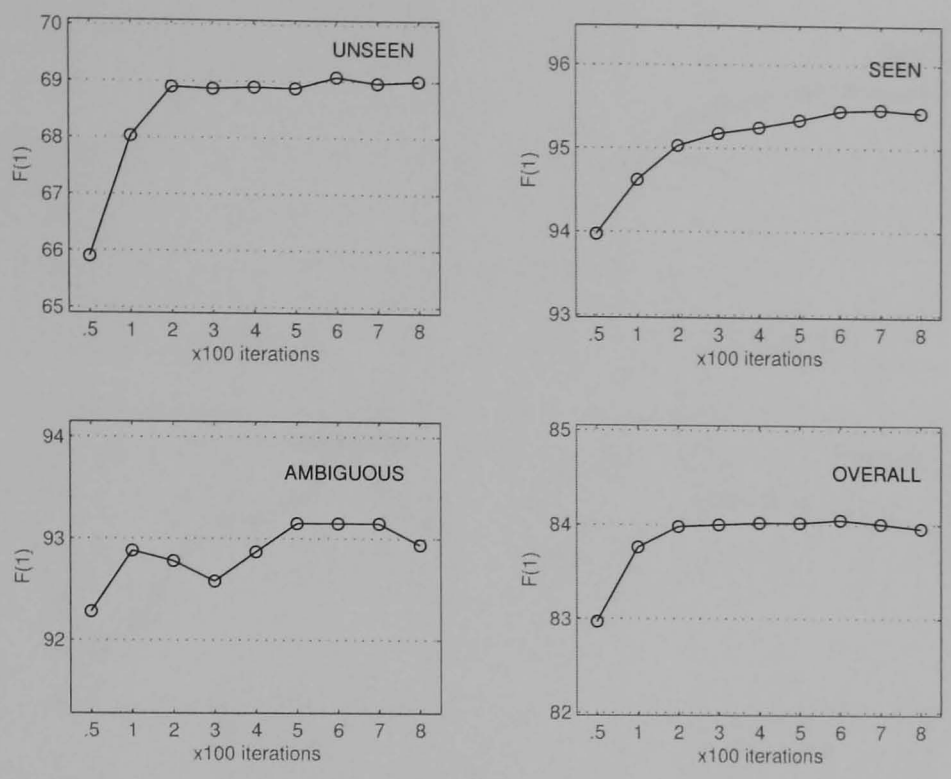


Figure 4.5: Experiments with MOLI MENE V2: the number of iterations for the GIS algorithm changing from 0.5x100 to 8x100. Data is represented in FMLU notation, context windows are set to [1,1] and [2,2] respectively and cutoff is set to 3. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

suggests that it is the recall on seen events that can be improved by allowing more iterations, but, at the same time, more false positives are being generated, which keeps the overall performance at the same level. A further analysis of the output given by scorer program confirmed this hypothesis.

From figure 4.5, it can be concluded that 200 is the best value for this parameter, as higher values do not contribute to this version's performance but significantly increase the time required for training the models. Therefore, 200 GIS iterations will be used for upcoming experiments.

As usual, these parameters —i.e. cutoff=3, iterations=200— are also tested on the MUC-7 formal test corpus. Figure 4.6 presents the results for such experiments. This figure indicates that 200 iterations was not a bad decision, though 300 would have been a better overall option. It clearly shows that there is room for improvement with lower numbers of iterations (under generalisation) and that higher values start to produce too many spurious named entities (over generalisation).

Interestingly, the trends on this corpus follow the trend on the dryrun corpus with the exception of ambiguous named entities, for which increasing the number of iterations produces a rise of the performance that did not happen in the experiment with the dryrun test corpus.

The flexibility introduced by MOLI MENE has allowed it to outperform LexMENE

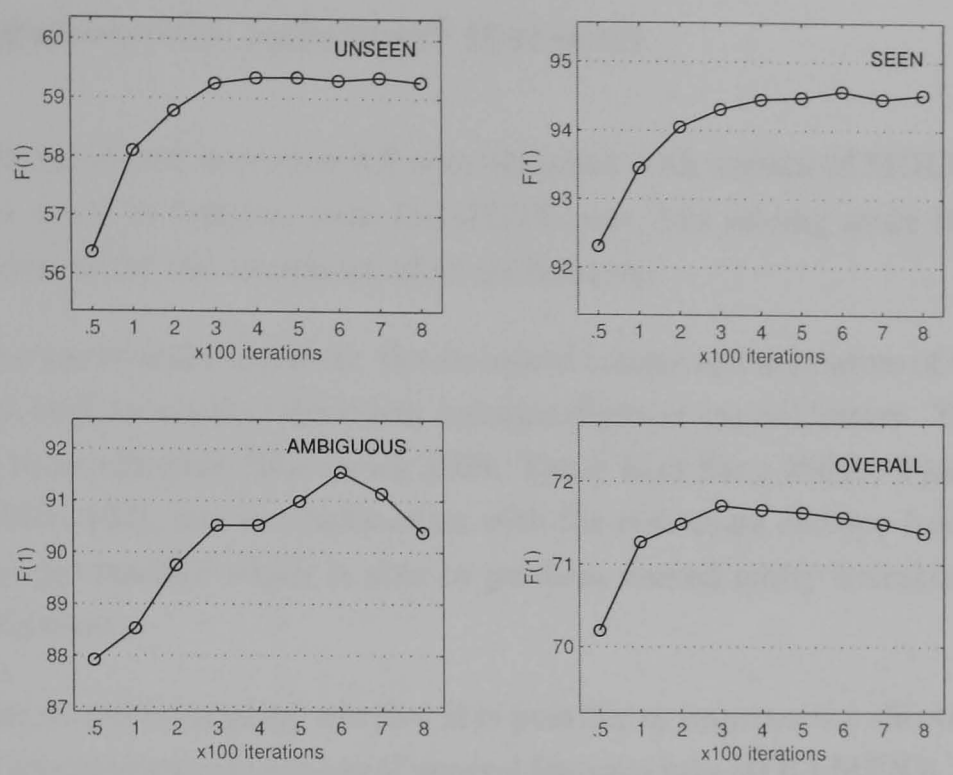


Figure 4.6: Experiments with MOLI MENE V2: the number of iterations for the GIS algorithm changing from 0.5x100 to 8x100. Data is represented in FMLU notation, context windows are set to [1,1] and [2,2] respectively and cutoff is set to 3. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

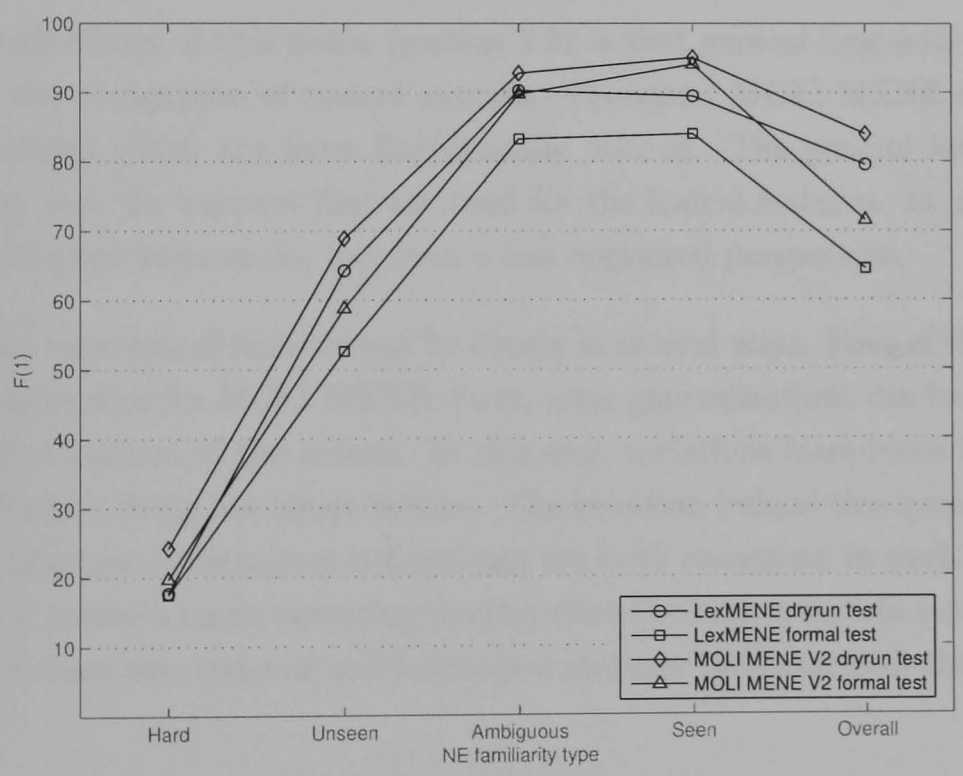


Figure 4.7: Comparison of LexMENE and MOLI MENE V2: Best (overall) F-score obtained by each NEE system. Corpora: MUC-7 training corpus and evaluation on both MUC-7 test corpora.

without adding new kinds of features. Figure 4.7 presents a comparison of the best overall performance obtained with these features. It can be seen that MOLI MENE obtains a 4.5% higher F-score on the dryrun test corpus than LexMENE and —more importantly— a 6.5% increase on the formal test corpus.

4.3 Linguistically informed features

All the results presented in section 4.2 were obtained with version of MOLI MENE that use the same types of features than LexMENE uses, but adding more flexibility and alternative options for the treatment of these features.

These features are lexically oriented: the strings of tokens and indicators of the orthography of tokens, such as whether the token contains digits or capital letters. These features have proved to be effective (Borthwick 1999, Tjong Kim Sang 2002b, Tjong Kim Sang and De Meulder 2003), and in combination with the maximum entropy framework, they provide a strong classifier which is able to perform named entity extraction with high levels of performance.

In this section, it will be studied whether it is possible to improve the effectiveness of the approach by introducing other types of general features into MOLI MENE. For example, it has been reported that by adding morphological features —such as 3-letter prefixes and 3-letter suffixes of tokens— to the lexical features, the performance of an extractor can be boosted (Tjong Kim Sang 2002b, Wu et al. 2002, Wu, Ngai and Carpuat 2003).

One of the hypotheses of this thesis (section 2.3) is that general linguistic information may help in the recognition of named entities. Therefore, MOLI MENE also includes a pool of features which are more linguistically related. The general idea is to add generalisation over the lexemes that are used for the lexical features. In a way, this is what morphological features do, but from a less organised perspective.

Generalisation over lexical features can be obtain in several ways. Four of these alternatives will be examined for MOLI MENE. First, some generalisations can be obtained by considering the lemmas of the tokens. In this way, variations introduced by inflection can be captured by firing one single feature. The intuition behind this type of feature is that if —for example— the tokens said and says are both recognised as weak (infrequent) indicators of a person's name occurring nearby, then the lemma say will concentrate the frequencies of these two lexemes and becomes a stronger indicator that the system can make use of.

The second generalisation form is to consider the part-of-speech (PoS) of tokens. This would allow MOLI MENE to identify certain kinds of words whose occurrence increases the chances of finding a named entity in the vicinity. For example and specifically for the MUC task, a preposition may help the system to identify locations in sentences like *they met at JFK Airport*, *he will be going to London next week*, etc., and dates from texts like *in August 2002*.

The third generalisation corresponds to the introduction of synonyms. It is quite intuitive for humans that the texts *Jack Yamili, chairman of FSFY* and *Yori Ugut, president of*

FSFM are closely related, as both are reporting something about a person who occupies an important position in an organisation. However, lexical features cannot capture this kind of relation and, in the best case, the tokens *chairman* and *president* will be used as indicators of person and organisation names separately. By introducing a feature of the form *syn(leader of an organisation)* which would be fired by these tokens —and other synonyms of these words— a more predictive indicator might be obtained.

As mentioned previously, one of the important advantages of maximum entropy models is their ability to combine information from different sources. Making use of this advantage, the fourth method for providing generalisation in MOLI MENE will be achieved by introducing syntactic information about the structure of the phrases in which tokens are found.

The intuition is that recognising syntactic patterns can yield good predictors of named entities. In the example above, both texts have the same syntactic pattern: *NP[Jack Yamili] PUNC[,] NP[chairman] P[of] NP[FSFY]* and *NP[Yori Ugut] PUNC[,] NP[president] P[of] NP[FSFM]*. Thus, noting that this syntactic pattern often contains a person and an organisation name might help MOLI MENE to recognise more named entities embedded in the text, even though they might be quite different in lexical terms.

4.4 Obtaining the new features

On the one hand, syntactic information can be obtained from a parser. There are literally hundreds of parsers or parsing techniques that can be considered. However, a detailed discussion of these alternatives is beyond the limits of this thesis. Nonetheless, it must be mentioned that in order to exploit the portability of MOLI MENE, such a parser should be based on machine learning methods. Furthermore, full parsing, which is computationally expensive and normally with limited coverage, is unnecessary and shallow parsers have become the common tool utilised for language engineering (Stevenson 1998).

Even narrowing the appropriate parsers in this way, the number of options is still very large. For example, several alternatives can be found in Cardie et al. (2000) and Daelemans and Zajac (2001).

To collect syntactic information for MOLI MENE, the MBSP parser has been used (Daelemans, Veenstra and Buchholz 1999). This decision was based on practical reasons, rather than theoretical arguments. MBSP stands for Memory-Based Shallow Parser, as it employs three memory-based learning modules applied in cascade. These modules carry out part-of-speech tagging, text chunking and identification of basic syntactic

relations respectively. Each module is trained on the the Wall Street Journal section of the Penn Treebank (Marcus, Santorini and Marcinkiewicz 1993).

Daelemans, Veenstra and Buchholz (1999) report remarkably good performances for MBSP: 94.6% accuracy for the PoS tagger, around F-score 94% for chunking module and about F-score 78% in subject/object detections. However, the main reason for selecting this shallow parser is that it is freely available: an on-line demo of MBSP has been applied to each document in the MUC-7 corpora and the syntactic information has been appropriately annotated. This selection was validated later as it was this shallow parser that was used to provide the corpora of the CoNLL-2002 and CoNLL-2003 shared tasks with syntactic information.

On the other hand, obtaining lemmas and synonymic relations of words has normally been addressed by building morphological analysers and ontologies. However, ontologies are commonly oriented to capturing the most important concepts involved in a specific extraction task (Gaizauskas and Humphreys 1997). This is so because the knowledge required to construct a general-purpose ontology, which at the same time can be helpful for a determined extraction process, is unmeasurably large.

Fortunately, the general-purpose lexical database WordNet®, developed at the Cognitive Science Laboratory of Princeton University over several decades now, can be used for this purpose.

WordNet has been described as a "lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory" (CSL 2004). There are important features of WordNet which MOLI MENE can make use of:

1. A morphological analyser is included for looking up inflected words
2. Nouns, verbs, adjectives and adverbs are organised into synonym sets —called *synsets*— each representing one underlying lexical concept
3. Several relations are incorporated to link synonym sets, among which hyponyms, hypernyms and coordinate terms might be particularly useful

Thus, WordNet provides all the necessary information for implementing the generalisations over the meaning of tokens described in section 4.3. Given the number of years invested in the development of Wordnet, there exists a number of different versions of the database. In all experiments with MOLI MENE, version 1.7.1 is used. This version contains 146,350 nouns, verbs, adjectives and adverbs, which are organised into 111,223 synsets. The most important syntactic category are nouns, which account for 109,195 words divided into 75,804 synsets (CSL 2004).

4.5 Organising the new features

The more linguistically informed features will be organised in three different versions of MOLI MENE for the experiments. These new versions follow the discussion in section 4.3: version 3 (V3) will consider the syntactic structure around the focus token; version 4 (V4) will consider features that provide generalisations over inflection, namely lemmalisation and PoS tags, and unstructured semantic information; finally, version 5 (V5) will consider semantic information but in a more organised approach. Note that these versions of MOLI MENE do not accumulate feature sets, that is the only features in common are the lexically-oriented ones inherited from LexMENE — i.e. lexical, orthographic and zone features.

4.5.1 MOLI MENE V3: syntactic patterns

MOLI MENE V3 adds syntactic features to the basic lexically-oriented features. The pool of new features corresponds to the tag of the chunks and the head word within the chunks of the basic constituents occurring in a fixed-size context window around the chunk that contains the focus token. This window will be referred to as the *chunk window*. For example, consider the following sentence.

NP[He] VP[will succeed] NP[Amilie Jackson] PUNC[,] NP[chairwoman] P[of] NP[FSFY] PUNC[,] NP[next October] PUNC[.]

Supposing that the size of the chunk window is fixed to two, the features fired by the token Amilie would be

tag_{-2}=NP, $head_{-2}$=he, tag_{-1}=VP, $head_{-1}$=succeed, tag_0=NP, $head_0$=jackson, tag_{+1}=PUNC, $head_{+1}$=',', tag_{+2}=NP, $head_{+2}$=chairwoman

As explained in section 4.3, these new features might provide valuable information about patterns that are more likely to contain named entities.

The number of chunk features could be a free parameter of the system, but it has been fixed for the initial experiments to the information gathered from the four chunks on each side of the focus chunk. This number follows the intuition that most useful modifiers for named entity extraction, such as appositive noun phrases, can be contained in a chunk window of size [4,4].

4.5.2 MOLI MENE V4: lemmas, PoS tags and synonyms

MOLI MENE V4 includes the generalisations over the tokens found in the lexical window in the pool of features. It could be possible to use a different context window, but this would increase the number of parameters for MOLI MENE that need to be set and would not be a direct generalisation of the lexical features.

The new types of features include the lemma of a token, its part-of-speech tag and the token's related synsets. There are several options for what can be included in the set of synsets fired by a token. Here, the most general alternative has been adopted and this set will consist of the synsets for the token, the synsets of its direct hyponyms, the synsets of its direct hypernyms and the synsets of its coordinate terms.

WordNet's hierarchical organisation of words is straightforward. A synset is defined as a synonym set, that is, a set of words that are interchangeable in some contexts. A hyponym of a noun or a verb w corresponds to a member of the a class defined by w: w' is a hyponym of w if w' is a (kind of) w. Similarly, a hypernym of a noun or a verb w' describes a whole class of which w' is a specific instance: w is a hypernym of w' if w' is a (kind of) w. Finally, coordinate terms are nouns or verbs that share the same hypernym (CSL 2004, adapted from the Glossary of Terms).

It must be noticed from the definitions above that adjectives and adverbs can only be generalised over synonyms. There are many details omitted here. For instance, the hypernym relations between nouns are obtained by applying different criteria to those for obtaining the hypernym relations between verbs. The reader is referred to Fellbaum (1998) for an exhaustive discussion of the design and contents of WordNet and papers reporting some pieces of research that utilise WordNet.

For example, consider the focus token *chairman* in the sentence *Jack Yamili, chairman of FSFY*. WordNet finds the following synsets for this token[2]:

> noun-08577148: president, chairman, chairwoman, chair, chairperson – (the officer who presides at the meetings of an organization; "address your remarks to the chairperson")
>
> verb-01918094: chair, chairman – (act or preside as chair, as of an academic department in a university; "She chaired the department for many years")

This token appears playing a noun's function: it is found inside a basic NP and is labelled with the part-of-speech for common nouns. Therefore, MOLI MENE V4 ignores the synset verb-01918094 and only consider the synset noun-08577148 for further processing. This works analogously for the case in which the token is functioning as a verb.

[2]An attempt to maintain WordNet's answers literally will be made for these kinds of examples, though some grammatical forms and words might be found peculiar. Some alterations have been made in order to make the examples clearer.

This is the only effort that MOLI MENE —in all versions that use semantic information— makes to filter out synsets provided by WordNet. No explicit disambiguation of the meanings of tokens is performed, as the maximum entropy model should be able to capture the most relevant senses based on the frequency with which these they occur in the contexts of named entities.

After the cleaned set of synsets is obtained, WordNet is asked for the hyponyms, hypernyms and coordinate terms of each synset contained in this set. The following is the information provide by WordNet for the sysnset noun-08577148:

Hyponyms
noun-08800476: vice chairman

Hypernyms
noun-08577370: presiding officer

Coordinate terms
noun-08467159: moderator; noun-08577148: president, chairman, chairwoman, chair, chairperson; noun-08706865: Speaker

Thus, after trimming away repeated synsets —and repeated lemmas— MOLI MENE would be supplied with the following features fired by the focus token chairman[3]:

$lemma_0$=chairman, PoS_0=NN, $synset_0$=noun-08467159, $synset_0$=noun-08577148, $synset_0$=08577370, $synset_0$=noun-08706865, $synset_0$=noun-08800476, $lemma_{-1}$=',', PoS_{+1}=',', $lemma_{+1}$=of, PoS_{+1}=IN

The features fired by this example token can help MOLI MENE V4 in the extraction process. Indeed, if the token *chairman* is found to be a predictor of the occurrence of a named entity nearby, the system will also be inclined to use this predictor when the tokens *president*, *chairwoman*, *chair*, *chairperson*, *vice chairman*, *moderator* or *speaker* are found.

4.5.3 MOLI MENE V5: trigger synsets

MOLI MENE V5 is inspired in a generalisation over the concept of trigger words. Trigger words is a general term to designate words which co-occur with high frequency with linguistic events in text (Rosenfeld 1996). They have been used extensively by NEE systems (Gaizauskas et al. 1995, Carreras et al. 2002, Zhang, Shen, Zhou and Tan 2004, for example). For the MUC task, trigger words commonly contain personal

[3]This assumes a lexical window of size [1,1]. The other tokens in the window, namely ',' and *of*, do not fire synsets because they are not included in WordNet's database. In these cases, the lemma is assumed to be the lexical form.

titles (e.g. *Mr.*, *PhD.*, *MD.*), organisation designators (e.g. *Ltd.*, *Corp.*, *PLC.*), organisation keywords (e.g. *Bank*, *Services*, *Association*), location keywords (e.g. *mountain*, *lake*, *river*, *city*), job titles (e.g. *chairman*, *president*, *executive officer*, *general*, *MP*) and time keywords (e.g. *earlier*, *ago*).

By providing the appropriate sets of trigger words, NEE systems can recognise —or at least be more alert to— the presence of a named entity in the text. However, these lists of words are highly domain dependent and normally handcrafted from the training texts. Usually, an analysis of both the initial lists and the domain is conducted to add other potentially helpful trigger words which do not occur in the training data.

MOLI MENE V5 attempts to overcome this portability limitation by replacing trigger words by trigger WordNet synsets. The intuition behind this approach can be best explained with an example. Consider the following (partial) sentences:

Senator Kelly criticised...
Reverend Smith denied...
Coach Johnson promised...

It is clear that these sentences make use of a common pattern to report an action executed by a person, which is identified by his/her name. Traditionally, the recognition of this type of pattern would be approached by adding these words into a list of trigger words for named entities of class person.

Because synsets are organised into hierarchies in WordNet, each synset is part of at least one hierarchy which is headed by a root synset called a unique beginner. The paths from each synset to unique beginners form a structure known as an *hypernym tree*. For the candidates to trigger words in the running example, WordNet returns the following hypernym trees[4]:

```
noun-08663173: senator
   => noun-08409434: legislator
      => noun-08405572: lawgiver, lawmaker
         => noun-07904081: leader
            => noun-00005303: person, individual, someone, ...
               => noun-00003135: organism, being
                  => noun-00002956: living thing, animate thing
                     => noun-00013067: object, physical object
                        => noun-00001742: entity, physical thing
               => noun-00004911: causal agent, cause, causal agency
                  => noun-00001742: entity, physical thing

noun-08151518: clergyman, reverend, man of the cloth
   => noun-07801084: spiritual leader
```

[4]Only one sense (synset) per word is presented for this example.

```
        => noun-07904081: leader
           => noun-00005303: person, individual, someone, ...
              => noun-00003135: organism, being
                 => noun-00002956: living thing, animate thing
                    => noun-00013067: object, physical object
                       => noun-00001742: entity, physical thing
              => noun-00004911: causal agent, cause, causal agency
                 => noun-00001742: entity, physical thing
noun-08154623: coach, manager, handler
   => noun-08778227: trainer
      => noun-07904081: leader
         => noun-00005303: person, individual, someone, ...
            => noun-00003135: organism, being
               => noun-00002956: living thing, animate thing
                  => noun-00013067: object, physical object
                     => noun-00001742: entity, physical thing
            => noun-00004911: causal agent, cause, causal agency
               => noun-00001742: entity, physical thing
```

Thus, although the synsets for the tokens *senator*, *reverend* and *coach* might appear just once in the training corpus, the synsets for the concepts *leader*, *person*, *agent*, etc. are *hit* three times —not considering other senses for the tokens under consideration, the other words in these example sentences nor other training sentences. These concepts might be considered *trigger synsets* for the named entity class person.

Of course, not all synsets can be considered trigger synsets. They could be a source of noise if they show affinity for more than one class of named entity or for words that are not targeted by the extraction task. Moreover, concepts close to the top of hypernym trees might be hit too many times to provide useful information. Synsets at the leaves of the trees might be hit too seldom to be considered as reliable predictors, and their frequency might not be much different from simply using the lexemes themselves.

This can immediately be seen as a machine learning task: high-level synsets are too general hypotheses of what a relevant piece of text is; and low-level synsets are hypotheses that are too specific to provide any useful learning. Therefore, it will be mainly the job of the maximum entropy framework to recognise the most relevant trigger synsets to be considered by the model. Nonetheless, some preprocessing over the synsets will be applied in order to help the GIS algorithm in selecting those with more predictive power.

The process starts by collecting all synsets hit by the tokens in the lexical window. According to the settings fixed in section 4.2, this generates three lists of initial trigger synsets: the first list contains synsets that might predict that the next token on the right belongs to a given class; the second list gathers the synsets fired by focus tokens which might predict that the current token is of a given class; and the third list stores

synsets which might predict that the previous word appertains to a given class. These lists are referred to as the *next-is list*, the *this-is list* and the *previous-is list* respectively.

At the same time that these lists of synsets are being generated, counts of the frequency they are seen with each named entity class —and the not-a-name class— are also compiled.

The first filtration of the synsets lists is applied at this stage, and corresponds to discarding synsets that are hit less than three times, as they will be filtered out anyway when the frequency cutoff is enforced.

But the counts collected can also be used to determine the predictive power of each synset in the initial lists. For example, consider the following counts gathered for two potential trigger synsets in the MUC extraction task:

Synset	not-a-name	date	location	money	organisation	percent	person	time
noun-12726340	5	0	1	0	5	0	0	0
noun-11422319	37	1	0	0	1	0	0	0

Thus, the synset *noun-12726340* was seen 45% of the time predicting the class not-a-name and another 45% of the time predicting the class organisation. In contrast, the synset *noun-11422319* was seen 95% of the time predicting the class not-a-name. Clearly, the latter gives much more information than the former.

What is needed now is a way of capturing this intuitive idea of more informed or more predictive synset. One possibility is to use the distance between the probability distribution that a set of counts defined and the least informative distribution, that is the uniform distribution. This is normally done using the Kullback-Leibler (KL) distance[5] —also called KL divergence, KL number and relative entropy (Kullback and Leibler 1951, Cover and Thomas 1991). This number is usually denoted D and its definition —for discrete distributions— is given in equation 4.1. KL distance measures the difference between two probabilistic distributions p and q. Thus, if q is a true distribution, the relative entropy can be used to measure how good an approximation of q the distribution p is. Similarly, if q is the uniform distribution, the KL distance can be used to determine how concentrated the probability mass is among the possible outcomes of the distribution. The greater the relative entropy, the more concentrated the mass distribution is — i.e. the more distant from the uniform distribution in which the probability mass is equally divided among the possible outcomes.

$$D(p||q) \equiv p(x) \log \frac{p(x)}{q(x)} \tag{4.1}$$

[5]Although this number is normally called a distance, it does not satisfy the triangle inequality and is therefore not a true metric. Nonetheless, it satisfies important mathematical properties, such as that it is is always nonnegative and equals to zero if and only if the distributions being compared are equal.

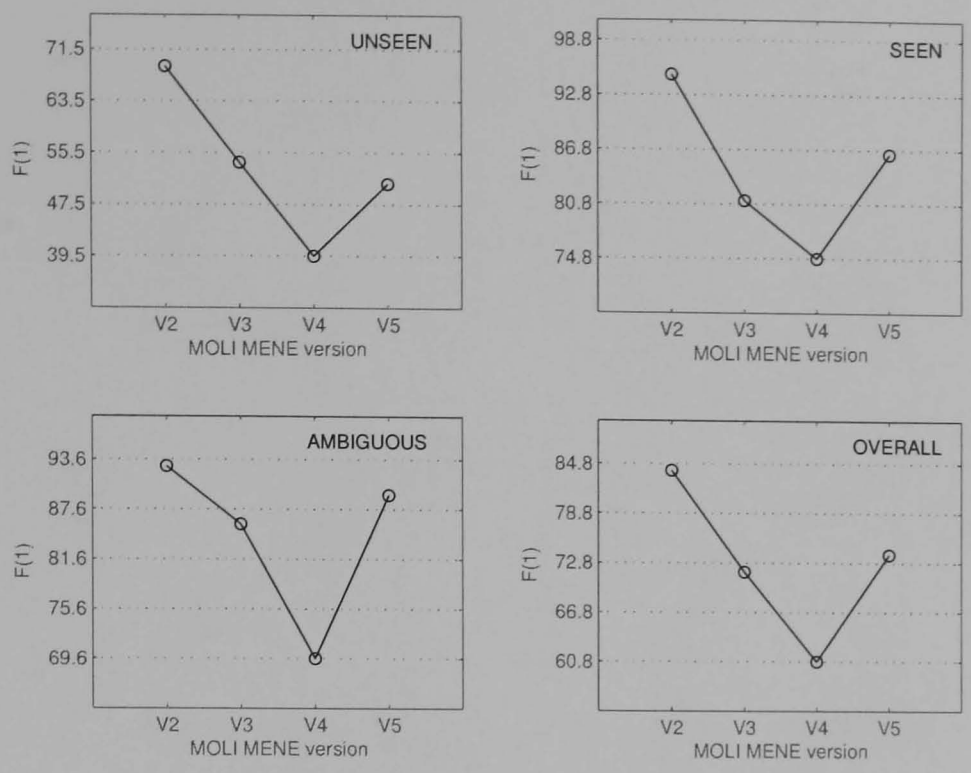


Figure 4.8: Comparison of the performances of version V2, V3, V4 and V5 of MOLI MENE: data is represented in FMLU notation, context windows are set to [1,1], [2,2] and [4,4] respectively, cutoff is set to 3 and GIS iterations to 200. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

Therefore, the second pre-processing step corresponds to filtering out synsets whose counts define a probability distribution that has a relative entropy —with the corresponding uniform distribution— which is less than 2.0. This bound coincides with distributions that divide all their probability mass between at most two named entity classes. Thus, the worst kind of trigger synsets that are accepted are those which predict "the token is either of class y_1 or of class y_2".

Note that this value is for the MUC extraction task, which defines seven named entity classes plus a not-a-name default class. However, this parameter can be easily calculated automatically from the number of named entities to be identified and, therefore, it does not reduce the portability of the approach.

4.6 A comparative experiment

Figure 4.8 presents a comparison of the performance obtained by versions V2, V3, V4 and V5 of MOLI MENE described in section 4.5. For this experiment, parameters set for MOLI MENE V2 are used; thus the lexical window size is set to [1,1], the orthographic window size is set to [2,2] and precedence is applied, the unknown words threshold has value three, the frequency cutoff is fixed to three and the GIS algorithm runs 200 iterations. MOLI MENE V3 is using a chunk window of size [4,4] as explained in section 4.5.1.

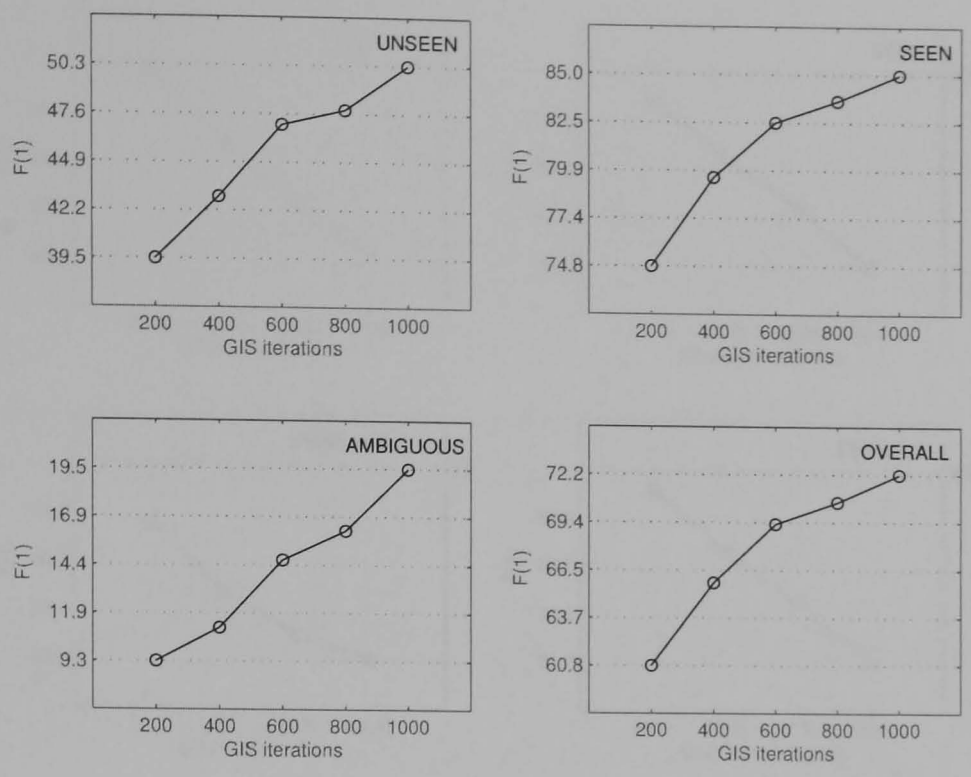


Figure 4.9: Experiments with MOLI MENE V4: the number of iterations for the GIS algorithm changing from 200 to 1000. Data is represented in FMLU notation, context windows are set to [1,1] and [2,2] respectively and cutoff is set to 3. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

The main observation from figure 4.8 is that the most simple version, namely V2, obtains the best results. However, this does not necessarily imply that the more linguistically informed features are irrelevant. If that were the case, the maximum entropy model would have set its weights to values that would simply ignore these features (Rosenfeld 1996, Mikheev 1998).

The negative effect of the newly added features on MOLI MENE's performance might have more to do with the convergence of the parameters of the maximum entropy model. Firstly, the size of the model has been dramatically increased and there are many more weights that need to be calibrated by the GIS algorithm. Secondly, it can almost be said for certain the new features are overlapping. For instance, all tokens considered in the example of section 4.5.3 for V5 hit the concept *leader*, which invariably meant that they also hit the concepts *person*, *organism*, *living thing*, etc. These hits are not independent but due to the hierarchical organisation of WordNet. As a result, more iterations of the GIS algorithm are needed (Ristad 1998, Borthwick 1999).

To check this hypothesis, a further experiment with MOLI MENE V4 was conducted, in which the iterative scaling algorithm is allowed to run for 400, 600, 800 and 1,000 iterations. Figure 4.9 shows the results for this experiment, in which the system obtains significant increases in performance in all familiarity types when more iterations are allowed. In this figure —although there are signs of stabilisation— the curve seems far from convergence, which suggests that V4 could get benefits from allowing the iterative scaling algorithm to run many more iterations.

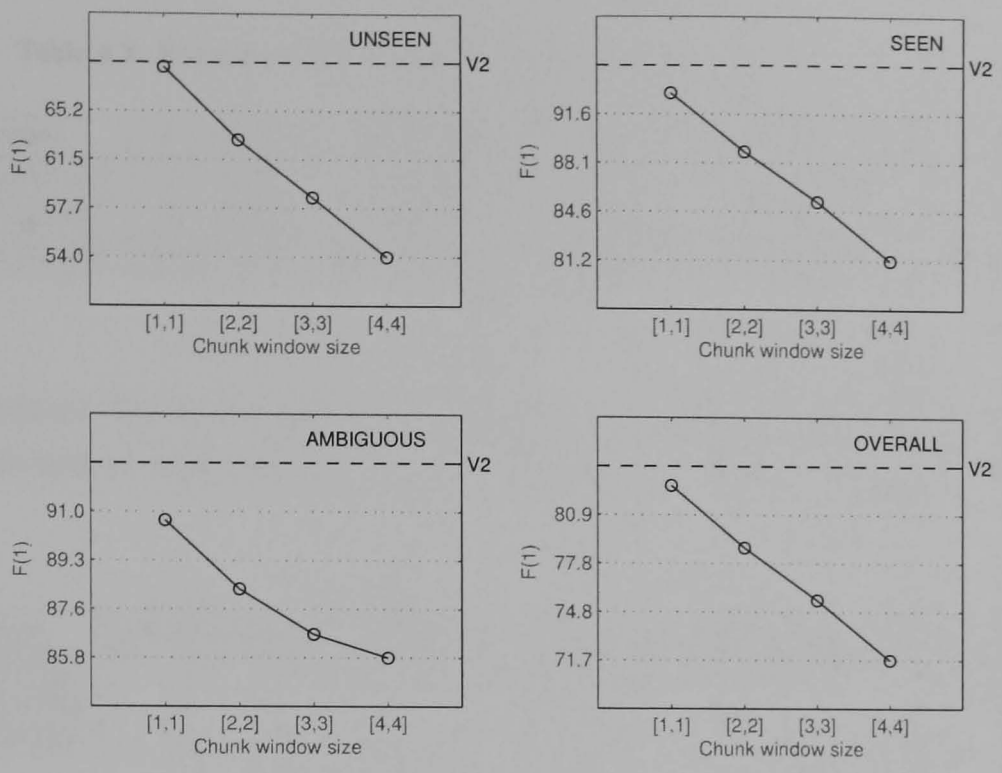


Figure 4.10: Experiments with MOLI MENE V3: the size of the chunk window changing from [1,1] to [4,4]. Data is represented in FMLU notation, the other context windows are set to [1,1] and [2,2] respectively, cutoff is set to 3 and GIS iterations to 200. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

However, the improvement described above is very expensive: building the model with 1,000 iterations took two days and required a machine with several gigabytes of memory. This is an unacceptable increase of computational resources considering that MOLI MENE V2 obtains better performance with a model which only takes 15 minutes to be built on a normal desk computer with 1GB of memory.

4.7 Reducing feature pools

One possible solution to this problem is to reduce the number of linguistically informed features that arrive at the iterative scaling phase. Therefore, the following experiments aim to determine whether by directly reducing the number of features, MOLI MENE can obtain a better maximum entropy model.

In the first experiment, the chunk window used by MOLI MENE V3 is reduced to sizes [3,3], [2,2] and [1,1]. Thus, instead of generating 18 chunk features per token, the GIS algorithm will have to deal with 14, 10 and 6 features per token respectively, in addition to the basic features used by MOLI MENE V2. Results for these experiments are shown in figure 4.10.

The second experiment aims to determine whether better predictors can be obtained by reducing the number of trigger synsets that MOLI MENE V5 considers. To do this, it is necessary to rank the trigger synsets so that the best ones can be selected. However,

Table 4.2: Example of different, possible measures for ranking trigger synsets.

Synset	f	KLD	$f * KLD$	$Split * KLD$	$log_2 f * KLD$	$F(1)(log_2 f, KLD)$
noun-10371413	3	3.000	9.000	0.000	4.755	2.074
noun-11549024	9	3.000	27.000	0.000	9.510	3.083
noun-11513257	58	2.874	166.712	0.361	16.838	3.856

the KL distance from the uniform distribution —used for filtering out less relevant synsets— is not an appropriate index for doing this ranking. Consider the following counts:

Synset	not-a-name	date	location	money	organisation	percent	person	time
noun-10371413	3	0	0	0	0	0	0	0
noun-11549024	0	0	0	0	0	0	9	0
noun-11513257	57	0	1	0	0	0	0	0

The KL divergences for these synsets are 3.0, 3.0 and 2.874 respectively. However, if decoding data presents approximately the same frequencies for trigger synsets as the training data —which is also the assumption on which the application of a statistical machine learning approach is based— then it makes sense to consider that the second synset provides more information than the first one, and that the third synset is more predictive than the other two. This is so because the first synset was only fired by the minimum number of examples to be considered a trigger synset, whereas the second and third synsets were fired three times and 20 times more often respectively.

Thus, the frequency with which a trigger synset is fired must be considered in the index for ranking them. There are several alternatives for doing this, such as multiplying the KL distance by the absolute frequency of the synset or using the Split Information measure (Quinlan 1986). Table 4.2 summarises some of these alternatives for the sample synsets above.

MOLI MENE uses a direct approach which was observed to produce the desired effect on a small set of examples tested: the weighted harmonic mean —exactly like the F-score used for measuring performance (van Rijsbergen 1979)— of the logarithm of the synset's frequency and its KL divergence with respect to the uniform distribution (last column of table 4.2).

Using this score, trigger synsets are ranked according to their predictive power. Then, only the most important synsets can be selected to be considered by MOLI MENE.

With this strategy at hand, a new experiment with V5 was conducted in which the size all three lists of trigger synsets was fixed to a value that varies from 1,000 to all synsets

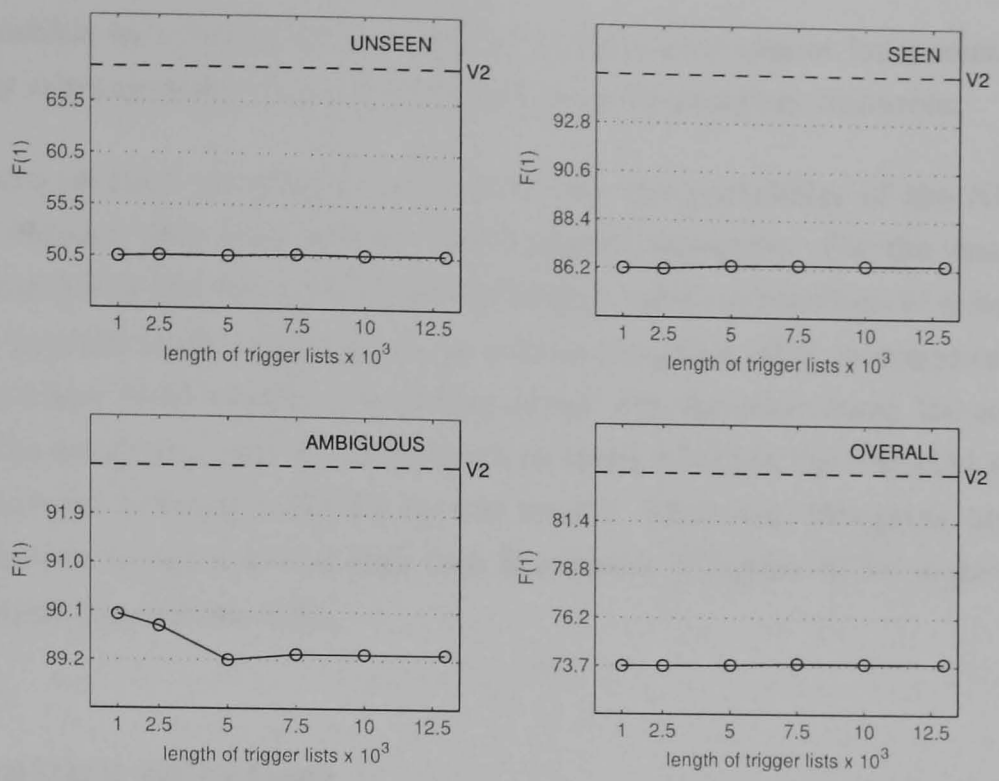


Figure 4.11: Experiments with MOLI MENE V5: the size of the three lists of trigger synsets changing from 1,000 to around 13,000. Data is represented in FMLU notation, context windows are set to [1,1] and [2,2] respectively and cutoff is set to 3. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

considered relevant (around 13,000 elements). The results of this experiment are shown in figure 4.11.

Unfortunately, neither of these experiments exhibit an improvement over MOLI MENE V2. On the one hand, the performance of V5 is fairly constant and much lower than the one shown by V2, no matter what the size of the lists of trigger synsets is. On the other hand, the performance of V3 seems to be affected by the size of the chunk window, but the best results of figure 4.10 —obtained with the minimum size [1,1]— is not enough to outperform V2 and this performance decreases as the chunk window grows in size.

In conclusion, directly reducing the number of features to be considered by the maximum entropy model of MOLI MENE does not solve the convergence problem, and a more sophisticated approach is needed. Such an approach might consider a much more careful selection of features, as well as the creation of complex features in which more than one of the atomic features are combined. For example, firing a feature "previous word is a trigger synset for the class person and the current token is Smith" might be more informative than just firing the two features individually, reducing the number of iterations required to introduce this information into the model.

Traditionally, it is the modeller's work to design and select features that will provide the maximum entropy model the necessary information to perform the target task well. This is normally done by trying a set of features, investigating where the system is making mistakes and, based on this information, proposing new features that can correct these weaknesses. This process is repeated several times until a set of features

seems impossible to improve on —or whose performance cannot be increased without significantly compromising other issues, such as computational resources.

However, this method introduces limitations into the portability of the NEE system. This is so because this task requires two types of expertise. On the one hand, the modeller must know the domain in order to identify valid conjunctions of atomic features that might provide useful information, as well as recognise what information is causing mistakes and how these problems could be solved. On the other hand, the modeller has to master the maximum entropy framework to check whether the intended effect of the designed features is being reflected by the model. Moreover, designing and selecting complex features is not a trivial task (see Borthwick (Chapter 6) for a good example, briefly discussed in section 4.8).

4.8 Feature selection

From the above results, it has become necessary to utilise some method for selecting features from the large pools of features managed by MOLI MENE V3, V4 and V5, in order to facilitate the task to the underlying maximum entropy framework.

Selecting relevant features before applying an inductive method is a common practice in machine learning (Blum and Langley 1997), and there have been few attempts to perform this selection automatically for maximum entropy models (Berger et al. 1996, Mikheev 1998). These methods aim to select the most relevant features from an initial pool of features, as large as possible. However, the nature of the task studied here makes these approaches impractical. Consider MOLI MENE V3 for example. When the chunk window is set to [4,4], 18 atomic features are introduced into the framework in addition to the atomic features used by MOLI MENE V2: the chunk tag and the head word of the focus chunk, the four chunk tags and the four head words on the left of the focus chunk, and the four chunk tags and the four head words on the right of the focus chunk. Considering only these newly introduced atomic features, there are 262,143 possible combinations. The tag features have around 13 different values and the word features have around 2,900 distinct lexemes, which raises the theoretical number of valued complex features to a staggering 211,322,798,375,569,000,000,000,000,000,000,000,000,000. Although only a fraction of this theoretical number of combinations will be seen during training, the amount is still prohibitively large for trying all possible complex features.

These difficulties were already discussed in section 2.4.4, and one of the possibilities suggested to overcome the problem was the use of decision trees to obtain complex features from a pool of atomic features only, as proposed by Park and Zhang (2002). However, as also stated in section 2.4.4, there have been reports that even powerful,

commercial decision tree induction algorithms are not able to cope with this sheer number of attributes (Ratnaparkhi 1998, Borthwick 1999).

Considering these facts, MOLI MENE utilises an efficient rule learning algorithm: Ripper (Cohen 1995). Ripper is specially suitable for the purpose discussed above for several reasons:

- ▷ like many inductive algorithms, Ripper creates a lists of rules that try to use only the most relevant valued attributes —i.e. features— from a potentially large set of attributes, so that a general hypothesis to explain the observation can be formed
- ▷ although the induction is greedy —based on Fürnkranz and Widmer's (1994) IREP— its performance is extremely competitive with the more developed, expensive C4.5 algorithm
- ▷ due to this greediness, the algorithm is extremely efficient as it scales nearly linearly with the number of training examples — actually $n \log^2 n$
- ▷ the latest versions have been provided with an extended feature-vector representation which can manage, in addition to continuous and nominal features, *set-valued* features (Cohen 1996); this characteristic is quite useful, especially for NLP tasks in which it is not uncommon to find token features —as in the features used by MOLI MENE— which may contain an arbitrary number of lexemes

The approach is quite simple and closely follows the idea of Park and Zhang (2002). First, all atomic features are provided to Ripper, which induces a list of complex rules — that is, conjunctions of valued, atomic features. Because one of the objectives of this chapter is to determine the impact of more linguistically-informed features, the default features used by MOLI MENE V2 are left out of this procedure. Then, each of the resulting rules is considered a complex feature for the maximum entropy model, which can be fired by any training example disregarding the class it predicts. In the next step, these complex features are joined with the default features. Finally, a maximum entropy model for MOLI MENE is obtained from this pool of atomic and complex features, following an application of the cutoff selection.

In the decision list approach, rules are incrementally induced —one at a time— and when each new rule is added to the list, all examples it covers —positive and negative— are removed from the set of training examples. Thus, each rule aims to separate the sample space into two sets: examples that belong to a particular class and examples that cannot be safely classified at that time. However, when rules are used as features, they might overlap and a training example can even fire two —or more— contradictory rules. Figure 4.12 represents this situation graphically. It is the job of the maximum entropy

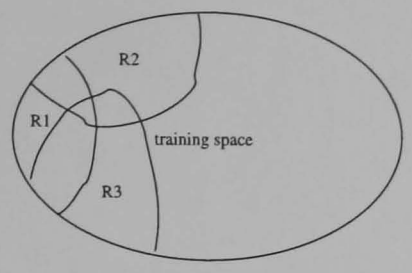


Figure 4.12: Situation when Ripper rules are used as complex features: rules might overlap.

framework to obtain the appropriate probabilities for overlapping rules, whether they predict the same class or are contradictory, as well as combine this information with the other (default) atomic features.

4.9 The new versions

Following the ideas presented in section 4.8, three new pools of features were obtained —one for each MOLI MENE version presented in section 4.5— by applying the Ripper rule inducer to the original pools of features. Ripper proved to be very efficient: none of the new pools of features took more than 45 minutes to be built.

The new pools contain the linguistically-oriented features, both atomic and complex, that Ripper considers most useful for performing the extraction task. Appendix C presents the text representation for these hypotheses as given by the implementation of Ripper in use (Cohen 1996).

For experimentation, three further versions of MOLI MENE have been created: versions V6, V7 and V8, which use the new pools of features that result from applying Ripper to the linguistically-oriented atomic features of versions V3, V4 and V5 respectively.

It has been computationally infeasible to determine the exact reduction in the size of the feature space obtained by this approach. An estimation for 10 training documents was performed. Results indicate that 382,869,287 unique complex features —that occur 2,572,661,588 times— were produced for MOLI MENE V3 from the atomic features contained in these documents. Ripper, when considering all training documents, selects just 16 complex features that use only 37 distinct valued atomic features. Therefore, the reduction in the size of the problem is quite important.

4.10 Results of the new versions

Figure 4.13 presents the performance obtained by MOLI MENE V6, V7 and V8, as defined in section 4.9, on the MUC-7 dryrun test corpus. All parameters have been

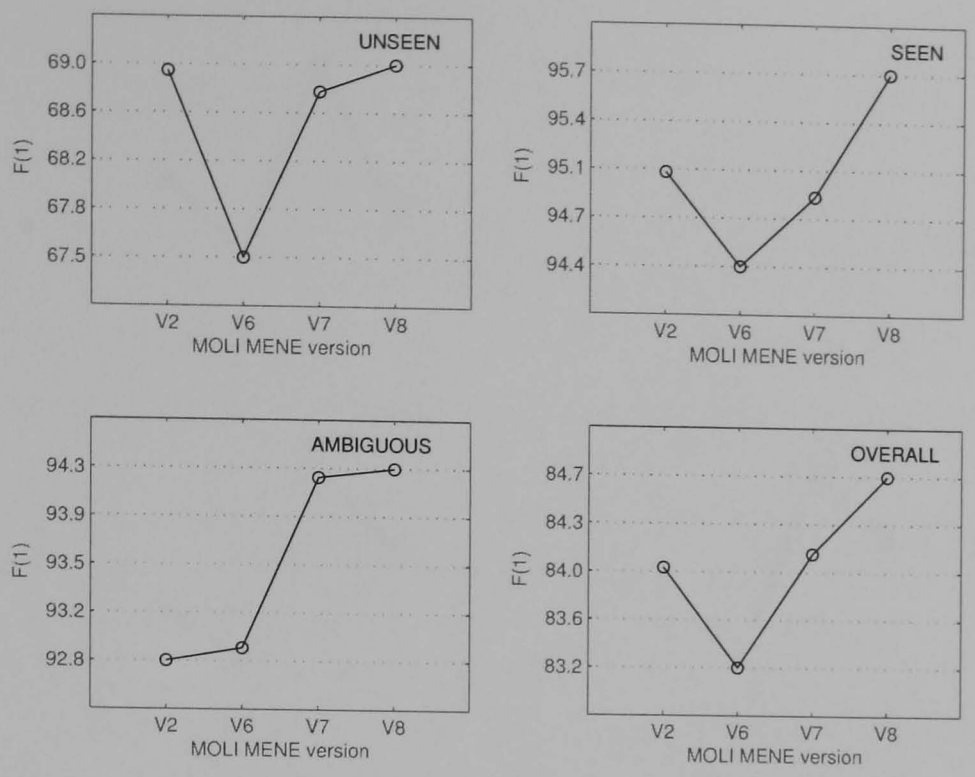


Figure 4.13: Comparison of the performances of version V2, V6, V7 and V8 of MOLI MENE: data is represented in FMLU notation, context windows are set to [1,1], [2,2] and [4,4] respectively, cutoff is set to 3 and GIS iterations to 200. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

kept fixed to the values determined in section 4.2, V6 uses a chunk window of size [4,4] and V8 considers all trigger synsets.

The first observation is that version V6 fails to outperform the simpler version V2 overall, but versions V7 and V8 do obtain slightly better overall performance.

The lack of contribution of chunk tags and head words has been somehow validated by recent experiments (CoNLL-2002 and CoNLL-2003) in which several authors have reported the same effect, though it could also be attributed to the noise introduced by the MBSP parser which, after all, has been shown quite accurate in a different source of text and no results are known for the MUC-7 documents (see section 4.4).

The key factor for the success of MOLI MENE V7 and V8 seems to be the improvement in the performance of extracting ambiguous named entities. For this familiarity type, Ripper appears to capture relevant information from the synsets provided by WordNet which, even without applying any explicit effort for disambiguation, allows the system to decide better whether an ambiguous phrase is acting as a named entity or not.

A second important observation is that Ripper seems to effectively reduce the size of the set of training complex features, as the generalised iterative scaling algorithm is able to estimate an appropriate set of weights for MOLI MENE's maximum entropy model in just 200 iterations.

Comparing these results to versions V3, V4 and V5 (figure 4.8), a substantial improvement in the performance can be observed. The performance obtained by V7 with 200

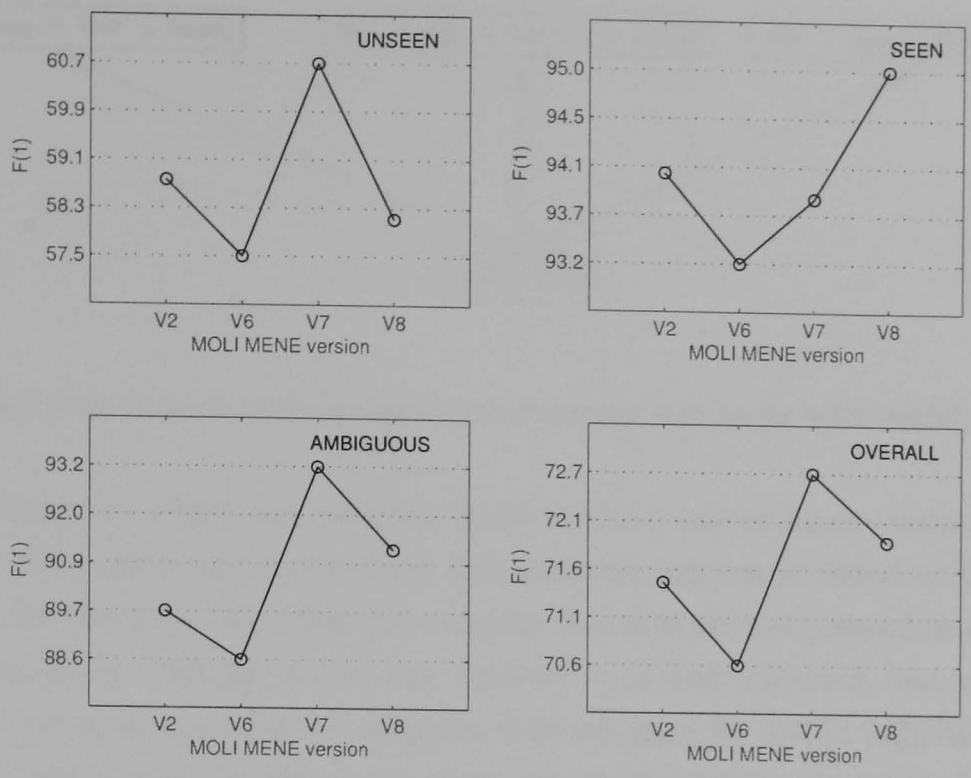


Figure 4.14: Comparison of the performances of version V2, V6, V7 and V8 of MOLI MENE: data is represented in FMLU notation, context windows are set to [1,1], [2,2] and [4,4] respectively, cutoff is set to 3 and GIS iterations to 200. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

GIS iterations is about 12 F-score points higher than its counterpart V4 when trained with 1,000 GIS iterations. This is quite remarkable as both versions share the same pool of atomic features. This confirms that Ripper is selecting the most relevant atomic features and generating useful complex features.

Figure 4.14 shows the results obtained by repeating the above experiment on the MUC-7 formal test corpus. The overall performances follows the trend observed on the dryrun test corpus. However, MOLI MENE V8 is not the overall best system, as on the former corpus, and it is outperformed by version V7. This seems to be due to an important increase in the performance of V7 on unseen named entities, which was not observed in the dryrun test documents. Recall that the MUC-7 dryrun test corpus was selected from the same domain as the MUC-7 training corpus, whereas the MUC-7 formal test corpus is a collection of documents from a slightly different domain. These facts suggest that gathering information about close synonyms might contribute with a better generalisation for the identification of unseen named entities when they do not follow exactly the patterns seen in the training text.

4.11 Adding generalisation of features

In section 4.8, a method for feature selection based on collocations of features — proposed by Mikheev (1998)— was discussed. One of the most interesting characteristics of this approach is that it provides a simple way of performing generalisation

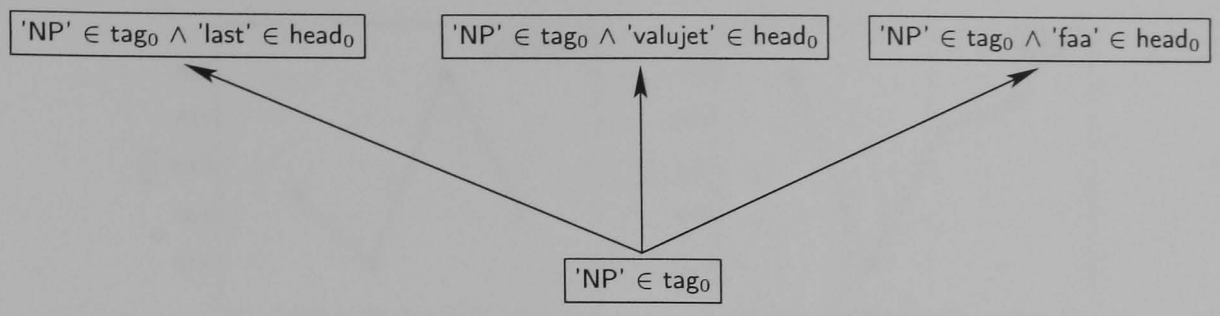


Figure 4.15: Example of the generalisation of complex features for MOLI MENE V6.

of complex features: when two complex features share one or more atomic features, in conjunction with other atomic features, this common section is added to the pool as a new feature. In this way, when the two complex features are not present in the decoding data, the maximum entropy model can back-off to a less informed, but more general feature. For example, consider the complex features used by MOLI MENE V6 that are presented in figure 4.15. In this case, there are three complex features that have one common atomic feature, namely that the focus chunk is labelled as a noun phrase. The generalisation procedure will add this common atomic feature into the pool as a weaker indicator, which can nevertheless be useful in the absence of the complex features.

This idea has been easily introduced into the MOLI MENE versions that obtain complex features through Ripper. After the list of rules induced by Ripper has been transformed into complex features, discarding the classes predicted, they are considered as observed feature collocation nodes in a lattice. Then, a new process takes these collocations and builds the next level in the lattice. This corresponds to creating new feature collocation nodes from each original node by removing one of the features in the collocation, that is one condition from the rule. Then, empty nodes, nodes that already exist in the lattice and nodes that support only one node of the original level are removed from the new level. This procedure is repeated until no new nodes can be obtained. The features resulting of this generalisation procedure are also detailed in appendix C.

4.12 Results of generalisation

Applying the generalisation procedure described in section 4.11, three new versions of MOLI MENE have been obtained: V9, V10 and V11. These versions consider the generalised pools of features of versions V6, V7 and V8 respectively.

The generalisation process turned out to be quite fast and only took a few seconds for each list of rules. The main reason for this efficiency is the moderate size of these lists: V6 uses only 16 complex features, which are extended to 18; V7 manages 221 Ripper rules, which are expanded to 252; and V8 utilises 193 conjunctions, which are generalised to 211 features.

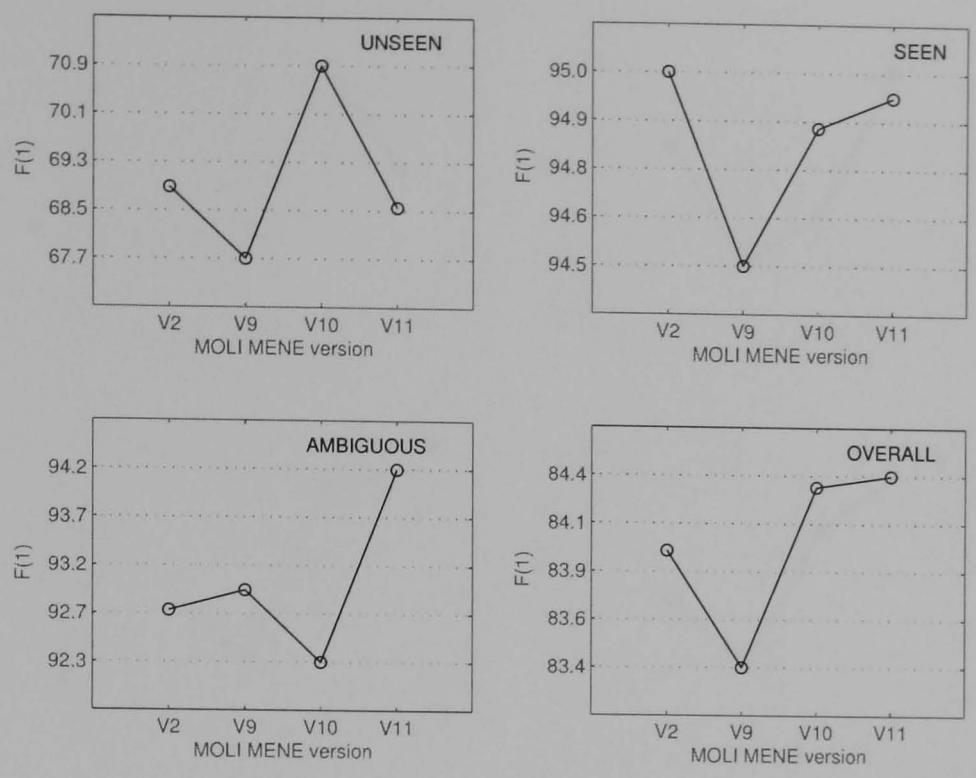


Figure 4.16: Comparison of the performances of version V2, V9, V10 and V11 of MOLI MENE: data is represented in FMLU notation, context windows are set to [1,1], [2,2] and [4,4] respectively, cutoff is set to 3 and GIS iterations to 200. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

Figure 4.16 presents the performance obtained by MOLI MENE V9, V10 and V11 on the MUC-7 dryrun test corpus. It can be seen that generalisation has had only a marginal effect on the performance of MOLI MENE V6, V7 and V8 on this test corpus. Versions V6 and V7 are helped slightly by the introduction of generalised rules, but V8 has decreased its performance a little.

Interestingly, the improvement in recognising unseen named entities obtained by features that use close synonyms that was observed for the MUC-7 formal test corpus, has also appeared here, at the cost of a slight decrease in the accuracy on ambiguous named entities.

Figure 4.17 shows the results for this experiment when repeated with the MUC-7 formal test corpus. It can be immediately noticed that there is a similarity between these curves and those of the not-generalised versions. MOLI MENE V10 is even better at identifying unseen named entities than V7, though a small decline in the performance on ambiguous named entities can be observed.

Another interesting effect of this experiment is that MOLI MENE V11 also presents a decrease in the performance on ambiguous named entities with respect to the dryrun corpus, but this time this fall is not enough to prevent this version outperforming the other systems.

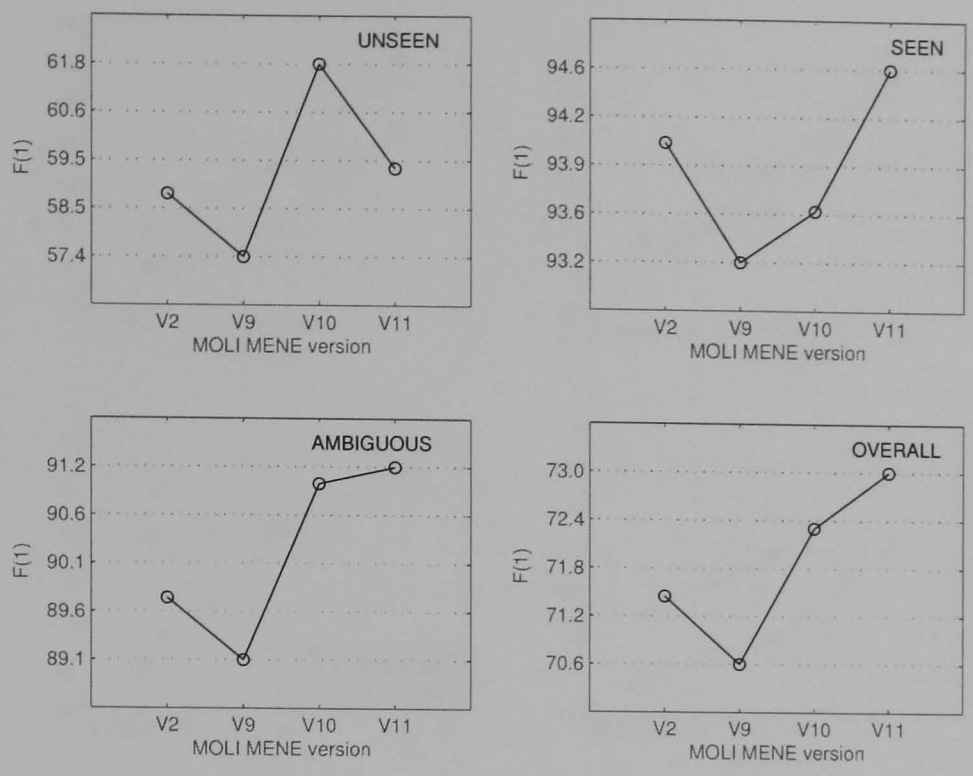


Figure 4.17: Comparison of the performances of version V2, V9, V10 and V11 of MOLI MENE: data is represented in FMLU notation, context windows are set to [1,1], [2,2] and [4,4] respectively, cutoff is set to 3 and GIS iterations to 200. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

4.13 Ripper as a baseline system

Questions may have arisen from the discussion in the previous sections about whether Ripper might be accounting for all the improvements observed in later approaches of MOLI MENE, and whether this approach alone could obtain the same —or even better— results for this task. These are reasonable questions as Ripper is itself a strong classifier and it has been successfully applied to other NLP tasks (Cohen 1996).

Therefore, two NEE approaches based on Ripper have been designed to clarify these open questions. Unfortunately, it is not possible to conduct these experiments with the same data as used by MOLI MENE. Because Ripper's rules do not predict probability distributions, but a single class for each token, MOLI MENE's training data cannot be used as it is in FMLU notation which requires the application of a Viterbi search in a post-processing step. Nevertheless, beside the difference in the notation of named entities' classes, Ripper is provided with the same features and parameters as used by MOLI MENE.

The first Ripper approach is named R(V2+R(V5)) as it uses rules obtained from MOLI MENE V2's lexically-oriented features in addition to rules induced from MOLI MENE V5's linguistically-oriented features, that is mimicking the approach used for version V8 but replacing the maximum entropy model by a Ripper induction step.

The second Ripper approach is named R(V2+V5) because it applies Ripper's rule induction on the pool of atomic features that result from combining MOLI MENE V2's

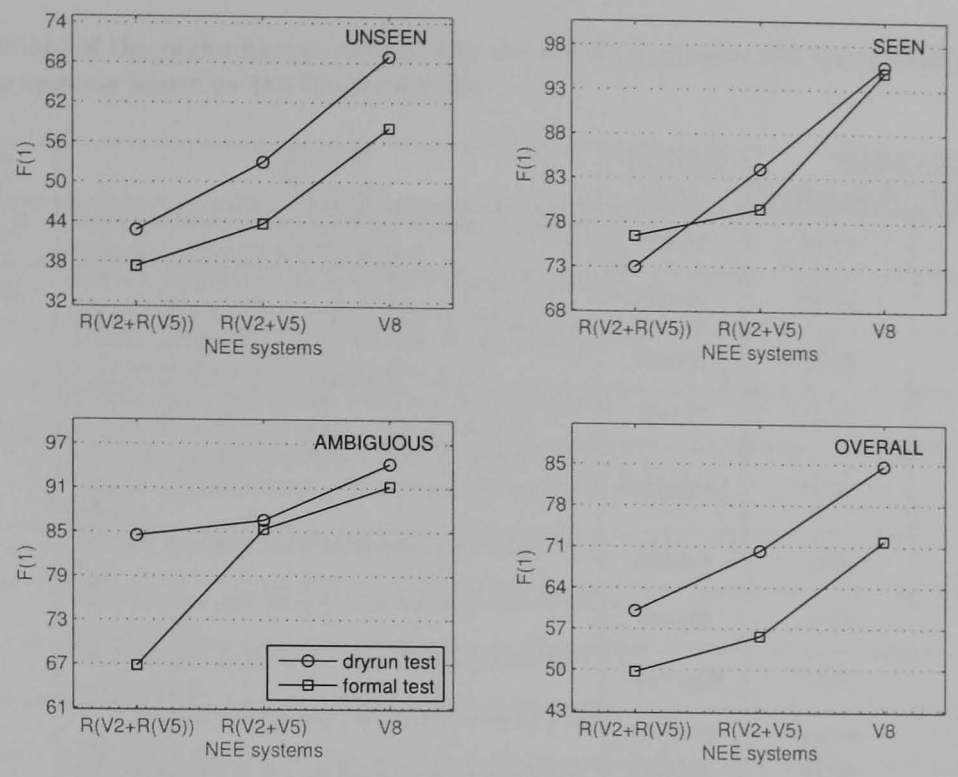


Figure 4.18: Comparison of the performances of R(V2+R(V5)), R(V2+V5) and version V8 of MOLI MENE. Corpora: MUC-7 training corpus and MUC-7 test corpora.

and MOLI MENE V5's features.

Figure 4.18 presents a comparison of the above Ripper-based systems and MOLI MENE V8. The latter utilises the abilities of both Ripper and the maximum entropy framework on the same features used by the Ripper-based approaches.

The results are conclusive: the combination of Ripper and maximum entropy models yields much better performance on the extraction in all familiarity types on both test corpora.

4.14 Summary and discussion

Table 4.3 presents a summary of the best F-scores obtained in the experiments conducted in this chapter. It also includes the results for the baseline systems reported in sections 3.4.2 and 3.4.3. The training time included in this table for each system has been obtained in a normal desktop computer, with the exception of MOLI MENE V4 (marked with an asterisk) which was trained in (a window of time of) a bigger machine provided by the York White Rose Grid.

The first observation is that the more informed features seem not to have a great impact in the performance of the system. Indeed, the best F-scores of MOLI MENE are just 0.70 and 1.56 higher for the dryrun test and formal test corpora respectively, with

Table 4.3: Summary of the performances obtained by the baseline systems, the approaches reported in this chapter and two systems based on the Ripper inducer.

Version Name	Pool of Features	Training Time	Performance	
			Dryrun test	Formal test
siNymble	lexicals (tokens) in a [1,0] window, binary lexicals with precedence in a [1,0] window	4m37s	81.20	66.12
LexMENE	lexicals (tokens) in a [2,2] window, binary lexicals without precedence in a [2,2] window, zone	19m55	79.54	64.51
V2	lexicals (tokens) in a [1,1] window, binary lexicals with precedence in a [2,2] window, zone	15m13	83.98	71.49
V3	V2's features plus chunk tags in a [4,4] window and head words of chunks in a [4,4] window	50m51s	71.71	57.69
V4	V2's features plus lemmas in a [1,1] window, PoS tags in a [2,2] window, close synonyms in a [1,1] window	9h48m18s*	60.81	51.98
V5	V2's features plus trigger synsets hit in a [2,2] window	46m46s	73.81	61.64
V6	V2's features plus Ripper rules induced from V3's new features	21m46s	83.22	70.61
V7	V2's features plus Ripper rules induced from V4's new features	1h16m42s	84.09	72.65
V8	V2's features plus Ripper rules induced from V5's new features	1h1m57s	**84.68**	71.93
V9	V6's features but Ripper features are generalised	22m02s	83.36	70.64
V10	V7's features but Ripper features are generalised	1h18m10s	84.54	72.60
V11	V8's features but Ripper features are generalised	1h02m52s	84.38	**73.05**
R(V2)+R(V5)	Ripper rules induced from V2's features plus Ripper rules induced from V5's features	13m8s	60.02	49.68
R(V2+V5)	Ripper rules induced on V2's and V5's features	16m17s	70.22	55.68

respect to the performance obtained by V2 which uses only the uninformed pool of feature of LexMENE. This represents about a 1% improvement.

The explanation of this result might lie in the great improvement obtained by just re-arranging the lexical-based features from LexMENE to MOLI MENE V2. This parametrisation meant a 6% and an 11% improvement respect to the maximum entropy baseline system, making the approach hard to improve on.

Although marginally, WordNet features seems to be contributing to the identification of more unseen and ambiguous named entities. This fact deserves further investigation, as a better parametrisation of the model or the Ripper algorithm could lead to more significant improvements.

However, it might be the expected that the introduction of naïve more-informed features —which require no human intervention, though— will not be able to increase the performance much. Thus, more elaborated features, or at least a better parametrisation, are needed in order to obtain greater improvements. For example, it is likely that by applying even a simple word sense disambiguation algorithm, the trigger synsets features could provide more valuable information; neither is it clear that a window of [1,1] is the correct setting for this type of feature.

Moreover, it might be possible to boost the contribution of the more linguistically informed features by combining them with other kinds of features which have been

shown to be useful for named entity recognition, like those used in the maximum entropy approaches presented in the CoNLLs (Roth and van den Bosch 2002, Daelemans and Osborne 2003).

Nevertheless, this chapter has made an important contribution by introducing and empirically testing a new way of selecting relevant complex features from a huge space of possible constraints for the maximum entropy model. The 1% increase in performance discussed above could have not been possible without this selection method, as experiments with MOLI MENE V3, V4 and V5 show. Version V4 was not near to being able to rival the simpler version V2, even when the GIS algorithm was allowed to run 1,000 iterations.

This method is simple, general and fast, mainly because of the efficiency of the Ripper algorithm, and it may be suitable not just for maximum entropy models but for other models too that have difficulties with large and sparse spaces of features, such as memory-based algorithms.

Chapter 5

Biasing LexMENE

This chapter present a different approach to extending LexMENE than the one presented in the previous chapter, which aims to assess the validity of the second hypothesis proposed in section 2.3. The idea is to bias LexMENE towards examples that are *similar* to the piece of text being classified.

In this chapter, the reasons to believe that making this biasing might improve the performance of LexMENE are discussed. Then, the strategies to represent and retrieve similar pieces of text are presented. Finally, experiments with the approach are explained and evaluated.

5.1 Why biasing LexMENE?

There are two main reasons to expect that LexMENE could benefit from considering similar training examples when classifying new text:

- ▷ it might help LexMENE to recognise *exceptions* —i.e. low-frequency named entity occurrences— in the text
- ▷ LexMENE could implicitly use the information utilised to obtain similar training examples, if this information is different from the one it considers for the classification

These two arguments can be better explained with some examples. Consider the classification of the last noun phrase in the sentence *we're going to Paris*, in which a location is found. It is very likely that there are many similar pieces of text in which the same

lexical features are present but not associated with any location: *we're going to make a study*, *we're going to the party*, *we're going to the NLP Conference*, etc.

If this kind of irrelevant sentence is highly frequent in the training texts —or for that matter in the decoding documents— LexMENE's approach, being a statistical one, would be inclined to overlook the sample sentence and not consider the possibility of the presence of a named entity in it.

Even if these irrelevant sentences are not frequent enough to dominate LexMENE's decisions, they are nonetheless considered in the model. A human reader would recognise the inappropriateness of considering these sentences just by realising that the modifier/argument of the verb *going* in the sentence *we're going to Paris* is a noun phrase whose head word is not preceded by a determiner and written in capitalised style, whereas none of the irrelevant sentences meet all these conditions.

Therefore, although it would be rather difficult to include all this knowledge in LexMENE, some of this information can be captured if the system takes into account more examples that contain —for this particular piece of text— locations: *she's going to London tomorrow*, *they're flying to Washington*, *we're going to the City for a meeting*, etc.; and fewer example of similar texts which do not contain locations. In this manner, features in LexMENE will be *biased* towards locations.

The example above will be used for its simplicity, but the same principles apply to more complex examples. For instance, the word *Clinton* is very frequently seen as a person name: *President Clinton, who received a letter...*; *...the foreign policy of the Clinton administration...*; etc. These are irrelevant pieces of text when the —much rarer— piece of text *...spokesman for Foster Wheeler Corp. of Clinton, N.J., said...* —in which the word *Clinton* acts as the name of a location— is submitted for the extraction of named entities. For this rare example, text such as *...arrived into Port Newark, N.J., from...*; *...Karen Harris of Irmo, S.C., took a...*; and *...by Aviation International of China*; are more useful to obtain a correct classification. It is clear that a statistical-based approach to named entity extraction could easily be deceived by the disparity in the frequency of these two events.

There have been some attempts at forcing a classifier to consider infrequent training examples through boosting (Freund and Schapire 1999). In the Shared Task of the versions 2002 and 2003 of the CoNLL Conferences (Roth and van den Bosch 2002, Daelemans and Osborne 2003), five NEE systems that utilised boosting were presented, with different levels of success.

The boosting meta-learning method tries to produce accurate classifiers by combining the predictions of several simple, moderately inaccurate classifiers. As such, the boosting algorithms of the AdaBoost family train a set of weak classifiers sequentially in a series of

rounds. In each round, the weak learner focuses on the examples which were incorrectly classified in the preceding round.

The problem with the boosted approaches above is that the final hypothesis is a weighted majority vote of the weak predictions based on their *global importance*, which is estimated in inverse proportion to the number of errors they make. Therefore, this linear combination is fixed for all decoding examples, wasting the ability of certain classifiers to perform better on particular exceptional examples.

The baseline system which will be extended here uses a maximum entropy model to obtain its predictions and one previous work has shown that AdaBoost did not improve the performance of this kind of classifier when applied to a shallow parsing (Park and Zhang 2002).

These last observations are suggesting that a different approach should be used to provide LexMENE with a better handling of exceptional named entities.

5.2 Formalisation

To formalise the basic idea behind Biased LexMENE, hereafter biLexMENE, some vocabulary from memory-based learning methods will be utilised, in particular the discussion and terms in Burkhard (1998) will be followed.

Given a decoding piece of text, hereafter a *query*, that contains a named entity of class c, then the training examples, hereafter *cases*, considered by the system should contain similar pieces of text that also contain named entities of class c. Unfortunately, this would require to know *a priori* the named entities contained in the query, which is exactly what the system is trying to determine.

Therefore, biLexMENE will be able to make only *approximated biases* for each query. These approximations must be based on some sort of *similarity* which increases the likelihood of *retrieving* cases with the same sort of named entities contained in the query.

Memory-based approaches seem to fulfil these characteristics. In the CoNLLs, Tjong Kim Sang (2002b), Hendrickx and van den Bosch (2003) and De Meulder and Daelemans (2003) presented NEE systems which utilised instance-based techniques with moderate success only, even when applied in combination with meta-learning methods —such as stacking (Wolpert 1992)— that aim to correct errors originally made by the memory-based classifier, or when using extra information derived from unannotated data.

The poor performance of the systems above could be related to the *adaptation* method employed, that is the way in which similar cases are used to obtain the query's class. All

the above approaches utilise the k-nearest neighbours method, in which the classification for the query is determined on the basis of the majority class of the cases found in a vicinity of the query. It may be speculated that this adaptation can be affected by the presence of many irrelevant, but similar, sentences that mislead the classifier.

This problem is not unknown for memory-based methods and some attempts at pruning noisy instances from the training material have been studied (Zhang 1992). However, Daelemans, van de Bosch and Savrel (1999) showed that removing apparently useless instances —based on measures of their *typicality* and class *prediction strength* (Salzberg 1990)— normally decreases the performance of an instance-based learning method when solving an NLP task. Moreover, they found that the decrease in performance is related to the degree of deletion applied and the number of *exceptions* that are removed from the data.

5.3 The proposed approach

The extension for the baseline system proposed here is based on the combination of memory-based methods and statistical classification. The idea is to build a maximum entropy model for each query considering cases which are similar to that query, in hope that the selected training instances will contain the same kinds of named entities as the ones contained by the query.

With this approach, there is no need for relying on a potentially inaccurate generalisation, but locally-biased hypotheses can be obtained. On the other hand, because the adaptation is more complex than just considering the classes of neighbour training instances —but also adding feature frequencies into consideration— it can be speculated that there are more chances of escaping the misleading noisy cases.

The hypothesis is that this characteristic will help biLexMENE to face two important problems. First, because feature frequencies are considered from a selected number of training examples only, less training material is needed to obtain accurate classification of infrequent events. And secondly, selecting similar cases might break the imbalance of the data which is inherent in named entity extraction tasks —Coates-Stephens (1992) found that only about 10% of newswire text consists of proper names— helping biLexMENE to take better decisions on infrequent or unknown pieces of text.

5.4 Getting cases and queries

The first step in the new approach is to divide the training corpus into cases and the test corpora into queries. Words —or even tokens— could be used as possible cases/queries

units. However, this would probably create too many cases/queries to be practical.

Windows of words would be more appropriate but they present a major difficulty: deciding their boundaries. A fixed-length window does not seem applicable because a named entity could become separated into two different windows, affecting its similarity with other cases/queries. A variable-length window would again need to know the named entities in the text, so that no boundaries are placed in the middle of them.

With this in mind, *constituents* —i.e. basic chunks identified by a shallow parser— will be used as the basic unit for cases and queries. There are a number of reasons for this decision:

- ▷ a constituent has a better linguistic meaning than a random window of words
- ▷ a shallow parser, namely the MBSP parser (Daelemans, Veenstra and Buchholz 1999), can be used by MOLI MENE to recognise boundaries between constituents with relative confidence
- ▷ although there are examples of named entities involving more than one basic constituent, such as *NP[the University] P[of] NP[York]*, most named entities are contained within a single constituent, normally a basic noun phrase (Collins and Singer 1999)

Now the problem has been reduced to determining the number of units that will constitute a case/query. Clearly, considering just one constituent as a case/query is of little use because, as most named entities are contained within a noun phrase, it would provide trivial information for the determination of their similarity. Thus, it is necessary to consider more than one constituent when looking for similar cases. Fortunately, now that units are limited to constituents is much easy to define a fixed-size window.

The size of the context window could be a free parameter of the system, but this size has been fixed in biLexMENE to four constituents on each side of the focus constituent[1]. This number follows the intuition that most modifiers can be contained in a window of this size. For example, appositive noun phrases —a complex common modifier in terms of the number of constituents involved— are normally contained within such a window:

left context: NP[the president] P[of] NP[Spain] CONJ[,]
focus: NP[José María Aznar]
right context: CONJ[,] VP[said] ...

left context: ... VP[said]
focus: NP[Patricia Clain]
right context: CONJ[,] NP[chairman] P[of] NP[the awarded airline]

[1]This kind of decision will be frequent in this chapter because the number of parameters that biLexMENE introduces is very large and they cannot be empirically parametrised in practice. See section 5.6.1 for an explanation.

As a result of using a window of constituents as a case/query, new information can be gathered: the structure in which a named entity occurs. This new information can be used as a part of the similarity measure between cases and queries that will drive the bias in biLexMENE.

5.5 The similarity measure

One of the main advantages of the approach proposed here is that the characteristics utilised to retrieved similar cases —i.e. the similarity measure— do not need to be based on the same features utilised to perform the classification. Indeed, the similarity function that biLexMENE uses includes information about the structure of the sentence and close synonyms, which is not considered by the underlying maximum entropy model to extract named entities.

In general terms, the context for cases and queries defined above allows the system to recognise that all the following sentences exhibit the pattern *NP VP P NP*: *she's going to London tomorrow*, *they're flying to Washington*, *we're going to the City for a meeting*, and they might be considered similar to queries that contain the same constituent pattern, such as *we're going to Paris*.

Although this pattern will differentiate sentences such as *we're going to make a study*, it will still consider *they'll fly at 10pm today*, *we're going to the party*, and *we're going to the NLP Conference* as close cases. Here is where the lexical and orthographic features of LexMENE can help.

Examining the prepositions in the cases being compared, it can be seen that *P[to]* is different from *P[at]*. This information might help the system to determine that these cases are less similar than those with the same preposition.

Analogously, it can be established that *NP[Paris]* is closer to *NP[London]*, *NP[Washington]* and *NP[the City]* than to *NP[10pm]* and *NP[the party]* by looking at the capitalisation of the head words. Nonetheless, sentences like *we're going to the NLP Conference* will also be retrieved for the example query.

BiLexMENE makes an effort to use the lexical database of WordNet®, in a similar way in which MOLI MENE uses it (see section 4.5), to reduce the number of these occurrences. Thus, biLexMENE could discover that the words *Paris*, *London* and *Washington* are semantically related[2]. This will bring closer these cases and distance the query from the irrelevant case about the conference.

[2]WordNet contains entries for these and many other locations. However, this type of semantic relation will not always be available as WordNet does not attempt to be an exhaustive database of proper names.

This seems to be an appropriate way of approximating a bias for LexMENE as the similar cases that can be retrieved for a particular query are more likely of containing the same type of named entity.

However, comparing windows of constituents properly is not a simple task. It requires a large amount of linguistic knowledge and certainly requires looking much further away at the surroundings. For example, the system should recognise constructions such as appositives —among many other linguistic phenomena— that can move around the focus constituent without changing the meaning, recognise and ignore irrelevant constituent, such as adjectival phrases, and manage combinations of these. In other words, biLexMENE should ideally be able to determine that the following pairs of texts are very similar.

... said Olin Edwards, chairman of free.com.
... said the chairman of Pratter Inc., Stephen Daark.

... flying to Florida today for ...
... flying soon to Paris for ...

This degree of understanding could be quite expensive, in terms of both linguistic resources and portability. Therefore, biLexMENE opts for a much more naïve approximation: it compares the left and right contexts of two cases separately, trying to *maximise* the match between them. This would solve many problems presented by the variability of natural languages, in particular the use of adjectival and adverbial phrases that should be ignored for named entity extraction purposes.

To get this maximum match, pairs of constituents need to be compared in the *best alignment* of the contexts of the two cases. BiLexMENE utilises a classical dynamic programming algorithm from bioinformatics to generate optimal alignments —originally for sequences of bases or amino acids— namely the Needleman-Wunsch-Sellers (NWS) algorithm (Needleman and Wunsch 1970, Sellers 1974). All adaptations of this algorithm as used by biLexMENE do not consider gap penalties or negative mismatches —that is, unmatched elements do not contribute to the similarity between cases/queries but they also do not affect it negatively— as these aspects of the general NWS algorithm shown to have little impact on the list of cases retrieved. With the NWS algorithm, the following alignment results for the last pair of sentences in the example above.

VP[flying]		P[to]	NP[Florida]	NP[today]	P[for]
VP[flying]	ADVP[soon]	P[to]	NP[Paris]		P[for]

The NWS algorithm requires a scoring function that estimates the similarity between the elements in the sequences. BiLexMENE considers several levels of information for these functions, which are explained in the following sections.

5.6 Case Retrieval Nets

The management of past cases in biLexMENE is inspired by the Case Retrieval Net (CRN) framework, proposed by Lenz (1999). The key idea of CRNs is to store both the case base and the similarity relationships among them in the same memory structure.

Two kinds of nodes can be found in a CRN: one for representing *information entities* and the other to represent *cases*. Information entities are defined as atomic pieces of knowledge, which in the simplest structure correspond to attribute-value pairs, but they might well represent text, pictures or indeed any other type of data (Lenz and Burkhard 1996). Cases, on the other hand, are defined as sets of information entities associated with unique identifiers. This definition does not imposes any restriction on the structure of a case and it seems flexible enough to capture all past events (Lenz and Burkhard 1996). Moreover, it makes evident that a comparison between cases should be based on their constituting information entities. Following these ideas:

1. information entity nodes are linked to other information entity nodes by arcs which represent the similarity between them. This similarity is expressed as a weight labelling each *similarity arc*, which is determined by a *similarity function* $\sigma : E \times E \rightarrow \Re$, where E is the set of information entity nodes in the CRN; and
2. information entity nodes are linked to case nodes by arcs which represent the relevance that the information entities have for the cases. This relevance is also expressed as a weight labelling each *relevance arc*, which is determined by a *relevance function* $\rho : E \times C \rightarrow \Re$, where E is as above and C is the set of case nodes in the CRN

The main goal of CRNs is to find all relevant cases in memory which are, to some extent, similar to a specific problem —i.e. a query— that can be used to extend the information contained in the query. This approach for case retrieval —called *case completion*— is quite appropriate for doing the kind of cased-based classification needed here[3]. Indeed, the extraction task can be modelled so that each case contains a known special information entity, namely the named entity class, which is not known for queries. Under this view, *a query is just a case whose named entity class is unknown.* Therefore, CRNs try to find sufficient similar cases, on the basis of the other known information entities, to deduce the missing class of a query.

[3]Case completion is considered a different approach from cased-based classification (such us the k-nearest neighbours method). The main difference is that in cased-based classification only a small set of cases —or even just one case— is retrieved, which is then interpreted as containing the class of the query; whereas in case completion the goal is to retrieve all relevant information in memory for the query.

CRNs perform case retrieval in a bottom-up fashion re-constructing the cases from memory in a process called *activation*. The activation procedure is performed in three clear steps:

1. first, information entities which are present in the query are initially activated
2. then, this activation is propagated to other information entities following the similarity arcs; a newly activated information entity obtains an activation value which is proportional to the corresponding weight in the similarity arc
3. finally, these activation are collected in the associated case nodes according to the weights in the relevance arcs

In this way, an activated case is assigned an activation value that is proportional to the activation of the information entities that define it, whose activation values are in accordance to the information entities that define the query. Thus, the higher the activation value of a case, the more similar it is to the query.

Lenz (1999) showed that the theoretical complexity of the case retrieval procedure in CRNs is not worse than the linear search through the case base. However, important speed-ups are reported when activation is compared to linear search (Lenz and Burkhard 1996).

It might be noticed the similarity between the retrieval of cases with CRNs and the retrieval of documents with Latent Semantic Indexing (LSI) (Deerwester, Dumais, Landauer and Harshman 1990): both approaches are three-step procedures which consist of obtaining initial matches from a query, expanding these matches according to a similarity criterion to finally gather the stored answers associated to these matches.

5.6.1 Unsolved issues

One of the main disadvantages of memory-based learning approaches is that the computational cost of classifying new instances can be high, as most of the processing is performed in decoding time (Mitchell 1997).

CRNs are a contribution in this sense because they normally provide a fast way of gathering similar cases. However, two main difficulties in applying CRNs to generic named entity extraction remain unsolved, and further study is required to overcome them. The two problems are:

- ▷ although CRNs are efficient, they still have complexity problems for domains in which the attributes can take values from a large space (Lenz 1999) — such as lexical features. Thus, biLexMENE requires great amounts of memory and exhibits unsatisfactory run time for practical applications
- ▷ CRNs introduce a large number of parameters to estimate the similarity and the relevance weights, which cannot be empirically parametrised in practice

The first difficulty as been managed by dividing the CRN into different structures, applying the storing and the activation procedures in sequential steps and providing biLexMENE with powerful computational resources. More details can be found in appendix D.

For the second issue, sensible choices that are based on intuitions have been applied. When one of these decisions is not entirely clear, small experiments have been conducted to confirm that the option selected has been correct. For example, it will be explained later in the chapter that contextual constituents closer to the focus constituent have more importance in the similarity measure between cases. An experiment in which all contextual constituents had the same importance showed that this intuitive choice was indeed correct.

5.7 Representing cases

Cases/queries are represented as three different information entities that provide a description of their lexical structure, the lexical forms that compose them and the orthographic characteristics used for these lexemes.

5.7.1 Sentence structure

The lexical structure of a case/query is represented in the CRN as *constituent pattern* information entities. A constituent pattern is the sequence of four chunk tags that accompanies the *focus constituent* on either side. Therefore, each case/query has a left constituent pattern and a right constituent pattern.

Constituent patterns from different contextual sides are considered unrelated and, consequently, similarity arcs exist only between left constituent patterns and between right constituent patterns, but not between a left constituent pattern and a right constituent pattern.

The weight on a similarity arc that connects two constituent patterns corresponds to the sum of the similarity of each chunk tag in the best alignment that the NWS algorithm

	VP[flying]		P[to]	NP[Paris]		P[for]		
	VP[flew]		P[to]	NP[London]		P[for]	=	12
ADVP[soon]	VP[flying]		P[to]	NP[Washington]			=	11
	VP[arrived]	ADVP[Monday]	P[in]	NP[York]			=	11
	VP[coming]		P[to]	NP[London]	NP[today]		=	11
	VP[prepared]		P[to]	NP[look]		P[for]	=	9.75
	VP[come]	ADVP[later]	P[to]		VP[propose]		=	8.75

Figure 5.1: An example of constituent pattern information entities and their similarity weights with the constituent pattern produced by the text *flying to Paris for*.

can obtain. As explained above, this algorithm requires an estimation of the similarity between the elements in the sequences, which in this occasion correspond to the chunk tags in the patterns. This estimation is straightforward:

1. if the two tags are different, they have zero similarity
2. otherwise, a similarity weight is given depending on the minimum distance of the matching tags from the focus tag; matches closer to the focus are considered more relevant and are assigned higher weights
3. penalties are applied to this similarity weight when the tags are in different positions within the patterns and/or they are optional constituents (i.e. adjectival phrases, adverbial phrases or verb particles).

Unfortunately, the number of similarity arcs produced in this way is too large to be directly added to the CRN, as any two patterns that share at least one chunk tag in the sequence are linked. Therefore, a *heuristic restriction* —in Lenz and Burkhard's (1996) terms— has been applied and patterns whose closest constituents to the focus do not match perfectly are discarded. This pruning has considerably reduced the number of arcs maintaining only the links between constituent patterns that have more probabilities of being relevant.

Figure 5.1 presents an example to clarify the way in which this procedure operates. The reference case —which can be a query— represents the text *flying to Paris for*. It can be observed that less similar pieces of text are assigned to lower similarity weights than the ones that look more alike the reference case. Notice that the maximum similarity weight between two constituent patterns is 12. The CRN retrieval procedure, outlined in section 5.6, manages any real value (Lenz 1999).

More details of this approach can be found in appendix D.

5.7.2 Orthographic pattern

The orthographic characteristics exhibited by a case/query is added to the CRN as *orthographic patterns* information entities, which correspond to the sequences of ortho-

graphic features of the *focus tokens* —i.e. the tokens in the focus constituent— and the tokens in a orthographic window of sizes [2,2] — as defined for MOLI MENE V2 (section 4.2). Thus, the length of this kind of pattern is not fixed but varies according to the number of focus tokens contained in each case/query.

As with the constituent patterns, the weight that labels a similarity arc between two orthographic patterns corresponds to the sum of the individual similarity weights of each orthographic feature in the best alignment of those patterns, which is obtained by the NWS algorithm.

Notice that in order to work with a single orthographic pattern per case/query, each token is associated with a unique orthographic feature on the basis of their precedence — also as defined for MOLI MENE V2 (see table 4.1).

The similarity function estimates the similarity between two orthographic features by looking up the corresponding value in one of three tables of similarity weights according to whether the features describe tokens that contain numbers, tokens for normal words or other types of tokens — such as punctuation symbols.

The tables of similarity weights were built by hand, but little time was spent on this task which consisted of analysing a small set of tokens with different orthographic features, determining which of them were related and conjecturing in what proportion.

However, these tables have an important drawback: they are domain dependent. For example, the feature *number with dash* is considered more similar to the feature *number with slash* than to the feature *only digits number* in these tables, a fact that is exclusively due to the observations that dates —one of the named entities to be extracted— are written in these two styles: *05-03-2004* or *05/03/2004*.

These tables were actually intended as a first approach to testing the system, but they have remained an unsolved issue that will need to be addressed in future work.

The similarity function for orthographic features also applies penalties when matches are less informative. There are two situations considered: matches between focus features and contextual features have their similarity weight reduced; and matches between contextual features from different sides of the focus constituent are re-assigned to value zero.

Once more, the number of similarity arcs is too big to be manageable when any pair of related orthographic features is considered relevant. Consequently, biLexMENE imposes a new heuristic restriction and only orthographic patterns that have at least three consecutive perfect matches keep their arcs.

	ucp[flying]	ucp[to]	icp[Paris]	ucp[for]			
	ucp[flew]	ucp[to]	icp[London]	ucp[for]		=	48
ucp[soon]	ucp[flying]	ucp[to]	icp[Washington]			=	36
		ucp[arrived]	icp[Monday]	ucp[in]	icp[York]	=	36
	ucp[coming]	ucp[to]	icp[London]	ucp[today]		=	48
	ucp[prepared]	ucp[to]	ucp[look]	ucp[for]		=	0
	ucp[come]	ucp[later]	ucp[to]	ucp[propose]		=	0

Figure 5.2: An example of orthographic pattern information entities and their similarity weights with the orthographic pattern produced by the text *flying to Paris for*.

Figure 5.2 shows the orthographic patterns generated by the texts of the running example[4]. Note that the links between the reference case and the most different sample cases have been pruned by the heuristic restriction. Also notice that the maximum similarity value for each pair of orthographic features is 12, but that the similarity value between two orthographic patterns depends on the number of pairs of features that could match. These weights will required further processing to obtain a similarity measure for cases/queries which is independent of the lengths of their respective patterns, an issue that is discussed later in section 5.8.3.4.

For further details of this approach see appendix D.

5.7.3 Lexical pattern

Lexical information from cases/queries is introduced in the CRN as *lexical patterns*. These information entities correspond to the sequence of lexemes of the tokens/words[5] occurring in the focus constituent and in a lexical window. The lexical window is also defined as for MOLI MENE V2 (section 4.2) of sizes [1,1] on either sides of the focus constituent. As with orthographic patterns, lexical patterns may be of different lengths.

Once more, the similarity weight associated with each arc between lexical patterns corresponds to the sum of the similarity of the lexical features in the best alignment that the NWS algorithm can obtain for these patterns.

However, the similarity function for lexical features is more complex than for previous information entities. The main source of information for comparing two lexical features is their *meaning*. If meanings are not available for both lexical features, the similarity function makes use of less informed, more lexically-oriented characteristics to estimate their similarity.

[4] In this example, ucp stands for "uncapitalised token" and icp stands for "initially-capitalised token".

[5] A word in this context is a compound lexical form of more than one token which is included in the WordNet lexical database. Examples of words are Prime Minister, New York, executive-officer, etc. See appendix D for details.

When WordNet meanings can be obtained for the two lexical features to be compared, the similarity function applies the following criteria to estimate the strength of their semantic relation:

1. if the lexical features are essentially the same word, that is they have the same lemma, their match is considered perfect. For example, the pairs *president/presidents* and *says/said* both produce perfect matches
2. if the lexical features are synonyms, that is to say they are found in the same WordNet synset, their match is considered strong. For instance, the pairs *president/chairman* and *said/told* both generate strong matches
3. if the lexical features are siblings, that is they are coordinate terms in WordNet's terminology, their match is considered moderate. For example, the pairs *president/prime_minister* and *said/proclaims* are considered moderate matches
4. if the lexical features are found in a superordination or a subordination relationship, which WordNet calls the hypernym and the hyponym relations respectively, their match is considered weak. For example, the pairs *presidents/presiding_officer* and *say/announced* are considered weak matches
5. otherwise they are considered semantically unrelated

Note that this procedure also follows a similar approach to LSI (Deerwester et al. 1990), as matches are sought on concepts rather than on individual tokens.

When a comparison based on WordNet meanings cannot be undertaken, because at least one of the lexical features being compared is a token not included in the lexical database, the similarity function applies the following criteria to estimate a possible connection:

1. if the lexical features exhibit the same text, their match is considered important
2. if the lexical features are found in a list of special matches, they are considered related as defined in that list
3. if the lexical features are morphologically related, that is they have a common prefix or a common suffix, their match is given a relevance in proportion to the length of the common portion
4. if the lexical features are punctuation marks, their match is considered moderately important
5. otherwise they are considered unconnected

	flying		to	Paris		for		
	flew		to	London		for	=	42
soon	flying		to	Washington			=	30
arrived		Monday			in		York =	0
	coming		to	London	today		=	24
prepared			to		look	for	=	24
	come	later	to		propose		=	18

Figure 5.3: An example of lexical pattern information entities and their similarity weights with the lexical pattern produced by the text *flying to Paris for*.

The list of special matches mentioned in the procedure above includes:

- ▷ tokens that are not included in WordNet but that are nonetheless related, such as the word *and* and the *&* symbol,
- ▷ tokens that are not included in WordNet but that can be considered to have the same meaning as tokens that are included in WordNet, such as 're and the verb to be, the symbol % and the word percent, etc.; and
- ▷ tokens whose similarity might be incorrectly estimated by the morphological analyser, such as the fact that the numeric tokens *DD-DD-DD* is closer to the token *DD/DD/DD* than to the token *DD:DD:DD*, because the former tokens are both used to express dates, whereas the latter is used to express time

In this list, the similarity weight is explicitly given for each special match included, so that the similarity function can use this weight directly. Although not intensively used in the current implementation, this is a tool provided for specialising the retrieval of similar cases to specific languages, named entity extraction tasks and domains.

Finally, as with the similarity function for orthographic features, matches between focus lexical features and contextual lexical features have their similarity weight penalised, and matches between lexical features from different contextual sides have their similarity weight changed to zero.

Figure 5.3 presents the lexical patterns corresponding to the texts in the running example. In this example, the tokens *flew* and *flying* obtain perfect matches, because they have the same lemma as the reference token *flying*, as do the tokens *to* and *for*, because they present the same text as the reference lexical features. The tokens *coming*, *come*, *London* and *Washington* are assigned moderate similarity weights as they are found to be coordinate terms of the referential tokens *fly* and *Paris* respectively in WordNet database. However, it can be noticed that the similar piece of text —for named entity extraction purposes— *arrived Monday in York* is assigned a zero similarity because none

of its lexical features was found to be related to the reference text, even though it can be argued that *flying* might be understood as *arriving* by plane to some place, and that there exists a place named *York*. The former is a semantic relationship that is hard to obtain from WordNet (de Boni 2004) and the latter is an omission of this sense for the word *York* in WordNet.

For more details on this approach, appendix D can be consulted.

5.8 Case retrieval

Section 5.7 explained the approach followed to represent and store past cases in memory. This section explains the procedures for obtaining an effective retrieval of cases from the CRN.

5.8.1 Reducing the size of the problem

As stated earlier, the application of memory-based approaches can be costly. This is specially true for biLexMENE that has to build a maximum entropy model —which can be expensive *per se*— for each query.

Therefore, a heuristic was introduced in the approach to reduce the number of queries that need to be processed. Basically, a query is processed when it contains a *relevant* lexical feature, that is a token which might be part of a named entity.

The heuristic is applied to each focus lexical feature in the query sequentially. It first analyses the frequency of the token and determines if it is *very probable* that the token is relevant or if it is very probable that the token is not relevant. In both situation, the heuristic stops with an answer "process" or "no process" respectively.

When this decision cannot be safely obtained from frequencies, the heuristic utilises a binary classifier which estimates the probability of the token being not relevant. If this probability is *very high*, the heuristic answers "no process". Otherwise it answers "process".

Despite applying this heuristic with conservative parameters, the number of queries that biLexMENE processes is significantly reduced and with a tolerable loss of recall. For the MUC-7 dryrun test corpus, this number drops from 53,829 to 14,489 queries, which include 6,068 out of the 6,163 queries that contain named entities. Thus, the application of the heuristic delimits the maximum recall for biLexMENE to 98.46%.

Further details of this heuristic can be found in appendix D.

5.8.2 Gathering similar cases

The first step to retrieve similar cases is to obtain a representation of each relevant query in terms of the information entities presented in section 5.7. This is done with the same algorithms used to transform training cases into information entities, as queries are just incomplete cases from the CRN perspective.

The next step is to look up the information entities that represent the query in the CRN and activate them. In the normal activation process of basic CRNs, this search is enough to activate all the information entities that represent the query. This is so because information entities are atomic —by definition— and it is assumed that the values they can have are those contained in the CRN.

It is clear that these assumptions cannot be made in the approach discussed here, especially in the case of lexical patterns as it is almost certain that some queries will contain unseen lexemes that will unavoidably create unseen patterns.

The solution to this problem passes through separating information entities into *microfeatures* (Lenz and Burkhard 1996). This can be naturally applied to the information entities defined previously as they are patterns of constituent, lexical and orthographic features. The idea is that each microfeature in an information entity —i.e. a feature in a pattern— can be individually activated and the activation of an information entity corresponds to the weighted sum —in the straightest combination— of the microfeature-level activations.

In this way, it is unlikely that a query does not activate any memorised pattern. When a query's information entity is found in the CRN, the pattern is activated with maximum value —which also corresponds to the weighted sum of the activation values of all of its features— and then, the similarity arcs are followed to activate similar information entities. On the other hand, if a query's information entity is not found in the CRN, microfeatures are activated instead and all patterns that share some of these features are activated.

The final step is to follow the relevance arcs and activate the cases that are linked to activated information entities. Essentially, the activation value of a case corresponds to the sum of the activation values of its information entities, though the approach required for biLexMENE is slightly different as explained below.

5.8.3 Obtaining final similarities

The normal activation procedure in CRNs, which was presented in section 5.6, cannot be directly applied on the cases/queries as defined previously. This is because the

activation of the features in the information entity patterns have considered up to now disassociated information. This section explains the approaches to re-estimate and combine these activation values into activation values for cases.

5.8.3.1 Activation of constituent patterns

The activation of constituent patterns cannot be made following the standard procedure, as defined for basic CRNs, because that would only consider the chunk tags assigned to each constituent in the case/query, which produces an undesired flatness of the activation values.

For example, if the query's pattern contains the noun phrase *NP[the President]* and two memorised patterns contain, in the same position, the phrases *NP[the agenda]* and *NP[the Prime Minister]* respectively, then both information entities are activated with the same value, though clearly the concept *President* is semantically closer to the concept *Prime Minister* than the concept *agenda*.

To solve this discrepancy, the activation values of constituent patterns are re-calculated to also consider the head word of the constituents. The comparison of head words is similar to the calculation of the similarity between lexical features described in section 5.7.3. In appendix D, this algorithm and more details of this procedure can be found.

5.8.3.2 Activation of lexical patterns

The activation of lexical patterns cannot follow the standard procedure of basic CRN either because when the most similar cases are selected for a query (section 5.8.4), it can be possible that some lexical features in the query become uncovered for later processing. This is specially problematic for uncommon tokens, which are often part of named entities (recall the discussion in section 3.2), as they are directly responsible for the activation of few cases only.

This problem is managed in biLexMENE by maintaining different rankings of similar cases, one for each lexical feature in the query. This requires the computation of tailored activation values of all cases for every token in the query. Fortunately, this computation is naturally performed by applying the lexical window.

Consider the example of figure 5.4, in which the similarities between the lexical features of the cases *flying to Paris for* and *flew to London for* are re-computed. The lexical window is initially centred at the first focus token *flew*. Then, this window is enforced and the activation of the microfeature *flew* is estimated by adding the similarities of the microfeatures within the window (i.e. w'_{-1}, *flew* and *to*), which were previously

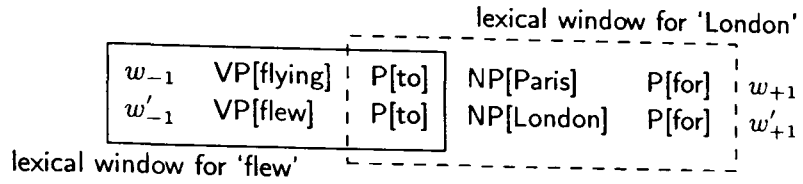


Figure 5.4: An example of the computation of the final similarity between two matching lexical features: a lexical window of size [1,1] moves from the first to the last focus token. The lexical windows applied for the words *flew* and *London* are shown in this example.

calculated as described in section 5.7.3. Then the window is shifted to the next focus lexical microfeature *to* and its activation value is re-computed. The process is repeated for every focus token in the lexical pattern.

This approach is computationally more expensive, specially in terms of space resources, but proved to be quite successful in retrieving cases for the majority of the tokens in a query. Further explanation can be consulted in appendix D.

5.8.3.3 Activation of orthographic patterns

The approach described above also requires tailored activations of cases for the orthographic features, so that the real contribution for each focus token can be determined. This process is analogous to the re-calculation procedure for lexical features explained above, in which the orthographic window is enforced at each focus orthographic feature in the query. See appendix D for a few more details.

5.8.3.4 Activation of cases

Following this approach, the final *local activation* of a case can be estimated for an individual lexical feature in a query. This corresponds to the sum of the activation values for its constituent pattern, the activation values of the focus features in its lexical pattern and the activation values of the focus features in its orthographic pattern. This creates a ranking of activated cases for each lexical feature in the query.

Nonetheless, a *global activation* is also estimated as shown in the equation 5.1, where $act_{local}(f_{lex})$ is the local activation calculated for a lexical feature f_{lex} and n is the number of focus lexical features contained in the case.

$$act_{global} = \frac{\sum_{f_{lex} \in case} act_{local}(f_{lex})}{n} \tag{5.1}$$

This normalisation is necessary because longer cases obtain higher activation values. Using the global activation, cases can be ranked according to the similarity with respect to the whole query.

5.8.4 Selecting similar cases

The final stage in the retrieval procedure is the collection of the *adaptation set*, which contains the similar cases that will be adapted for completing a query. One possibility is to retrieve all cases activated, but this would be computationally costly and might not contribute towards biasing LexMENE in the right direction. Therefore, under the premise that similar cases have a higher probability of containing the same kinds of named entities found in the reference query, only the most similar cases will be selected to produce a stronger bias.

A direct approach to selecting the most similar cases is to define a threshold. However, setting a suitable threshold value can be quite challenging because there are queries that activates thousands of memorised cases, whilst others activate only a reduced number of cases with low activation values. Applying a threshold to this latter types of queries might result in the neglect of the little information available for their classification.

Therefore, biLexMENE utilises the next simplest way, which is retrieving only a fixed number of similar cases. This requires the definition of a *sampling size*, which indicates the number of similar cases to be selected from the top of each ranking.

This value is *soft*, in the sense that if a query activates too few cases, the adaptation set will be incomplete; on the other hand, if the number of activated cases exceeds the sampling size, the actual number of cases retrieved might be higher than this size because all cases with the same activation value as the case that completes the set are retrieved.

The sampling size can have an important influence in the performance of the system, and there is not a clear method to define an appropriate value *a priori* for it. Consequently, it has been left as one of the parameters of the system.

It should be made clear here that only one adaptation set is retrieved for each query. The final size of this set depends of several variables —such as whether the query managed to activates enough cases— but in general, if the sampling size is set to m and a query contains n lexical features, then the adaptation set will have a maximum size of $(n + 1) \cdot m$ examples. This is because biLexMENE will retrieve m cases from each ranking created by the approach, that is to say from the rankings for each lexical feature and from the global ranking. However, the number of cases is normally lower because they tend to overlap.

5.9 Adaptation

As explained in section 5.8.4, for each query considered with probabilities of containing a named entity, an adaptation set of similar cases is selected. Longer queries —in term of numbers of lexical features— have more training material than shorter ones, and this material is prepared so that it contains examples for each lexical feature in the query as well as cases that are similar to the query as a whole.

The adaptation of past cases consists of creating a maximum entropy model using the cases in the adaptation set as training examples. Because biLexMENE is an extension of LexMENE, the entropy models correspond to a (new) version of this baseline system which has expanded its set of orthographic features to the set utilised by MOLI MENE (see section 4.1), which will be referred to as LexMENE-V2.

Thus, each maximum entropy model determines the probability for a lexical feature of being starting, continuing, ending, constituting or not related to, a named entity of one of the classes seen in the training examples — i.e. the FMLU notation is used, lexical and orthographic features are extracted from a context window of two tokens on either side of the focus token, all orthographic features fired by these lexical features are considered, the zone feature is also included, and lexical features that are not seen at least three times in the documents are considered unknown.

Once the maximum entropy model is ready, it is applied to each focus lexical feature in the query and a distribution over the classes seen in the adaptation set is assigned to each one of them. For queries that contain only lexical features considered irrelevant, that is to say it is unlikely that they contain a named entity (see section 5.8.1), the default distribution $P(\text{not-a-name}) = 1.0$ is assigned.

As previously, a Viterbi search is applied to each sentence to determine the best sequence of named entity tags given the distribution associated with each lexical feature. However, the probability distributions can be incomplete now, because the default distribution has been assigned or because the adaptation set did not contain all possible named entity classes in the task.

To solve this problem, biLexMENE utilises a *smoothing* approach so that the probability distribution of every lexical feature has a non-zero probability associated with each one of the 29 possible classes. This smoothing function is controlled by three parameters, namely α, β and γ, which indicate how uniform the resulting distributions should be. Appendix E gives more details and an example of how this function works. Because this is a new element introduced in the approach, the parameters α, β and γ have been left as free parameters and some experiments have been designed to look for good values for them as well as to assess the impact that this smoothing approach might have in the performance of the system.

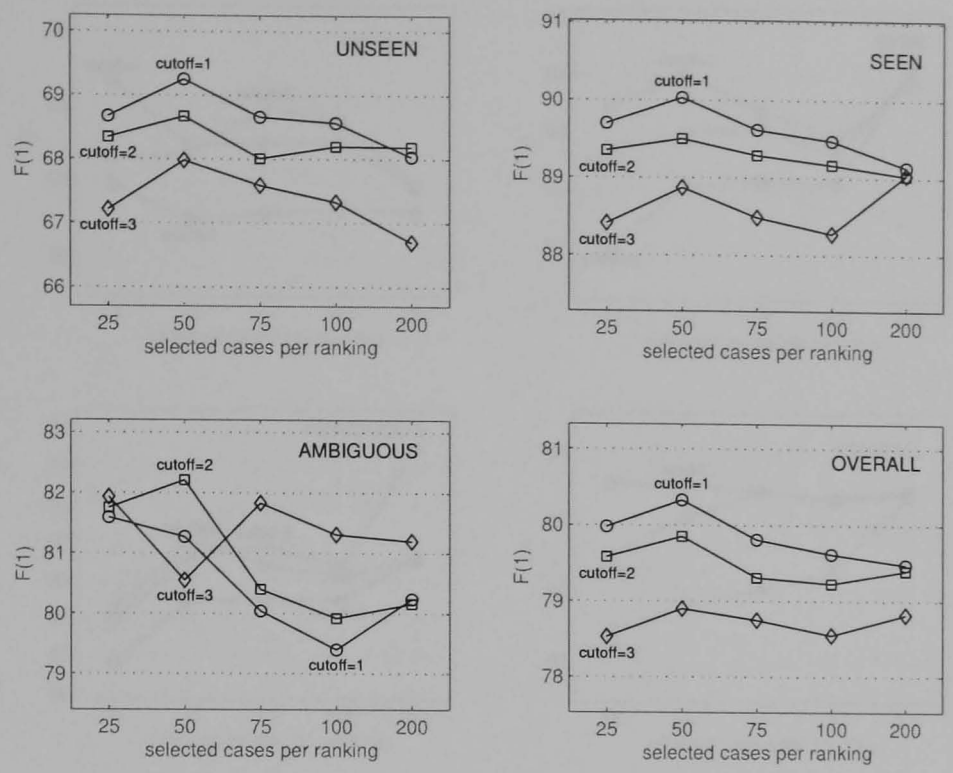


Figure 5.5: Comparison of the performances of biLexMENE for different cutoff thresholds and different numbers of selected cases: data is represented in FMLU notation, context windows are both set to [2,2] and 200 GIS iterations are performed. Cutoff varies from 1 to 3 and the number of cases selected from each ranking varies from 25 to 200. For this experiment, the same initial smoothing function is applied with values $\alpha = 0.4$, $\beta = 0.4$ and $\gamma = 0.001$. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

5.10 Experiments

5.10.1 Parameters setting

As in earlier chapters, the initial experiments in this section aim to determine a good set of parameters for building the biased versions of LexMENE-V2. In order to reduce the number of experiments needed for setting these parameters, those which were fixed for either LexMENE or MOLI MENE will be kept if there are no reasons to expect that maintaining these values would negatively affect the performance of the new approach.

In this way, all experiments presented in this section utilise the FMLU notation, 200 GIS iterations, a lexical window of sizes [2,2], a orthographic window of sizes [2,2] from which all fired features are considered, and the unknown words threshold set to three.

The first experiment aims to determine the cutoff threshold for the system. This parameter was not fixed for biLexMENE because, due to the limited training material provided to the maximum entropy models, it is not clear than the usual value three will retain the bias which is being introduced to the system. Figure 5.5 presents the results obtained for this experiment, in which the performance of biLexMENE when using cutoff thresholds at one, two and three is compared.

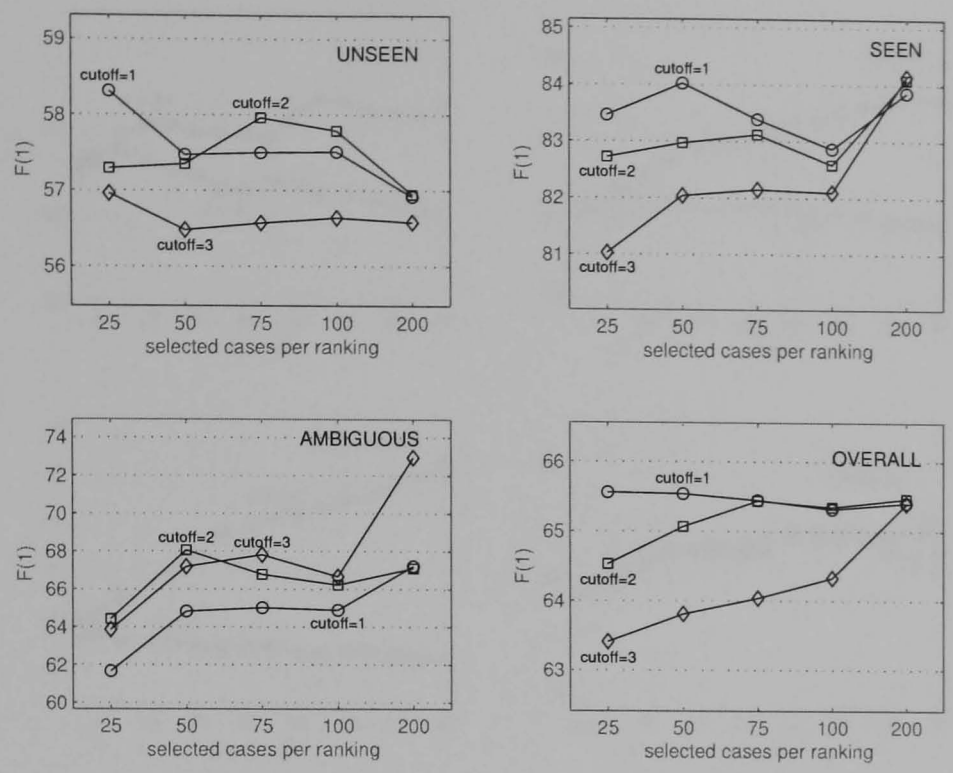


Figure 5.6: Comparison of the performances of biLexMENE for different cutoff thresholds and different numbers of selected cases: data is represented in FMLU notation, context windows are both set to [2,2] and 200 GIS iterations are performed. Cutoff varies from 1 to 3 and the number of cases selected from each ranking varies from 25 to 200. For this experiment, the same initial smoothing function is applied with values $\alpha = 0.4$, $\beta = 0.4$ and $\gamma = 0.001$. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

It can be seen from this figure that, with the exception of ambiguous named entities, biLexMENE consistently obtains better performance when the cutoff threshold is set to value one than when set to value two, which in turn produces better results than when the cutoff is set to value three.

Figure 5.5 also shows that the fewer training cases are selected from each ranking, the more significant is the difference in performance between the different cutoff values, and that the best performance is obtained when 50 similar cases are selected from each query's rankings.

These results suggest that the bias introduced by biLexMENE can be easily reduced by ignoring infrequent events —i.e. increasing the cutoff parameter— or providing too much training material for the maximum entropy models — i.e. increasing the number of cases selected from each ranking. Therefore, the cutoff threshold is fixed to the minimum possible value of one and the number of similar cases to be retrieved for each query is set to the moderate value of 50.

As in previous chapters, each of the experiments presented in this section is repeated on the MUC formal test corpus in order to established the reliability of the decisions, made from the experiments on the MUC dryrun test corpus, about the parameters that the system should use to obtain a good performance on unseen text.

Figure 5.6 presents the results obtained from this experiment. It can be observed that

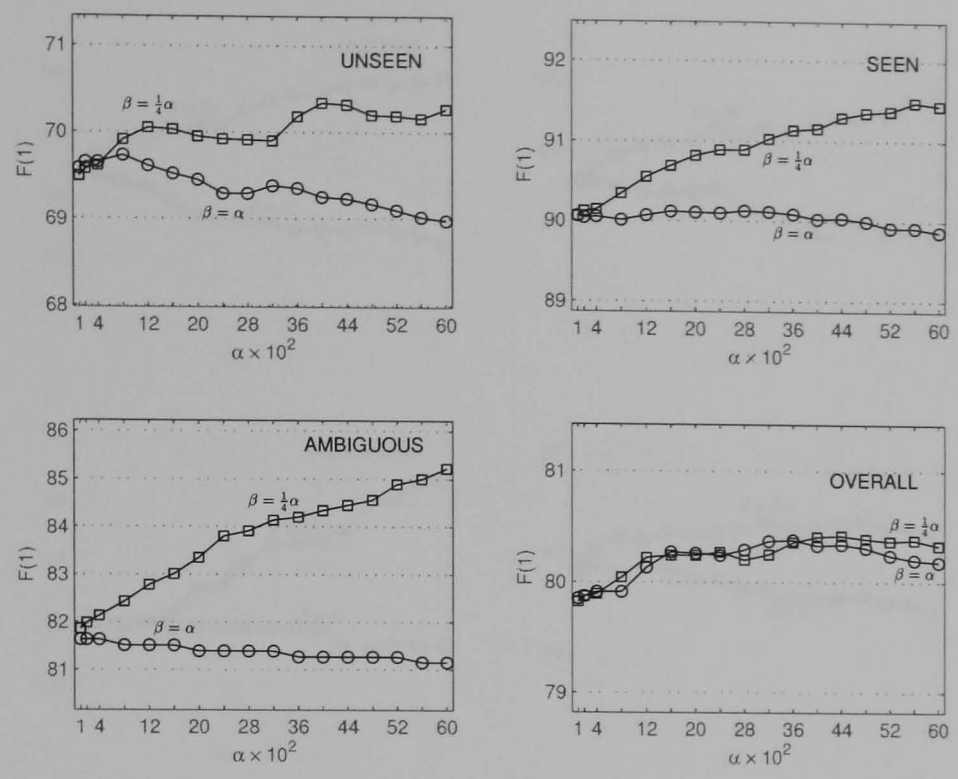


Figure 5.7: Comparison of the performances of biLexMENE for different smoothing functions: data is represented in FMLU notation, context windows are both set to [2,2], 200 GIS iterations are performed, cutoff is set to 1 and the number of selected cases is set to 50. Smoothing functions use value $\gamma = 0.001$, parameter α varies from 0.01 to 0.60 and two values are tested for parameter β: $\beta = \alpha$ and $\beta = \frac{1}{4}\alpha$. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

the influence of the cutoff parameter on biLexMENE's performance is less regular for this corpus. However, the behaviour seen on the dryrun corpus is —in general— also observed on this corpus, that is to say, the lower the cutoff threshold, the better the performance, and this difference is reduced as more and more cases are selected for training. Moreover, this last trend is much clearer here: selecting just 75 cases from each ranking levels the performances obtained with cutoff values set to one and two, and selecting 200 cases levels the performances obtained with all cutoff values tested. Therefore, although a slightly better overall F-score can be obtained by reducing the number of selected cases to 25, the parameters set considering the results of figure 5.5 seem to be good parameters for unseen texts.

The next two experiments aim to evaluate the influence of the smoothing function introduced in biLexMENE: the first one is conducted to establish the effect that the parameters α and β have in the performance of the system, whilst the second experiment attempts to determine the effect of the γ parameter.

In the first experiment (figure 5.7) the parameter α varies from 0.01, 0.02 and then from 0.04 to 0.60 in successive increments of 0.04. For the parameter β, two values are tested: $\beta = \alpha$ and $\beta = \frac{1}{4}\alpha$. These values for this parameter follow the observation that the probability associated with each named entity class is distributed between four possible tags —namely *F*, *M*, *L* and *U* depending on whether the token might be the first token, a middle token or the last token of a multi-token named entity or a named

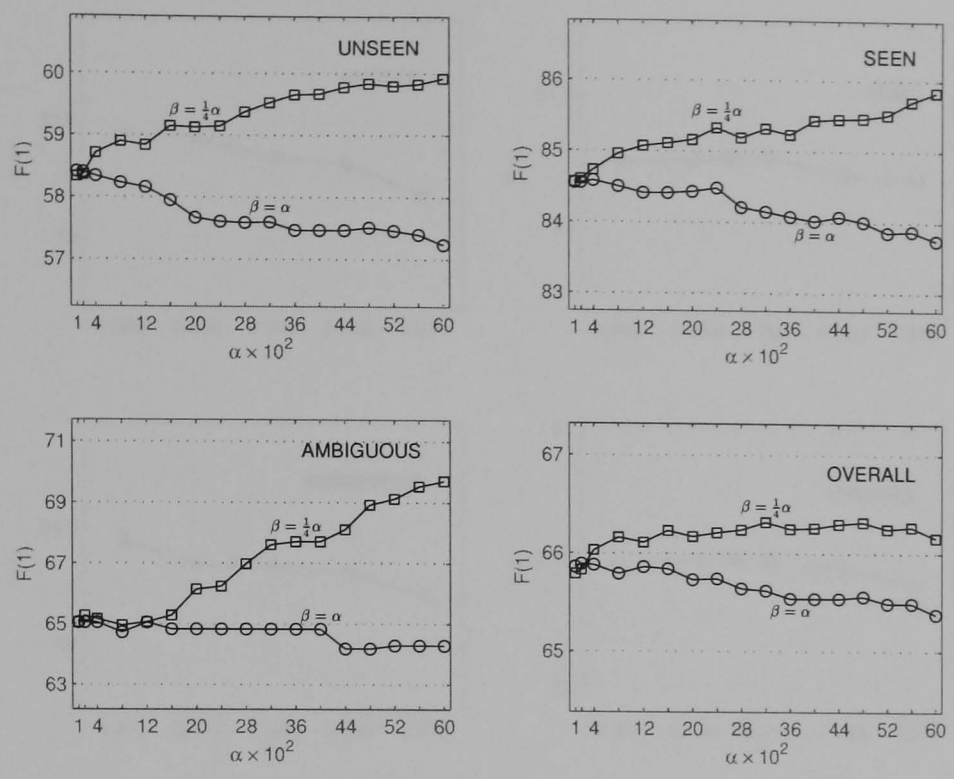


Figure 5.8: Comparison of the performances of biLexMENE for different smoothing functions: data is represented in FMLU notation, context windows are both set to [2,2], 200 GIS iterations are performed, cutoff is set to 1 and the number of selected cases is set to 50. Smoothing functions use value $\gamma = 0.001$, parameter α varies from 0.01 to 0.60 and two values are tested for parameter β: $\beta = \alpha$ and $\beta = \frac{1}{4}\alpha$. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

entity with a unique lexical feature— whereas the probability of the not-a-name class is concentrated over just one possible tag — namely the outside tag *O*. Therefore, the combined tag *O-not-a-name* can be considered equivalent to any of the other combined tags —such as *F-person*, *U-location*, etc.— which is expressed by making $\beta = \frac{1}{4}\alpha$. On the other hand, the tag *O-not-a-name* can be considered equivalent to the other classes, such as *person*, *location*, etc.— which is obtained by making $\beta = \alpha$.

At first glance, the results presented in figure 5.7 might suggest that the former option —i.e. making $\beta = \frac{1}{4}\alpha$— consistently yields better results. Indeed, this value for the β parameter helps biLexMENE to recognise much more seen, unseen and ambiguous named entities. However, on a closer look it seems that this value also drives the system to extract more spurious named entities, which makes the overall improvement much less significant.

Nevertheless, the combination $\alpha = 0.44$ and $\beta = \frac{1}{4}\alpha$ obtains the best performance for the approach and, consequently, these are the values selected for these parameters. These values are higher than expected, which suggests that the biased LexMENE models have a certain degree of overfitting —recall than the 200 GIS iterations value was set considering all training material— which the smoothing functions are to some extent correcting.

When this experiment is repeated on the MUC formal test corpus (figure 5.8), the

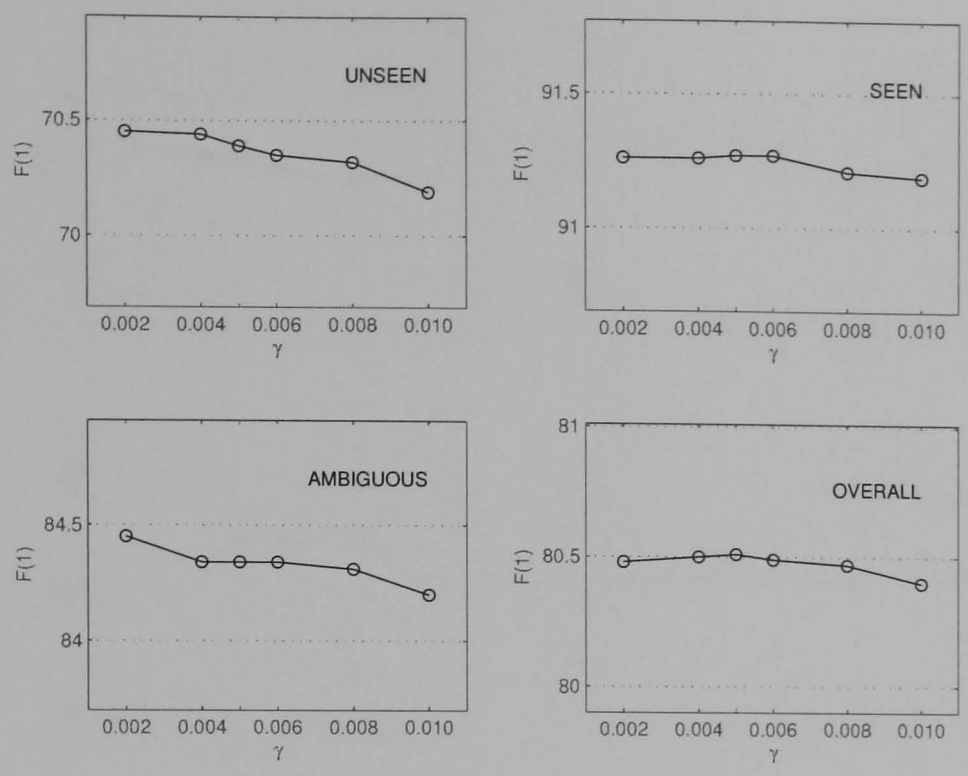


Figure 5.9: Comparison of the performances of biLexMENE for different smoothing functions: data is represented in FMLU notation, context windows are both set to [2,2], 200 GIS iterations are performed, cutoff is set to 1 and the number of selected cases is set to 50. Smoothing functions use $\alpha = 0.44$, $\beta = 0.11$, while parameter γ varies from 0.002 to 0.010. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

convenience of setting the parameter β to value $\frac{1}{4}\alpha$ is more evident. This figure also indicates that the values 0.32 or 0.48 for the α parameter would have been slightly better options. However, the improvement over the selected value 0.44 is marginal and this value remains a good alternative.

The second smoothing experiment is conducted to determine the influence of the γ parameter as well as a good value for it. It should be noticed that the values of α and β fixed above are used in this experiment. The results can be seen in figure 5.9.

These results indicate that, in general, the higher the value of γ the lower the performance of biLexMENE on unseen, seen and ambiguous named entities. However, this negative effect is not significant for the values plotted in figure 5.9, and a slight improvement in the recognition of hard named entities makes the overall behaviour almost constant. Nevertheless, it might be noticed than for the value $\gamma = 0.010$ a more considerable decrease in performance seems to be occurring. Further experiments confirm this trend and higher values for this parameter indeed have a negative effect on the approach.

Therefore, it can be concluded that the maximum entropy models used for adaptation are capturing fairly well the characteristics that indicate when a lexical feature is starting, continuing, ending or constituting a named entity. The performance of the system peaks at value 0.005, and consequently, this is the value selected for this parameter.

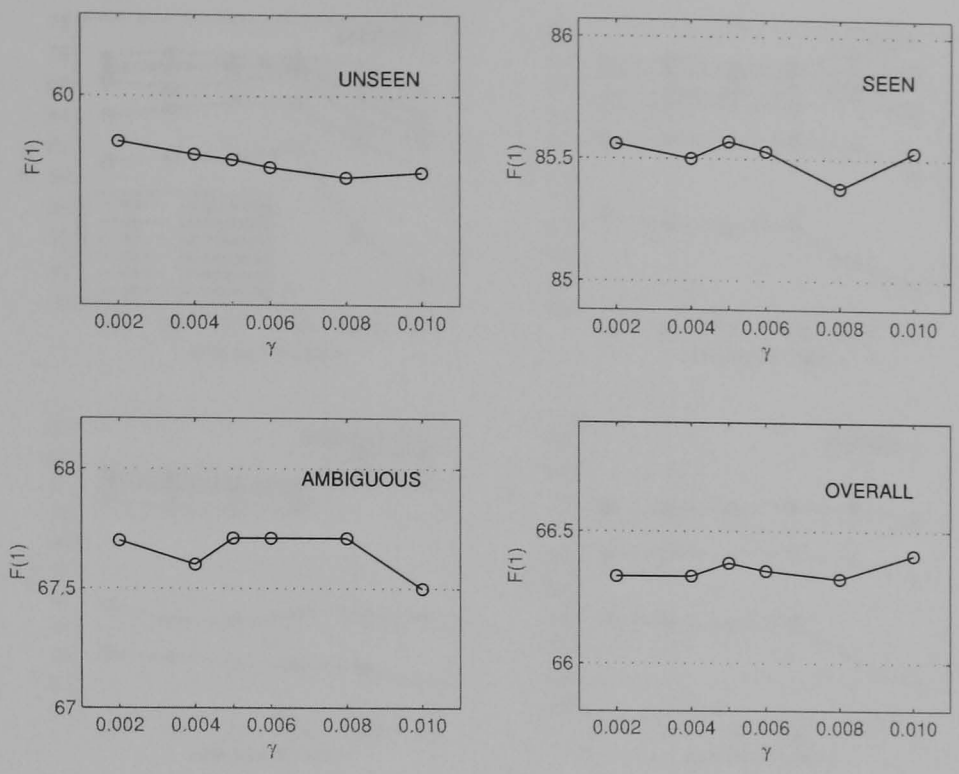


Figure 5.10: Comparison of the performances of biLexMENE for different smoothing functions: data is represented in FMLU notation, context windows are both set to [2,2], 200 GIS iterations are performed, cutoff is set to 1 and the number of selected cases is set to 50. Smoothing functions use $\alpha = 0.44$, $\beta = 0.11$, while parameter γ varies from 0.002 to 0.010. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

As earlier, the experiment has been repeated on the MUC formal test corpus. Results can be seen in figure 5.10. The effect of the γ parameter observed for the MUC dryrun test corpus is essentially the same effect shown in this figure. However, the curves seem to indicate that the performance of the system might be starting an increasing trend from the value 0.010. Further experiments discard this possibility and the improvement observed rapidly declines as higher values are tested for γ.

Therefore, the value 0.005 selected from the previous experiment is a good value for preparing biLexMENE to process unseen texts.

5.10.2 Assessing the hypothesis

Recall that the main hypothesis guiding the study of biLexMENE is that because of the bias introduced in the approach, less training material is needed to solve the task with a given level of accuracy and better classification can be obtained on infrequent or unseen named entities.

To assess this hypothesis empirically, an experiment has been set up in which the size of the training corpus is varied, so that the effect of the amount of the training material

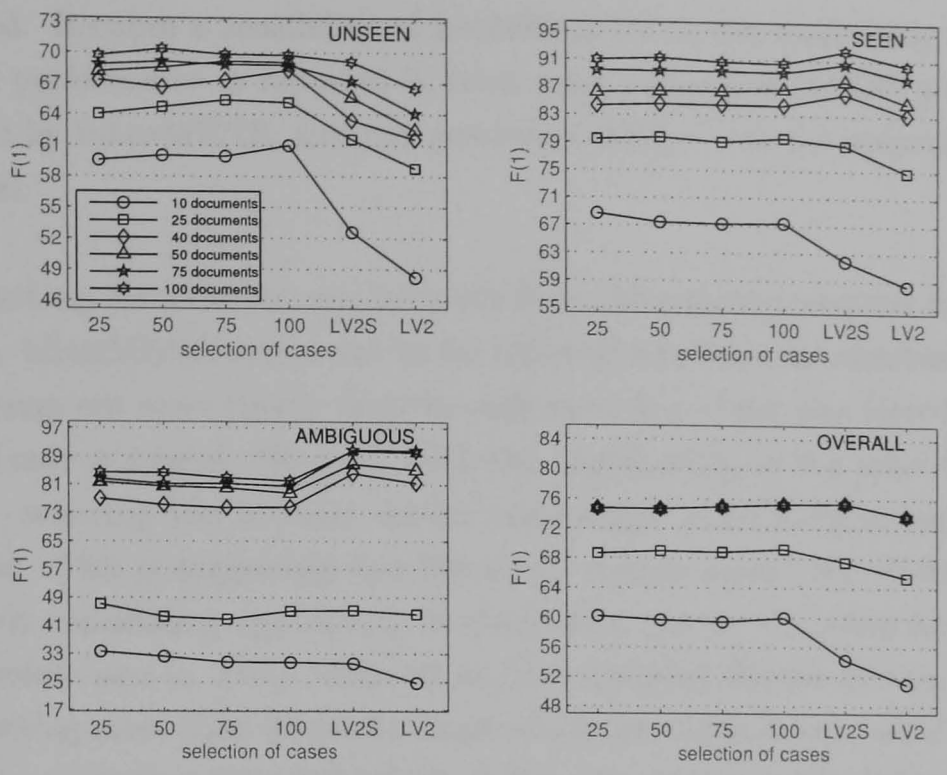


Figure 5.11: Comparison of the performances of biLexMENE for different sizes of the training corpus: data is represented in FMLU notation, context windows are both set to [2,2], 200 GIS iterations are performed, cutoff is set to 1 and the smoothing function uses $\alpha = 0.44$, $\beta = 0.11$ and $\gamma = 0.005$. The number of selected cases varies from 50 to 100. The performance of LexMENE-V2 is also included in the comparison, with and without smoothing. Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

available could be observed, as well as the behaviour of the system on different numbers of unseen named entities.

Initially, the sizes 25, 50, 75 and 100 documents were tested with the parameters set in the previous section. This experiment indicated that biLexMENE could obtain almost the same performance it shows when using the whole MUC training corpus with half of the training material, and that only three points of F-score are lost if the training corpus is reduced to just a quarter of its size. This motivated an extension of the experiment in to ways:

1. firstly, two new sizes for the training corpus were tested, namely 10 and 40 documents, to investigate whether the slow decrease in performance is maintained, and

2. the number of cases selected form each ranking was allowed to vary from 25 to 100 to determine whether this parameter has some incidence on these results

Table 5.1 on page 152 presents the outcome of this extended experiment in detail, whilst figure 5.11 shows a pictorial summary of it. In these results, the performance of LexMENE-V2 is also included, so that a comparison with the unbiased approach could

be established. Because a sensibility of LexMENE-V2 to the smoothing function was observed, its performance is reported in both ways, smoothed (LV2S) with the same function used by biLexMENE, and not smoothed (LV2) — as the approach would be normally used.

Several interesting observations can be made from the results presented in figure 5.11. For example, biLexMENE seems not to be affected much by the number of cases selected and brings out more clearly that the difference in performance introduced by this parameter is only marginal. However, with the introduction of the value 0.005 for the γ parameter, selecting the 75 most similar cases might yield a slight improvement in the extraction. This is suggesting that the set of similar cases been retrieved for each query, without considering repetitions, is maintained almost the same no matter how much its absolute size is. Only when 50 or more training documents are supplied for training, selecting more than 50 similar cases might introduce a few useful new cases in the set which could allow biLexMENE to slightly improve its performance.

The main observation however, is that biLexMENE is always able to obtain an overall performance that is higher than the one achieved by the unbiased approach, though the smoothing scheme makes this difference less significant. The same effect is observed when more training material is provided to the system: the improvement is quite important when only 10 training documents are available, less relevant when 25 documents are provided and no significant improvement is obtained when 40 or more training documents are considered. Therefore, the less the training material, the more important is to used the biased version to obtain a more accurate extraction.

This superiority of biLexMENE is more evident when only unseen[6] named entities are considered. No matter the size of the training corpus, biLexMENE is able to recognise much more unseen named entities than the unbiased version of the system, though this number —as stated above— is reduced as more training material is supplied. Moreover, biLexMENE also shows consistently better performance than LexMENE-V2 for ambiguous named entities when half or less of the training material is available. However, LexMENE-V2 obtains better results on the seen named entities, more significantly when smoothing is applied, which to some extent balances the overall performance of the smoothed approaches.

Table 5.2 on page 153 and figure 5.12 present the outcome of this experiment when repeated on the MUC formal test corpus.

[6]Clearly, when only a fraction of the training material is used the number of unseen named entities found in the test corpora rises. However, these results consider the original familiarity of the decoding named entities so that a better comparison with LexMENE-V2 and previous reports can be established. Nevertheless, the performance on this type of familiarity is even more relevant because many of the entities counted as seen or ambiguous were actually not included during training.

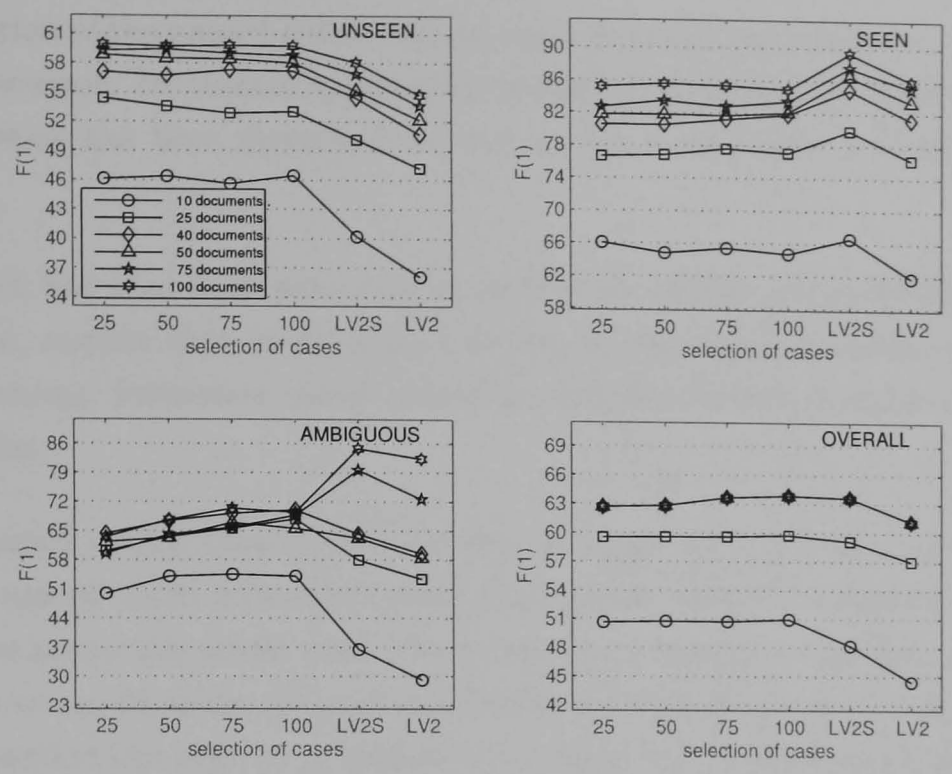


Figure 5.12: Comparison of the performances of biLexMENE for different sizes of the training corpus: data is represented in FMLU notation, context windows are both set to [2,2], 200 GIS iterations are performed, cutoff is set to 1 and the smoothing function uses $\alpha = 0.44$, $\beta = 0.11$ and $\gamma = 0.005$. The number of selected cases varies from 50 to 100. The performance of LexMENE is also included in the comparison, with and without smoothing. Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

These results closely follow the trends observed on the MUC dryrun test corpus, with the exception of the ambiguous named entities, for which LexMENE-V2 obtains better performance than the biased version when as few as 25 training documents are provided. In addition, the advantage of LexMENE-V2 on seen named entities is less steep than in the previous case.

Note that an experiment with a baseline system based on CRNs only, in the same fashion that the one presented in section 4.13, is not possible. In effect, a CRN is not a machine learning algorithm and they can just be used to store past cases and retrieve the most similar cases for a given query. As they do not attempt to perform any classification, an adaptation step is still needed and any comparison would be constrasting the two different adaptation approaches.

5.11 Summary and discussion

This chapter has presented an approach to biasing (a new version of) the maximum entropy baseline system towards pieces of texts that are similar to the text being analysed. This approach uses Case Retrieval Nets (CRNs), which is a successful framework for memory-based learning algorithm (Lenz and Burkhard 1996).

The formulation of the named entity task as past cases and new cases (i.e. queries) have been also discussed. Furthermore, the appropriate representation of these cases for the CRN framework has been described, as well as the storage and retrieval procedures utilised.

This approach has then been evaluated to survey the validity of hypothesis 2 proposed in this thesis, namely that introducing this bias to the baseline system based on the maximum entropy framework could help it to recognise better exceptions and unseen named entities.

Results obtained in this chapter indicate that it might be beneficial to use biased approaches to named entity extraction when the training material is limited. It would be interesting to assess this result with other methods of biasing a classifier. One possibility might be to use Boosting (Freund and Schapire 1999). However, it was claimed that the linear function this algorithm utilises to combine the classifications might waste the ability of some of these classifiers to perform better on certain named entities. This hypothesis deserves some exploration as well as the idea of creating a new approach in which the combination of the resulting classifiers should be based on the similarity between the query in process and the training examples that are correctly classified by them.

It was also determined in this chapter that the bias introduced in LexMENE-V2 does help to recognised infrequent named entities. However, the success obtained for unseen named entities is not equally repeated on ambiguous named entities. This suggests that the extraction of this type of named entity is quite difficult and that more world knowledge —perhaps in the form of word sense disambiguation— might be required.

The almost constant performance obtained by biLexMENE with different numbers of similar cases retrieved suggests that there is a reduced number of key examples for each query which might guide the extraction process. It could be worth looking for ways in which these key cases could be recognised and used in a method less expensive than the one described in this chapter. If this is achieved, further studies can be carried out to determine how these examples can be enriched to break the learning bound that biLexMENE seems to reach.

In this relation, the amount of computational resources that the biLexMENE requires to work is a remaining disadvantage. Several approaches could be studied to mitigate this limitation, but investigating other ways of detecting similarity between training and decoding texts would probably be the most relevant.

Another limitation which needs to be addressed is the manually coded similarity function for orthographic features. It might be possible to determine this similarity function

automatically on the basis of the features fired by lexical features in named entities of the same class, but further research is needed to test this possibility.

Finally, there is a large number of parameters to control the similarity between the information entities utilised in biLexMENE and their relevance for the cases defined. Although sensible decisions have been made to fix most of their values, an empirical study could contribute to locate better options. This study should be oriented to determine how to set these parameters so that the CRN approach retrieves the most relevant cases for each query. Of course, this relevance may not be easy to establish, but the activation of cases that contain named entities of the same class as the query, could be an initial useful approximation.

Table 5.1: Details of the performances of biLexMENE for different sizes of the training corpus: parameters are set as described in section 5.10.1, with the exception of the number of selected cases which varies from 50 to 100. The performance of LexMENE-V2 is also included in the comparison, with (LV2S) and without smoothing (LV2). Corpora: MUC-7 training corpus and MUC-7 dryrun test corpus.

	biLexMENE				LexMENE	
Selection:	**25**	**50**	**75**	**100**	**LV2S**	**LV2**
10 training documents						
UNSEEN	59.57	60.02	59.94	60.99	52.59	48.23
SEEN	68.67	67.41	67.22	67.38	61.90	58.31
HARD	30.40	26.02	26.45	28.10	21.36	12.77
AMBIGUOUS	33.61	32.23	30.62	30.54	30.24	24.71
OVERALL	60.18	59.67	59.42	59.96	54.31	51.03
25 training documents						
UNSEEN	59.57	60.02	59.94	60.99	52.59	48.23
SEEN	68.67	67.41	67.22	67.38	61.90	58.31
HARD	30.40	26.02	26.45	28.10	21.36	12.77
AMBIGUOUS	33.61	32.23	30.62	30.54	30.24	24.71
OVERALL	60.18	59.67	59.42	59.96	54.31	51.03
40 training documents						
UNSEEN	67.34	66.67	67.38	68.31	63.45	61.60
SEEN	84.33	84.58	84.50	84.47	86.15	83.19
HARD	36.23	35.97	37.41	35.29	10.39	10.96
AMBIGUOUS	77.00	75.23	74.47	74.57	84.22	81.52
OVERALL	74.73	74.66	74.96	75.28	75.39	73.71
50 training documents						
UNSEEN	68.18	68.58	69.20	69.09	65.66	62.47
SEEN	86.26	86.47	86.47	86.78	87.99	84.88
HARD	36.50	36.88	34.29	37.68	19.47	15.09
AMBIGUOUS	81.68	80.47	80.04	78.56	86.83	85.17
OVERALL	76.63	76.90	77.11	77.23	77.79	75.75
75 training documents						
UNSEEN	68.84	69.16	68.88	68.82	67.22	64.08
SEEN	89.42	89.36	89.37	89.25	90.54	88.37
HARD	30.66	32.35	31.88	33.58	18.64	18.18
AMBIGUOUS	82.64	81.21	81.29	80.51	90.79	90.63
OVERALL	78.73	78.81	78.81	78.71	80.27	78.98
100 training documents						
UNSEEN	69.74	70.39	69.87	69.78	69.12	66.54
SEEN	91.00	91.27	90.70	90.57	92.43	90.31
HARD	31.11	27.27	32.12	31.11	24.39	21.24
AMBIGUOUS	84.36	84.34	82.90	82.06	90.50	90.08
OVERALL	80.08	80.51	79.95	79.87	81.86	80.79

Table 5.2: Details of the performances of biLexMENE for different sizes of the training corpus: parameters are set as described in section 5.10.1, with the exception of the number of selected cases which varies from 50 to 100. The performance of LexMENE-V2 is also included in the comparison, with (LV2S) and without smoothing (LV2). Corpora: MUC-7 training corpus and MUC-7 formal test corpus.

	biLexMENE				LexMENE	
Selection:	**25**	**50**	**75**	**100**	**LV2S**	**LV2**
10 training documents						
UNSEEN	46.10	46.37	45.62	46.44	40.21	36.08
SEEN	66.06	64.79	65.39	64.73	66.62	61.80
HARD	23.53	23.26	23.81	21.18	18.18	18.42
AMBIGUOUS	49.92	54.07	54.55	54.10	036.48	29.14
OVERALL	50.65	50.75	50.72	50.95	48.21	44.43
25 training documents						
UNSEEN	54.38	53.57	52.87	53.02	50.15	47.21
SEEN	76.65	76.84	77.63	77.12	79.89	76.32
HARD	22.73	25.00	25.29	25.58	19.51	17.72
AMBIGUOUS	60.13	63.58	65.59	67.85	58.03	53.44
OVERALL	59.53	59.56	59.59	59.74	59.13	57.03
40 training documents						
UNSEEN	57.08	56.75	57.30	57.21	54.42	50.70
SEEN	80.53	80.52	81.21	81.82	84.89	81.25
HARD	22.73	22.99	25.29	27.59	17.28	17.95
AMBIGUOUS	64.36	67.40	69.19	70.05	64.24	59.56
OVERALL	62.70	62.82	63.66	63.86	63.65	61.16
50 training documents						
UNSEEN	58.83	58.51	58.45	58.04	55.16	52.17
SEEN	81.78	81.78	81.68	82.08	86.52	83.39
HARD	27.27	22.73	22.99	25.29	19.75	17.95
AMBIGUOUS	62.33	63.41	66.98	65.62	63.27	58.50
OVERALL	64.06	64.06	64.10	64.18	64.63	62.71
75 training documents						
UNSEEN	59.28	59.44	59.04	58.82	56.89	53.63
SEEN	82.69	83.46	82.81	83.46	87.64	85.08
HARD	25.00	27.27	24.72	25.00	24.39	17.72
AMBIGUOUS	59.53	64.16	65.83	68.76	79.63	72.67
OVERALL	64.47	65.23	65.09	65.25	66.88	64.87
100 training documents						
UNSEEN	59.91	59.75	59.83	59.79	58.06	54.74
SEEN	85.18	85.57	85.34	84.95	89.38	85.98
HARD	25.00	24.72	26.97	25.00	19.51	17.72
AMBIGUOUS	63.52	67.71	70.39	69.48	84.76	82.32
OVERALL	65.92	66.38	66.72	66.64	68.44	66.41

Chapter 6

Bootstrapping LexMENE

As was stated in chapter 2, preparing training material for an NEE task —and for NLP problems in general— can be expensive in terms of both time and expert resources. On the other hand, there is an unmeasurable abundance of (unlabelled) text in natural language. This disparity has made *bootstrapping*, or *semi-supervised learning*, a research topic of great interest to computational linguistics (Abney 2004). In this sense, here the term bootstrapping will refer to the techniques that aim to improve a classifier obtained from a small set of labelled training examples, by utilising a larger set of unlabelled examples.

In section 2.3, it was hypothesised that a NEE system based on the maximum entropy framework might take the advantages of semi-supervised learning, namely hypothesis 3, though some doubts were raised. This chapter will show that these concerns were justified and look into ways of overcoming these difficulties.

6.1 Why bootstrapping could or could not work for LexMENE?

On the one hand, named entity extraction is a task particularly compatible with bootstrapping concepts due to the existence of internal and external evidence for recognising names (McDonald 1996), as discussed in chapter 2.

Consider the following example, adapted from Riloff and Jones (1999): suppose that in a small set of annotated examples, the word *Spain* is seen several times annotated as a location name. This fact should be captured by the classifier which will both predict a high probability of finding a location name every time the word *Spain* is seen in an unlabelled sentence, and annotate them accordingly. Suppose also that among

these newly labelled sentences, there are several instances which contain the pattern *the president of Spain...* Then, by considering these examples as the training material for a new classifier, other location names can be captured from sentences like *the president of Venezuela*, *the president of France*, etc. These new location names could subsequently help to identify other contextual patterns, which in turn might assist in finding new location names. Riloff and Jones (1999) called this process *mutual bootstrapping.*

Although this mutual bootstrapping works well in principle, its performance can rapidly deteriorate with patterns which have affinity for more than one type of name (Riloff and Jones 1999, Pierce and Cardie 2001). For example, the context *the president of* may indicate the presence of a country name, as exemplified above, but it can also be found around names, or generic references, of organisations such as in *the president of Manchester United FC*, *the president of the committee*, etc.

Therefore, this kind of approach requires a filter for the bootstrapped instances of one iteration that will be considered in the training of the classifier on the next iteration. Riloff and Jones (1999) used an approach based on rules and make the system more robust by adding a second level of bootstrapping in which the five most reliable noun phrases produced after a number of iterations, based on the number of different rules that extract them, are selected and permanently added to the training material. Most researchers, however, utilised a more direct ranking approach (Blum and Mitchell 1997, Steedman, Sarkar, Osborne, Hwa, Clark, Hockenmaier, Ruhlen, Baker and Crim 2003, Ng and Cardie 2003).

Nonetheless, the main risk of bootstrapping LexMENE comes from the maximum entropy model it uses as the learning paradigm. It is not clear that this type of approach is suitable for bootstrapping methods since the maximum entropy framework obliges the learner not to make any assumption other than the constraints imposed by the training data. Therefore, it might be the case that the annotations made by a maximum entropy classifier would only reflect these constraints, and considering subsequently these annotations for training another classifier could make little or no contribution to the learning process.

However, experiments in previous chapters have shown that different variants of LexMENE are able to identify unseen named entities. The work in this chapter aims to assess whether this ability of LexMENE is enough to support a bootstrapping procedure.

There is another potential problem with bootstrapping LexMENE. Ng and Cardie (2003) identified and explained it quite well: there are feature-value pairs which alone can recognise and determine the class of a named entity. Probabilistic methods, unlike rule-based approaches, cannot take advantage of these pairs directly because they make

Algorithm 6.1: The original Co-training algorithm. Adapted from Blum and Mitchell (1997).

Input: a set L of labelled examples and a set U of unlabelled examples

procedure CO-TRAINING(L, U)
  Create a pool U' of examples by choosing u examples at random from U
  repeat
    Use L to train a classifier h_1 that considers only the x_1 portion of x
    Use L to train a classifier h_2 that considers only the x_2 portion of x
    Allow h_1 to label p positive and n negative examples from U'
    Allow h_2 to label p positive and n negative examples from U'
    Add these self-labelled examples to L
    Randomly choose $2p + 2n$ examples from U to replenish U'
  until k iterations are completed
end procedure

decisions based on a combination of features. Nevertheless, they also recognise that these methods had the advantage of being resistant to class skewness, a characteristic easily found on the data of many NLP problems —and NEE is not the exception— in which the number of negative examples utterly outnumber the number of positive ones.

When this research started in 2000-2001, there were no reports of attempts at bootstrapping maximum entropy models, perhaps for the reasons mentioned above, but two works which consider this learning approach have been recently published (Clark et al. 2003, Cui and Guthrie 2004). This research will be discussed later on this chapter.

6.2 Main bootstrapping approaches

Most of the previous work on bootstrapping in NLP problems are based on two techniques: *Yarowsky algorithm* (Yarowsky 1995) and *co-training* (Blum and Mitchell 1997).

Co-training has somehow become more popular, perhaps because there have been some theoretical work on the approach (Dasgupta, Littman and McAllester 2002, Abney 2002), and several variants are reported in the literature. Essentially, co-training exploits the redundancy in natural language texts by considering two *independent views* of the data. The method can be applied on an instance space $X = X_1 \times X_2$, where X_1 and X_2 are the two different views of an example which are not tightly correlated, and each view in itself is sufficient to correctly classify any instance x. Therefore, if f denotes the target function for any example $x = (x_1, x_2)$ associated with label l, there must exist the functions f_1 and f_2 so that $f(x) = f_1(x_1) = f_2(x_2) = l$. By using the small set of labelled examples, two weak predictors for f_1 and f_2 can be found, which then can be used to bootstrap the unlabelled examples following the procedure shown in algorithm 6.1 (Blum and Mitchell 1997).

Blum and Mitchell (1997) provided an analysis of why the co-training algorithm works

that is based on the independence between the different views of the data and the maximisation of the number of labels in which the classifiers agree. However, Abney (2002) argues that this analysis is flawed, because the co-training algorithm does not directly seek the agreement between the classifiers and because the independence assumption is very strong and normally violated by the data.

Nonetheless, the co-training method has proved to work in practice and successful applications have been obtained in NLP problems that have a natural view separation, such as document classification, studied in the original work (Blum and Mitchell 1997), named entity recognition (Collins and Singer 1999) and NP chunking (Pierce and Cardie 2001).

However, the sensitivity of the Blum and Mitchell's (1997) algorithm to the assumptions of independence and self-sufficiency of the two views has been shown to be an important limitation on NLP problems that do not have this clear separation. Muslea, Minton and Knoblock (2002) has shown that co-training is not very effective in classifying documents when the data does not allow uncorrelated views; Müller, Rapp and Strube (2002) found mostly negative results when applying co-training to coreference resolution, a result confirmed later by Ng and Cardie (2003).

These limitations have spurred researchers to investigate alternative co-training methods which do not require these views. The resulting variations have converged on replacing the two different views in a learner with two different *learning algorithms*. Goldman and Zhou (2000) studied the interaction of a decision list learner and a decision graph learner on several benchmark problems; Sarkar (2001) utilises two probabilistic models that form parts of a statistical parser to co-train each other; Steedman et al. (2003) also worked on parsing, but they used two completely different parsers; similarly, Clark et al. (2003) co-train two well-known probabilistic part-of-speech taggers; Ng and Cardie (2003) mixed decision lists and a Naïve Bayes classifier for coreference resolution. All these studies report successful results, some of them remarkably good. Hereafter, this variant approach of co-training will be referred to as *co-learning*.

Unfortunately, only Clark et al. (2003) studied co-learning using maximum entropy models, as they applied the technique to the TnT tagger based on a trigram HMM (Brants 2000), and the C&C tagger based on the maximum entropy framework (Curran and Clark 2003a). The approach is described in algorithm 6.2. Initially, the taggers are trained on a small set of *seed* labelled sentences. At each iteration, a fixed-size *cache* is selected from the set of unlabelled sentences. Each tagger is then applied independently to this set. The resulting sentences labelled by the TnT tagger is added to the training

Algorithm 6.2: The Naïve Co-learning algorithm for the TnT and C&C taggers. Adapted from Clark et al. (2003).

Input: a set S of labelled sentences and a set U of unlabelled sentences

procedure NAÏVE-CO-LEARNING-TNT-C&C(S, U)
  $L_{TnT} \leftarrow S$
  Train the TnT tagger on L_{TnT}
  $L_{C\&C} \leftarrow S$
  Train the C&C tagger on $L_{C\&C}$
  repeat
    Partition U into the disjoint sets C and U'
    $C_{TnT} \leftarrow$ application of TnT on C
    $C_{C\&C} \leftarrow$ application of C&C on C
    $L_{TnT} \leftarrow L_{TnT} \cup C_{C\&C}$
    Train the TnT tagger on L_{TnT}
    $L_{C\&C} \leftarrow L_{C\&C} \cup C_{TnT}$
    Train the C&C tagger on $L_{C\&C}$
    $U \leftarrow U'$
  until U is empty
end procedure

data for the C&C tagger, and vice versa[1]. At the end of the iteration, the cache is cleared and those sentences are removed from the total pool of unlabelled sentences. The algorithm stops when all unlabelled sentences have been used.

There are important findings in this work that should be considered here. Firstly, the co-learning approach was very successful and, with just 50 seed sentences and a cache of 500 sentences, is able to boost the performance of the C&C tagger from F-score 73.2% to F-score 85.1 after 50 iterations. Moreover, the experiments showed that the performance of both taggers gets better as the cache size increases. Secondly, the improvement is less impressive if the number of the labelled sentences (seeds) is incremented, making the co-learning approach ineffective when the taggers are initially trained with a large amount of manually annotated training examples.

However, the most relevant observation comes from a set of experiments which Clark et al. (2003) called *self-training*. They modified algorithm 6.2 so that each tagger is retrained on its own labelled cache at each iteration. They reported that the performance of the C&C remains constant under these settings, no matter the number of seed sentences provided. This finding could be indicating that the maximum entropy framework, discussed in section 6.1, is not suitable for bootstrapping.

One explanation for this disappointing result is given in Abney (2004), who in an attempt at understanding the Yarowsky algorithm from a theoretical point of view, proposed several bootstrapping algorithms which, under certain conditions, have been

[1]This is the naïve version of the algorithm. Clark et al. (2003) also studied an agreement-based co-training in which only a subset of the cache is selected to be added to the training material for the next iteration.

proven to optimise the negative log likelihood of a classifier —at least to a local minimum— with respect to the target labelling function.

To understand Abney's (2004) work, the following notation is needed. A bootstrapping approach considers a set of examples $X = \Lambda \cup V$, where Λ represents the portion of examples that are labelled and V the portion of unlabelled examples. Whilst X remains the same during the bootstrapping procedure, its components vary in time. Therefore, at a given iteration t, X is composed by the current sets of labelled and unlabelled examples $\Lambda^{(t)}$ and $V^{(t)}$ respectively. In this way, an example x is associated to a label $Y_x^{(t)}$ at iteration t, which takes a value $y = 1, \ldots, L$ —i.e. there are L possible classes— for labelled examples and the undefined value $\perp$ for unlabelled examples. Considering these associations, a *labelling distribution* $\phi_x^{(t)}(y)$ is defined for an example $x \in X$ and label y that takes the following possible values

$$\phi_x^{(t)}(y) = \begin{cases} 1 & \text{if } x \in \Lambda^{(t)} \text{ and } y = Y_x^{(t)} \\ 0 & \text{if } x \in \Lambda^{(t)} \text{ and } y \neq Y_x^{(t)} \\ \frac{1}{L} & \text{if } x \in V^{(t)} \end{cases} \tag{6.1}$$

Finally, in each iteration t, the bootstrapping process uses a *base learning algorithm* to draw a classifier $C^{(t+1)}$ from the space of supervised classifiers that can be obtained from the training data $(\Lambda^{(t)}, Y^{(t)})$[2].

Abney (2004) proposes a bootstrapping algorithm, named Y-1, that is remarkably similar to Clark et al.'s (2003) self-training (CST) algorithm. Indeed, the differences are not very significant:

- ▷ while CST uses a cache of unlabelled examples, Y-1 considers all of them. This can be seen as using a large cache in CST approach
- ▷ Y-1 transfers to the next-iteration training material only those examples for which the base classifier has declared itself. But using the C&C tagger as base learner, it is unlikely that Y-1 would predict uniform probabilities on many examples. Thus, the final results of both approaches would be quite similar.
- ▷ the main difference is that CST clears its cache after each iteration, while Y-1 considers the same training data

[2]In this notation, when an index is not specified, the concept refers to all the examples in the set. Thus, $Y^{(t)}$ denotes the set of labels for all examples in X. Similarly, $\phi^{(t)}$ will denote the labelling distribution as a function over all examples and labels.

Algorithm 6.3: Abney's (2004) Y-1 bootstrapping algorithm.

Input: a set X of examples and their initial labels $Y^{(0)}$

procedure MODIFIED-GENERIC-YAROWSKY-BOOTSTRAPPING($X, Y^{(0)}$)
  for $t \in \{0, 1, \ldots\}$ **do**
    Train a classifier $C^{(t+1)}$ on $(\Lambda^{(t)}, Y^{(t)})$
    for each example $x \in X$ **do**
      $\hat{y} \leftarrow \operatorname{argmax}_y C_x^{(t+1)}(y)$
      if $x \in \Lambda^{(0)}$ **then**
        $Y_x^{(t+1)} \leftarrow Y_x^{(0)}$
      else if $x \in \Lambda^{(t)}$ **or** $C_x^{(t+1)}(\hat{y}) > \frac{1}{L}$ **then**
        $Y_x^{(t+1)} \leftarrow \hat{y}$
      else
        $Y_x^{(t+1)} \leftarrow \perp$
      end if
      if $Y^{(t+1)} = Y^{(t)}$ **then**
        stop
      end if
    end for
  end for
end procedure

Algorithm 6.3 sketches Y-1's procedure. Abney (2004) proved that Y-1 maximises the likelihood of the probability of the full data set according to the classifier's model when the base learning algorithm satisfies inequality 6.2, with equality only if there is no classifier $C_x^{(t+1)}$ that makes $\Delta D_\Lambda < 0$.

$$\Delta D_\Lambda \equiv \sum_{x \in \Lambda^{(t)}} \sum_{y \in \mathcal{Y}} \phi_x^{(t)}(y) \log \frac{C_x^{(t)}(y)}{C_x^{(t+1)}(y)} \leq 0 \tag{6.2}$$

In other words, the base learner must reduce the divergence, measured in term of the Kullback-Leibler distance (Kullback and Leibler 1951, Cover and Thomas 1991), with the labelling function at each iteration.

On the other hand, recall from section 2.4 that the maximum entropy framework provides a model which coincides with the one that maximises log-likelihood ($\mathcal{L}$) with the training examples. Therefore, if $C^{(t+1)}$ is a maximum entropy model trained over the examples $x \in \Lambda^{(t)}$, its log-likelihood with respect to this set of examples will be higher than any other model drawn from the same space, such as $C^{(t)}$. Therefore, the following analysis can be written, where $\tilde{p}_x^{(t)}(y)$ corresponds to the empirical distribution at time t. Note the similarity of equations 6.2 and 6.3.

$$\begin{aligned} \mathcal{L}(C^{(t)}) - \mathcal{L}(C^{(t+1)}) &\leq 0 \\ \sum_{x \in \Lambda_t} \sum_{y \in \mathcal{Y}} \tilde{p}_x^{(t)}(y) \log C_x^{(t)}(y) - \sum_{x \in \Lambda_t} \sum_{y \in \mathcal{Y}} \tilde{p}_x^{(t)}(y) \log C_x^{(t+1)}(y) &\leq 0 \\ \sum_{x \in \Lambda_t} \sum_{y \in \mathcal{Y}} \tilde{p}_x^{(t)}(y) \log \frac{C_x^{(t)}(y)}{C_x^{(t+1)}(y)} &\leq 0 \end{aligned} \tag{6.3}$$

This makes evident that maximum entropy models try to reduce the divergence with respect to the empirical distribution $\tilde{p}_x^{(t)}(y)$ which, unfortunately, is normally different from the labelling distribution $\phi_x^{(t)}(y)$ in unrestricted sets of training examples.

Therefore, Y-1 is not guaranteed to converge when used with a maximum entropy model as base learner and, given the close correspondence between this algorithm and the CST algorithm, the unsatisfactory results reported in Clark et al. (2003) should not be surprising. In fact, preliminary experiments with the combination Y-1+LexMENE showed an oscillatory curve of performance from which it seems unable to escape.

The Y-1 algorithm is very close to the original Yarowsky algorithm, or the *Y-0 algorithm* in Abney's (2004) terms. There are only two differences:

- ▷ Y-0 allows non-hand annotated examples to become unlabelled again, whilst in Y-1 an example once labelled remains labelled, though the label may change
- ▷ at the end of each iteration, Y-1 labels all examples for which the basic learner predicts a non-uniform distribution among the classes, while Y-0 labels only examples for which the best probability exceeds a given threshold. In this respect, Y-1 can be seen as Y-0 with a threshold fixed to $\frac{1}{L}$ where L is the number of classes.

Thus, it might be possible that by selecting for the next iteration's training material only examples for which there is a high confidence on their predicted classifications, the algorithm could be able to learn new patterns without falling into the cycles observed in the preliminary experiments. Therefore, the initial experiments in this chapter aim to test whether the Y-0 bootstrapping approach can help LexMENE to obtain valuable information from unlabelled examples.

6.3 Experiment settings

In order to test Yarowsky approach applied to LexMENE, hereafter boLexMENE, it is essential that a confidence measure is provided with each prediction. It is relatively simple to obtain such a measure for each token from the distribution over the classes predicted by LexMENE's maximum entropy model. However, the class of a token is not (exclusively) decided based on these probabilities. LexMENE utilises the FMLU notation, which was found to produce better results than the BIO notation in chapter 4. As a result, these distributions cannot be used directly and the algorithm requires the application of a Viterbi search to annotate complete sentences with a valid sequence of labels. Thus, a confidence measure for the predicted sequence of labels is needed here.

Algorithm 6.4: The Viterbi Forward Backward Search algorithm (presented with Rabiner's (1989) notation). Adapted from Brushe, Mahony and Moore (1996).

Input: a state transition distribution a that represents the valid transitions among classes and that includes the special states *start-of-sentence* (SOS) and *end-of-sentence* (EOS); class distributions $\{b(i)\}_T$ given by LexMENE's maximum entropy model for each token O_t, $1 \leq t \leq T$, over every NE class i

Output: a normalised *a posteriori* distribution γ over the classes and the most likely sequence of classes q^* according to this distribution

1: **procedure** VFBS$(a, \{b(i)\}_T)$
2: **Initialisation:**
3: $\delta_1(i) = a_{SOS \to i} b_1(i) \quad 1 \leq i \leq N$
4: $\beta_T(i) = a_{i \to EOS} \quad 1 \leq i \leq N$
5: **Recursion:**
6: $\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{i \to j}] b_t(j) \quad 1 \leq j \leq N, \; 2 \leq t \leq T$
7: $\beta_t(i) = \max_{1 \leq j \leq N} [\beta_{t+1}(j) a_{i \to j} b_{t+1}(j)] \quad 1 \leq i \leq N, \; T-1 \geq t \geq 1$
8: **Termination:**
9: $\gamma_t(i) = \delta_t(i) \beta_t(i) \quad 1 \leq j \leq N, \; 1 \leq t \leq T$
10: Normalise γ
11: **Best sequence:**
12: $q_t^* = \operatorname{argmax}_{1 \leq i \leq N} [\gamma_t(i)] \quad 1 \leq t \leq T$
13: **end procedure**

Clearly, the product of the best probability of each token distribution is not an appropriate measure as longer sentences will be unfairly penalised with respect to the shorter ones. A normalisation step could be attempted but this is not directly applicable and there are novel, better ways of obtaining a confidence measure for the sequence of labels in which its length has little or no impact.

Several of these approaches originated from spoken language processing (Forney 1972, Hagenauer and Hoeher 1989, Junkawitsch, Neubauer, Höge and Ruske 1996, Morguet and Lang 1998, Li, Malkin and Bilmes 2004); others have been recently proposed for handwritten text recognition (Schlapbach and Bunke 2004) and information extraction (Culotta and McCallum 2004). Most of these methods can be adapted to work for boLexMENE with minor modifications.

However, boLexMENE utilises the Viterbi Forward Backward Search (VFBS) given in algorithm 6.4, which is an adaptation of the algorithm proposed by Brushe et al. (1996) and further developed in Brushe, Mahony and Moore (1998). VFBS computes an *a posteriori* probability measure for each class at each token which is maximised over all valid paths that consider that (token, class) pair. Although VFBS is computationally more expensive than a normal Viterbi search, it finds the same maximum likelihood sequence and also provides a distribution over the classes which can then be normalised in the standard way.

Now it is easy to obtain a confidence measure for the sequence of named entity classes assigned to a sentence: boLexMENE considers the Kullback-Leibler divergence (Kullback and Leibler 1951, Cover and Thomas 1991) of the distribution γ returned by the VFBS

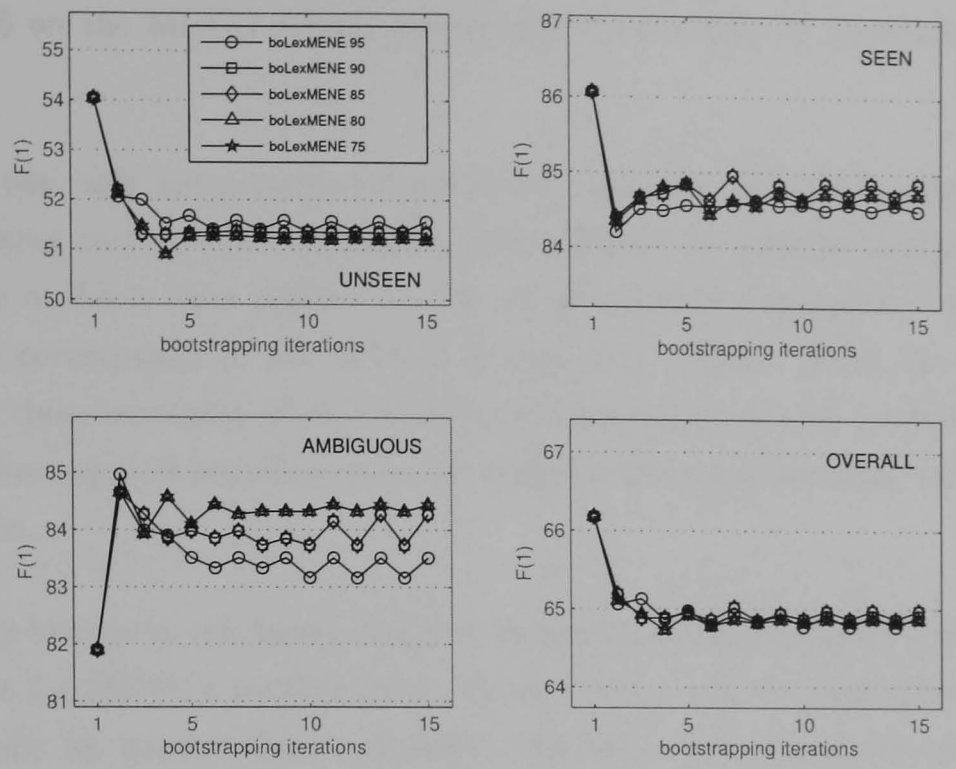


Figure 6.1: Comparison of the performances of boLexMENE with different thresholds: features as used by LexMENE-V2, cutoff is set to 3 and GIS iterations to 200. Bootstrapping: Y-0 with thresholds varying from 75% to 95%. Labelled examples: MUC-7 training corpus; unlabelled examples: MUC-7 dryrun test corpus; test examples: MUC-7 formal test corpus.

and the appropriate uniform distribution for that sentence. This measure also makes the definition of thresholds simple. For example, suppose that a probability distribution is defined —with the same number of outcomes as considered by boLexMENE— so that 90% of its mass is concentrated in one class and the rest shared out among the other classes. By telling Y-0 to use as threshold the distance between this distribution and the uniform distribution, only sentences labelled with at least 90% confidence will be selected.

6.4 Experiments

As in the previous chapter, in order to avoid a number of experiments for setting the parameters of boLexMENE, those fixed for either LexMENE or MOLI MENE will be used, as there are no reasons to believe that maintaining these values would unfairly affect their comparison. Thus, all experiments presented in this section utilise the FMLU notation, a cutoff set to three, 200 GIS iterations, a lexical window of sizes [2,2], a orthographic window of sizes [2,2] from which all fired features are considered, and the unknown words threshold set to three.

The first experiment aims to test the first approach to boLexMENE, which consists of the Y-0 algorithm with LexMENE-V2 as base learner, with different threshold values: 75%, 80%, 85%, 90% and 95% confidence. Figure 6.1 presents the performance of

boLexMENE on the MUC-7 formal test corpus for the first 15 iterations of the Y-0 algorithm.

At iteration one, only hand-annotated examples —which in this case corresponds to the MUC-7 training corpus— are utilised by boLexMENE to train its maximum entropy model. This model is then applied on the set of unlabelled examples, which in these experiments corresponds to the MUC-7 dryrun test corpus. Then the threshold is applied and those sentences whose labels have been assigned with a confidence higher than the value required are selected to be added to the set of training examples in the next iteration.

It can clearly be seen in this figure that the bootstrapping approach does not contribute but damages LexMENE's performance. There is some gain for ambiguous named entities, specially for modest threshold values, but this is not enough to compensate for the loss on the other familiarities and the overall performance declines. Moreover, no matter the threshold, boLexMENE also falls into oscillatory curves of performance — as happened in preliminary experiments with Y-1— which means that Y-0 does not converge as its stop condition is never satisfied.

There are two possible scenarios which might explain these results. The first one is that maximum entropy models have intrinsic difficulties to learn from themselves and thus are not appropriate for direct bootstrapping. To some extent, the experiments in Clark et al. (2003) suggest this as the performance of the maximum entropy tagger they use remains constant when retrained on its own output, but considerably improves in co-learning with another tagger.

The second option was also observed by Clark et al. (2003): their co-learning algorithm "was unable to improve the performance of the taggers when they had already been trained on large amounts of manually annotated data". Surprisingly, what Clark et al. (2003) call a large amount of manually annotated data seems to be just 500 sentences. The MUC-7 training corpus —used as the hand-labelled data here— contains 100 documents which sum up to 4,377 sentences.

The second scenario can be proved —or disproved— by repeating the experiment of above with a smaller set of manually annotated examples. Figure 6.2 shows the results of boLexMENE on the MUC-7 formal test corpus when using only the first 10 documents from the MUC-7 training corpus as hand-labelled examples. The other 90 documents and the 100 documents from the MUC-7 dryrun test corpus are provided for bootstrapping.

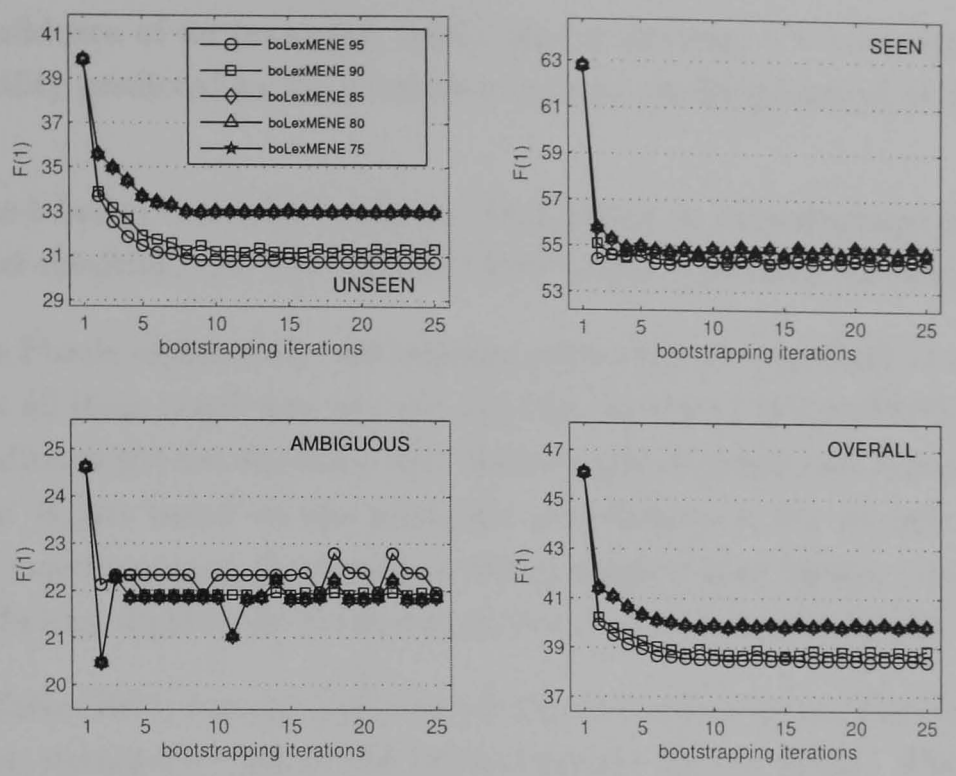


Figure 6.2: Comparison of the performances of boLexMENE with different thresholds: features as used by LexMENE-V2, cutoff is set to 3 and GIS iterations to 200. Bootstrapping: Y-0 with thresholds varying from 75% to 95%. Labelled examples: 10 documents from the MUC-7 training corpus; unlabelled examples: 90 documents from the MUC-7 training corpus + MUC-7 dryrun test corpus; test examples: MUC-7 formal test corpus.

It can be seen once again that boLexMENE would run indefinitely if not stopped after 25 iterations. Even worse, these results suggest that the amount of initial training material is not the main problem of the approach, but that there is some deeper issue that is responsible for the poor performance of the system.

These difficulties might be inherent to maximum entropy models, due to the maximum entropy principle, or even to probabilistic machine learning methods, as suggested in Ng and Cardie (2003), because they fail in capturing relevant clues that are nevertheless infrequent.

6.5 Lighting the way

In a recent work, Cui and Guthrie (2004) proposed a bootstrapping framework with a maximum entropy model as the learning component to perform semantic tagging, an NLP task that is similar to —but more general than— named entity extraction. They argue that if the following three conditions are satisfied:

1. feature frequencies are not smoothed
2. the set of features remains the same during the bootstrapping process

3. the confidence of an (example, label) pair at iteration $t+1$ corresponds to the probability predicted by the maximum entropy model generated at iteration t

then machine-labelled examples "make no contribution to the constraint set". Because this is counter-intuitive, Cui and Guthrie (2004) called this effect the *MaxEnt Puzzle*.

The MaxEnt Puzzle explains the self-training results reported in Clark et al. (2003), as it seems that all three conditions are met by their approach. BoLexMENE satisfies the first two conditions but not the third one. Nevertheless, it seems that using a confidence measure that is just based on the predicted probabilities is not enough to solve the puzzle. The experiments in the previous section suggest that feature expectations are changing, affecting slightly the performance, but that they remain largely unchanged.

Cui and Guthrie (2004) recommend to break the restrictions by re-selecting the feature set after each iteration or adjust the labels assigned by the model. They conducted experiments in which both recommendations are followed. First, they re-select features based on the Association Rule's principles of *support* and *confidence*. And secondly, they permanently correct some of the predictions produced by their model in two ways: adjusting the probabilities given by the model, such as setting to probability zero the least probable label, setting to probability one the most probable label and removing instances with flat distributions; and pruning illegal labels by using information from an external dictionary. Cui and Guthrie (2004) reported that the new approach reduced the error rate of the system, but the improvement was not impressive.

BoLexMENE is already correcting the maximum entropy's predictions by applying a Viterbi search. This effect was observed in chapter 4, in which the performance of the approach is considerably boosted by using the FMLU notation and the corresponding Viterbi algorithm. However, this corrective approach seems not to be enough to overcome the MaxEnt Puzzle, as observed in previous experiments. Adding external sources of correction, such as gazetteers, would necessarily compromise the portability of the system. Therefore, it seems necessary for bootstrapping LexMENE to introduce a re-selection of features procedure.

In chapter 5, the Ripper rule inductor (Cohen 1995) was found to be good at generating features for maximum entropy models. Thus, it seems a reasonable idea to use this learning algorithm to *inform* LexMENE of a set of potentially good features present in the training data. Then, LexMENE could *inform* Ripper of the classes of a set of examples for which they were previously unknown. Then, Ripper will inform back of a new set of good features to describe the new examples. This *co-informing* process may continue until no new rules are necessary to describe newly labelled examples or there are no more examples to bootstrap on.

Algorithm 6.5: LexMENE^{Ripper}: the co-informing approach applied to LexMENE-V2 and Ripper.

Input: a set L of labelled sentences, a set U of unlabelled sentences and a cache size n
Output: a LexMENE-V2 model trained on $L \cup U$

```
1: procedure CO-INFORMING-LEXMEME-V2(L, U, n)
2:     rules^(0) ← Ripper(L)
3:     t ← 1
4:     repeat
5:         Train LexMENE-V2^(t) with rules^(t−1) as features
6:         Apply LexMENE-V2^(t) on U
7:         S ← the n most confidently labelled sentences in U (selected with care)
8:         R ← Ripper(S)
9:         rules^(t) ← rules^(t−1) ∪ R
10:        if rules^(t) = rules^(t−1) then
11:            Stop
12:        end if
13:        U ← U − S
14:        t ← t + 1
15:    until U is empty
16: end procedure
```

6.6 LexMENE^{Ripper}

Following the discussion above, a new named entity extractor has been designed which is based on co-informing LexMENE-V2 and Ripper. This system has been named LexMENE^{Ripper}.

Algorithm 6.5 describes the co-informing approach applied to LexMENE-V2 and Ripper. Initially, the training sample —i.e. manually-annotated sentences— is handed to Ripper which obtains an initial set of rules from them. Then, an iterative procedure starts by training LexMENE-V2 on this set of rules and then applying it to all unlabelled sentences. The next step is the selection of a fixed number of newly labelled sentences from the pool of unlabelled sentences. This idea combines Riloff and Jones's (1999) and Clark et al.'s (2003) approaches. This corresponds to a cache of unlabelled sentences, which is cleared at the end of each iteration. However, the sentences selected to fill this cache are not selected randomly before labelling, but with the most reliably annotated sentences.

It is important to mention here that the approach always tries to obtain new information from the unlabelled examples. Consequently, the selection of the sentences for the cache is performed with the due care. For example, if only negative sentences are selected, Ripper would come out with an empty set of rules and the instruction of predicting the class `O` for every token. Another possible situation is that only sentences with information of the document, such as sentences describing the author or the date of the document, are selected. This kind of sentence normally shows little variability and Ripper could produced a set of rules which are already known. Both situations lead to a premature stop of the algorithm. Therefore, the selection of sentences for the cache

tries to maintain as much as possible the same number of sentences with each of these characteristics.

When a cache of newly labelled sentences has been selected, it is handed to Ripper with the mission of generating a new set of rules to classify them. These rules are then added to the current set of rules. If no new rules can be added, the algorithm assumes convergency and stops. Otherwise the process is repeated until all unlabelled sentences have been added into the cache.

It could be also possible to set a minimum confidence threshold for selecting newly labelled sentences, so that the bootstrapping process halts when there are no reliable sentences from which information can be drawn. Or alternatively, the size of the cache could be reduced after each iteration so that sentences labelled with low confidence are not considered in the feature production. These, and other more common, stopping criteria can be the subject of further research.

6.7 An experiment with LexMENE^{Ripper}

An experiment with LexMENE^{Ripper} is conducted here to determine whether a maximum entropy-based NEE system can benefit from semi-supervised learning.

As explained in section 6.6, the initial extractor corresponds to a maximum entropy model that uses as features the rules generated by Ripper from the training sentences, that is sentences that have been hand-annotated, which in this case corresponds to the MUC-7 training corpus.

After that, the bootstrapping procedure begins. The maximum entropy extractor is applied to unlabelled sentences, in this case the MUC-7 dryrun corpus, and the most reliable annotations made by the system are selected. Then, Ripper is applied on this cache of sentences and the resulting rules are added to the pool of features for the next maximum entropy extractor. Different cache sizes have been tested in this experiment, namely 50, 100, 200, 400 and 800 sentences.

This procedure continues until Ripper does not generate any new rule to be added to the pool of features or all unlabelled sentences have been used in the process. Note that this last condition implies that the maximum number of bootstrapping iterations depends on the cache size utilised — as each sentence is included into the cache just once. For example, as the MUC-7 dryrun corpus contains 4,113 sentences, with a cache of size 800 there can be at most six iterations while a with a cache of size 50 sentences more than 80 iterations could be run.

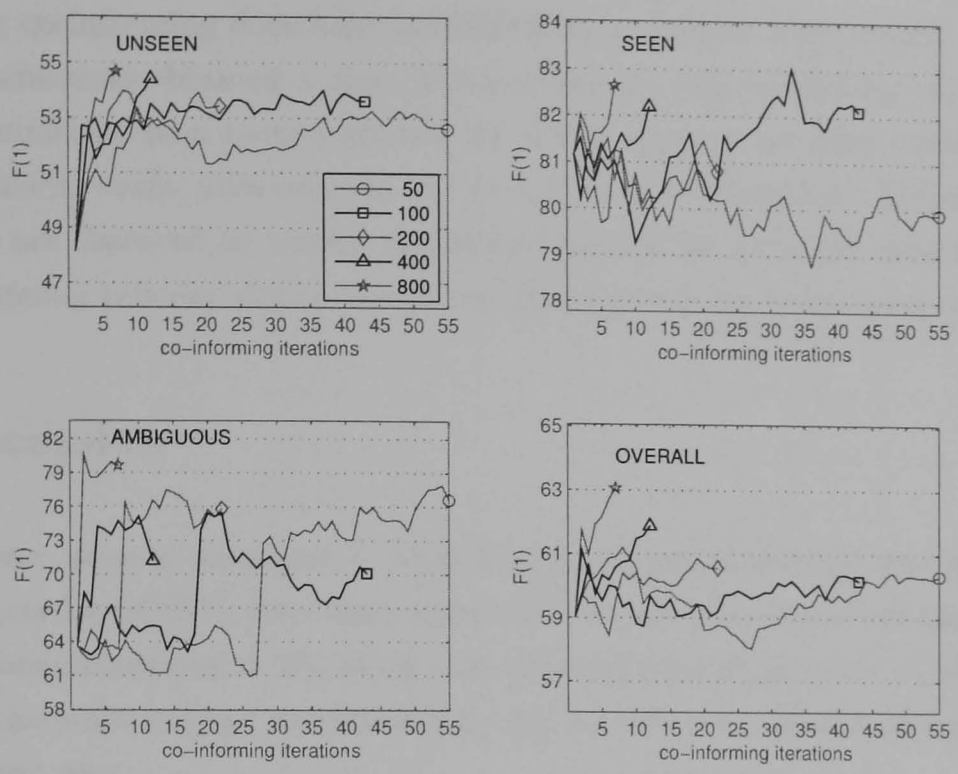


Figure 6.3: Comparison of the performances of co-informing LexMENE-V2 and Ripper for different cache sizes: features as used by LexMENE-V2, cutoff is set to 3 and GIS iterations to 200. Labelled examples: MUC-7 training corpus; unlabelled examples: MUC-7 dryrun test corpus; test examples: MUC-7 formal test corpus.

Figure 6.3 presents the performance of LexMENE^{Ripper} for the different cache sizes tested. It is interesting noticing that only with the cache of size 50 LexMENE^{Ripper} reaches convergence —i.e. no new rules can be obtained from the sentences in the cache— and for all other sizes the algorithm stops because the complete unlabelled material has been processed.

Figure 6.3 shows that LexMENE^{Ripper} is able to improved its initial overall performance —i.e. the one obtained with hand-labelled training data only— for all cache sizes tested. However, this experiment suggests that the approach is sensitive to the cache size parameter, an undesired characteristic which has also been reported for co-training (Pierce and Cardie 2001). In effect, versions with larger caches clearly outperform the versions with cache sizes of 200 sentences or less.

Moreover, the results shown when using caches of sizes 400 and 800 sentences follow a similar overall trend. Both start with a sharp increase during the initial bootstrapping iterations. Then the performance falls and then recovers to adopt a smoother, but rising path.

However, it is the cache of size 800 sentences the one that yields the best results obtaining increments of performance for all familiarity types of named entities, which is reflected in a 4% F-score overall improvement. Note however that LexMENE-V2 trained with the original lexical features, rather than the rules produced by Ripper, obtains better performance on this corpus.

Interestingly, co-informing does help LexMENE to recognise more unseen named entities as all cache sizes obtained a clear improvement on this familiarity type. However, results are mixed on seen named entities for which large cache sizes achieve improvements but smaller cache sizes actually led to a drop in performance. Although gains in performance are observed on ambiguous named entities for all cache sizes at the end of the bootstrapping process, the curves are oscillating with sharp increases and drops.

6.8 Discussion

In this chapter, several attempts to bootstrap a maximum entropy named entity extractor, namely LexMENE, have been presented. It has been explained that a MaxEnt Puzzle had been identified in the sense that providing large amounts of unlabelled examples to maximum entropy models could have no effect on their performance if used in the common way.

However, a new bootstrapping approach, named co-informing, has been proposed and tested. An experiment with co-informing shows a significant improvement of LexMENE's performance, though it also poses several questions. For example, it is reasonable to wonder whether this positive result would have been obtained if the initial rules produces by Ripper had maintained the performance of the uninformed version of LexMENE. Another interesting issue comes from the observation that figure 6.3 suggests that the bigger the cache, the better. Thus, it could be worth testing whether better results can be obtained from performing just one co-informing iteration with all unlabelled data. Different amounts of unlabelled data and cache sizes should be tested.

The main question, nevertheless, is that co-informing has introduced a new puzzle. It is normally expected that the performance of a learner deteriorates as bootstrapping progresses, as noise is unavoidably added to newly labelled data. This effect has indeed been observed in different bootstrapping experiments (Ng and Cardie 2003). In fact, co-informing seems to behave in this way in the initial iterations for all the different cache sizes after obtaining a peak in performance. However, and particularly for the versions with larger caches, this trend is reversed and the performance of LexMENE tends to rise.

There are two possible explanations for this phenomenon — or perhaps the combination of both. The first one, though unlikely, is that the rules produced by Ripper to classify the noisy sentences in later iterations, are complementing previous rules and, in this way, helping LexMENE to refine its maximum entropy model despite the occurrence of wrong labels. The second alternative is more plausible and is related to overfitting. Clearly larger caches lead to large sets of rules to represent the underlying classification task. Thus, the pool of features for the maximum entropy approach grows significantly after each iteration. But, because the number of GIS iterations has been kept constant,

the framework has to produce increasingly more general hypotheses which, in the end, bring about better results.

These open questions make difficult to declare hypothesis 3 valid —or invalid— as further exploration of the approach proposed here is required.

Chapter 7

Conclusions and future work

This chapter sums up the work conducted in this thesis and re-assesses the hypotheses that originated it. It also presents proposals for future research that aims to find solutions to unsolved issues, elucidate unanswered questions and extend the study presented here.

7.1 Conclusions

The main argument in this thesis is that it is possible to perform relatively accurate generic named entity extraction by combining the robustness of statistical learning methods, an appropriate management of exceptions and the utilisation of large amounts of unlabelled data. Such a system would be portable across named entity extraction tasks and, if the resources exist, across languages. However, this thesis has shown that several issues ought to be solved before this approach can be realised.

The maximum entropy framework has proved to be a strong classifier that is able to obtain good levels of performance by capturing relevant information from lexical data only, which demonstrates it is a suitable method for named entity extraction (basis 3). All the attempts presented in this work to improve this baseline approach have shown how challenging this might be and, though successful, most of the improvements obtained were not impressive.

Table 7.1 presents a summary of the best performance (F-score) of each approach studied for both MUC-7 test corpora. The first row corresponds to the performance obtained by version B of LexMENE which is the baseline system extended here. Row two presents the performance obtained by MOLI MENE V2 which uses exactly the same lexically-oriented features as LexMENE, but with different context windows. The third row shows the best performance achieved by MOLI MENE for the dryrun test corpus and for the formal test corpus. The former was obtained by MOLI MENE V8 and the latter

Table 7.1: Summary of the best performance (F-score) obtained by the named entity extraction approaches presented in this thesis.

	Approach name	Best Performance	
		Dryrun test	Formal test
(1)	LexMENE	79.54	64.51
(2)	MOLI MENE V2	83.98	71.49
(3)	MOLI MENE	84.68	73.05
(4)	biLexMENE	80.51	66.72
(5)	LexMENE^{Ripper}	78.18	63.06

by MOLI MENE V11. Both versions utilise, in addition to the features used by MOLI MENE V2, linguistically-oriented features —namely trigger synsets— extracted from WordNet®. Row four indicates the best results accomplished by biLexMENE. In both situations, the system is trained with all documents from the MUC-7 training corpus, but the highest F-score for the dryrun test corpus is obtained selecting 50 similar cases, whereas selecting 75 similar cases yielded the best result for the formal test corpus. Finally, the fifth row corresponds to the performance achieved by LexMENE^{Ripper} when using a cache of 800 sentences. However, it must be noticed that the dryrun test corpus was not the target corpus for this NEE system. The F-score reported here corresponds to the labels introduced by LexMENE^{Ripper} after six co-informing iterations on the dryrun documents when used as a source of unlabelled text.

It can be observed that the most significant increment in performance was obtained by re-arranging the context windows for the lexically-oriented features. Linguistically-oriented features do lead to better F-scores on both test corpora, but these improvements are not as important as expected: just around 1%. This makes difficult to decide on the validity of hypothesis 1, as these results were obtained using parameters estimated for MOLI MENE V2 and more significant improvements could be achieved with a different set of parameters. Thus, more research is needed to have more definitive evidence.

The source of linguistic information has been WordNet, which is general enough to not affect the portability of the approach. However, it might be possible that other sources could be more appropriate for this because WordNet does not explicitly make differences between concepts and instances of these concepts. For example, consider the word *president*. WordNet considers the texts *ex-president* and *Franklin Delano Roosevelt* as hyponyms of this word. This mix of conceptual levels makes difficult the estimation of both semantic distances and the identification of irrelevant senses (de Boni 2004), which might be valuable information to identify named entities.

The sheer amount of linguistic information collected from WordNet was another challenge to be faced. This problem was solved by using an efficient rule learning algorithm with the ability to manage set-valued attributes, namely Ripper (Cohen 1995, Cohen 1996), to select the most important features from the new data which are then fed into

the maximum entropy model. This approach was the most successful in increasing the performance of the NEE system.

Managing exceptions was less successful. Biasing the maximum entropy model, towards training material that is similar to the text being classified, obtained only marginal increments in performance which are not clearly related to a better handling of exceptions.

However, this material was retrieved by using a successful approach in the memory-based learning field, namely the Case Retrieval Nets (Lenz 1999), that turned out to be an unfortunate choice as the efficiency of the algorithm is affected when cases are pieces of free text. This remains the main unsolved issue of this idea, which discouraged testing the approach further with other ways of measuring similarity. As a consequence, it has not been possible to determine the validity of hypothesis 2.

Bootstrapping the maximum entropy approach was also intricate because of the so called MaxEnt Puzzle (Cui and Guthrie 2004). This puzzle states that unlabelled material could have no effect on the classifications made by a maximum entropy model, unless its output is externally processed before being added to the new, larger pool of training examples. Ripper was once again used for this task and a new bootstrapping framework was proposed, which was named co-informing. In the co-informing approach, Ripper informs the maximum entropy model of the most important features for classifying the new training material, which initially corresponds to hand-annotated examples. The maximum entropy NE extractor takes into account this information, obtains the classes for unlabelled sentences and informs Ripper of its finding.

Co-informing proved to be successful in overcoming the MaxEnt Puzzle and the performance increases as more and more unlabelled sentences are processed. Moreover, experiments in which larger number of sentences are processed at each iteration obtained better results and do not show signs of convergence, which could indicate that the performance of the approach might be increased further if more unlabelled material were available.

However, the overall performance reached by co-informing is below the one obtained by the baseline system. This was a consequence of lexical features being replaced with Ripper rules, rather than being added to — as above, which had a significant negative impact on the performance of the approach, degrading it in more than 7% F-score, at iteration one.

Once more, these findings do not allow a conclusive decision on the validity of hypothesis 3: on the one hand, LexMENE^{Ripper} improves by extracting new information from unlabelled text; on the other hand, this new information was not enough to outperform the baseline system that uses hand-labelled text only. Again, this could be related to the fixed parameters used in the experiment and further research is needed in order to obtain an assertive conclusion.

Nevertheless, the inital idea of combining these techniques remains valid. Bootstrapped examples can be easily added to the instance-based memory to be considered in the biasing process, and the co-informing framework could be applied to the pool of lexical features enriched with linguistical information drawn from WordNet. Further research is required on how to solve the issues that prevented this integretation here.

7.2 Future work

Future work has been divided into four areas depending on whether they will contribute to answer pending questions of the research conducted in previous chapters, as extensions of the approaches proposed, to improve the methods proffered to obtain NLP technology or to extend the research of the techniques used in this thesis.

7.2.1 Answering open questions

There are a number of questions that has been left open about the methods proposed in this thesis, which require further hypotheses and empirical work in order to be answered.

For example, the differences between the best performance obtained by each NEE system studied in this thesis are quite small. In particular, only a 1% improvement has been observed when linguistically-oriented features are added to MOLI MENE. It remains answered whether these differences have statistical significance, mainly because the procedure used for the MUC and CoNLL competitions was followed here, in which this consideration was not included.

In addition, the above marginal improvement was obtained by including linguistically-oriented features provided by WordNet® in the baseline maximum entropy NEE system, despite the fact that these features can be disorganised and noisy. Therefore, it also remains unanswered whether the application of word sense disambiguation approaches could produce a more helpful set of linguistically-oriented features that could further boost the performance of the baseline system.

In chapter 5, it was found that biasing a maximum entropy NEE system by applying memory-based techniques is only beneficial when little training material is available. However, these results are observed for the specific application discussed in this thesis which is based on Case Retrieval Nets (CRNs) (Lenz 1999). Although CRNs normally provide a fast case retrieval process, they show efficiency problems when managing attributes with a large range of values, as are normally found in NLP tasks. This prevented further parameterisation and experimentation with the approach. Thus, it is possible that other similarity functions or the use of sentences as cases —rather than fixed windows of tokens— could yield better improvements of a biased NEE system, but this would necessarily require abandoning CRNs and looking for an alternative method

for retrieving similar cases. One possibility to be explored is the TiMBL memory-based learner, which has been successfully applied to many NLP applications (Daelemans, Zavrel, van der Sloot and van den Bosch 2003).

It could be argued that it is not clear that by biasing a maximum entropy NEE system, a better management of exceptions is obtained. This ability was measured here by looking at improvements in recognising ambiguous named entities which presented a disparate occurrence between classes. A more objective evaluation could be made by studying indices designed specifically for measuring exceptionality such as class prediction strength, typicality and local typicality (Daelemans, van de Bosch and Savrel 1999, Salzberg 1990, Rotaru and Litman 2003).

Another open question is whether the co-informing bootstrapping approach proposed in chapter 6 is useful. Results indicate that it is indeed helping the underlying maximum entropy model to learn new features from unlabelled examples. However, the performance obtained by the resulting model is below the one obtained by the baseline system. Thus, it is not clear that this improvement could be observed if both algorithms start from the same level of accuracy.

Co-informing also posed a new puzzle: experiments with large cache sizes do not show the expected effect of noise being added to the training data. On the contrary, they indicate that if more unlabelled training material were available then the approach would be able to continue the learning process.

These two questions require further analysis and more experimentation, ideally on more than one classification problem and with larger amounts of unlabelled data.

7.2.2 Extensions of the proposed approaches

The corpora analysis presented in section 3.2 can be extended to include tokens and phrases that are not labelled as named entities in the target texts, but that have been seen as such during the training. This would provide information of the number of spurious named entities that a NEE system could wrongly recognise, which will improve the estimations of the complexity of the task and supply extra information of its ability to handle exceptions in the texts.

It could be possible to use the familiarity information of named entities to develop a NEE system with several components that are specialised on an individual type. Moreover, by adding the class "actually not a name", these specialised NEE components could be applied after an initial NEE extractor which aims to identify in the texts as many named entities as possible. Such an approach might provide evident benefits for the recall and the precision of the system that are optimised separately.

It must be observed that the approaches developed in this thesis complement each other. Thus, linguistically-oriented features can be included into both the biased and

bootstrapped approaches. Moreover, instance-based methods have the ability to easily include new examples to the case database and need no modification to be integrated with the bootstrapping method presented here. Therefore, once the efficiency problem of the biased system and the difficulties of the bootstrapping method are solved, all three NEE approaches can be combined into a single system.

Finally, in the co-informing settings presented here, Ripper provides the maximum entropy model with new features obtained from a cache of machine-labelled examples. There are a number of slightly different approaches which could be worth testing. For example:

- ▷ provide Ripper with only a subset of selected sentences from the cache
- ▷ let Ripper to get relevant features not only from the sentences in the iteration's cache but from all sentences selected so far
- ▷ apply Ripper to linguistically-oriented features only and then combine the resulting features with the lexically-oriented features

7.2.3 A end-user named entity extractor

Although obtaining high performance has not been the goal guiding this thesis, there are a number of simpler modifications which could contribute to obtaining a better end-user named entity extractor.

For example, the successful system MOLI MENE V2 presented in chapter 4 was empirically parametrised considering only lexically-oriented features. It was found that looking at the token on the left and the token on the right for lexical information yielded the best results. However, it was discussed in section 3.4.2 that such a limited contextual information could deprive NEE systems of important clues for identifying named entities.

Nevertheless, these facts are not completely contradictory as considering broader contexts for gathering lexical information could indeed make no contributions, but it could be possible that other types of information might benefit by more contextual material. On the other hand, this will increase the size of the pool of features and the application of Ripper will be necessary. Therefore, a better approach could be obtained by combining the selection method based on Ripper on both lexical and the "semantic" features and then performing an exhautive search of good parameters.

In addition, there is a large number of further features that could be added. For instance, both simple morphological features (Cucerzan and Yarowsky 2002, Tjong Kim Sang 2002b, Wu et al. 2002, Bender et al. 2003, Chieu and Ng 2003, Florian et al. 2003, Klein et al. 2003, Wu et al. 2003) and global features (Wu et al. 2003) —that relate different

occurrences of a candidate named entity— have been intensively used for extracting named entities.

Furthermore, a post-processing procedure could be included to identify and repair inconsistences in the labels that are output by the NEE system. The approach of retaining the boundaries of the named entities identified but then relabelling them showed improvements in 11 out of the 12 systems participating in CoNLL-2002 (Tjong Kim Sang 2002a). Also, the approach of partial matching (Mikheev, Moens and Grover 1999) can be applied so that the classifications of named entities that occur more than once in a document are based on the appearance with the best contextual information.

7.2.4 Future research

This thesis has established that WordNet could be a source of general information that contributes to the extraction of named entities. However, WordNet has been criticised because it is not consistent in the way it handles knowledge (de Boni 2004) which makes the extraction of useful semantic relationships difficult. Moreover, several ways of extending the knowledge of WordNet and ways in which this could be used to determine the semantic similarity between sentences have been proposed (Budanitsky and Hirst 2001, de Boni 2004). These approaches could be used to define more organised semantic features that might be more useful for a named entity extractor.

Moreover, these semantic relations could make sentences a more approapriate level of granularity for defining cases and queries in a memory-based biasing approach. Furthermore, the semantic distance between two sentences could be used as the similarity measure to identify relevant cases for a given query.

However, it has recently been suggested that a good treatment of exceptions in natural language applications is not only a property of instance-based methods. In fact, Ripper has been found to be quite good at this (Rotaru and Litman 2003). Although Ripper has been shown here to complement the statistical paradigm of maximum entropy models, it could be providing rules designed for exceptions rather than for regularity and it might turn out to be not an appropriate method for selecting features.

Therefore, an interesting source for further research is looking for ways in which the efficient induction used by Ripper can be modified to specifically obtain a robust method for selecting features for maximum entropy models. For instance, it is not clear than the pruning and deletion of rules applied by Ripper is appropriate for selecting features (Cohen 1995). Nor is the stopping criterion —based on the description length principle (Mitchell 1997)— that it uses. These techniques are included into the inducer as an attempt to increase generalisation and avoid overfitting. Thus, Ripper does not output all the rules and conditions it finds useful to undertake the extraction task.

But the maximum entropy framework deals with these issues and, when applied as a second learning step, it might be not necessary for Ripper to be concerned about overfitting the training data or creating irrelevant rules which could be managed by the generalisation of features described in section 4.11 and the appropriate estimation of the model's parameters.

Appendix A

Nymble implementation

As it might be expected, although Nymble's approach is well explained in Bikel et al. (1997) —and in some extent in Bikel et al. (1999)[1]— some of the details necessary for actually implementing the algorithm are missing and sensible guesses are required to fill the gaps. In the rest of this section, the implementation of siNymble version is discussed in detail.

A.1 Top-level model

The estimation of the probabilities for Nymble's top-level model is a simple procedure base in the function $c(e)$, which counts the number of times the event e occurs in the training data:

$$\Pr(NC|NC_{-1}, w_{-1}) = \frac{c(NC, NC_{-1}, w_{-1})}{c(NC_{-1}, w_{-1})} \quad \text{(A.1)}$$

$$\Pr(<w,f>_{first}|NC, NC_{-1}) = \frac{c(<w,f>_{first}, NC, NC_{-1})}{c(NC, NC_{-1})} \quad \text{(A.2)}$$

$$\Pr(<w,f>|<w,f>_{-1}, NC) = \frac{c(<w,f>, <w,f>_{-1}, NC)}{c(<w,f>_{-1}, NC)} \quad \text{(A.3)}$$

Bikel et al. clarifies that the model for name-classes (equation A.1) is conditioned on the previous *real* word of the previous name-class, unless the previous state is the start of a sentence in which case it would "be conditioned on the *+end+* word".

In the interpretation given here, siNymble contains actually two *completes models* only, which generate the top-level model and the backing-off/smoothing models (see table 3.4) by a full or a partial *instantiation* of their variables.

The first complete model is for generating probabilities for name-classes and contains three variables: the current name-class, the previous name-class and the previous word.

[1]This article describes IndentiFinder™, which seems to be remarkable similar to Nymble though a better performance is reported for this name entity recogniser.

$$(NC = a|NC_{-1} = b, w_{-1} = c) \tag{A.4}$$

The second complete model is for generating probabilities for words and contains six variables: the current word and feature, the previous word and feature, the current name-class and the previous name-class.

$$(w = p, f = q|w_{-1} = r, f_{-1} = s, NC = t, NC_{-1} = u) \tag{A.5}$$

Thus, $\Pr(NC = a|NC_{-1} = b)$ is interpreted as the sample probability of finding in the name-class complete model an event whose first variable has value a and second variable value b, considering all possible values for the third variable.

In general, a model $\Pr(x|y)$ is interpreted as the sample probability in the corresponding complete model, partially or fully instantiated with the values x and y. It will be written $\Pr(x|y) = \hat{Pr}(\hat{X} \models x|\hat{Y} \models y)$ to denote this operation. If the instantiation is partial, the count function will consider all possible values for all free variables. Formally, if a model $M = \Pr(x = \{x_1, \ldots, x_n\}|y = \{y_1, \ldots, y_m\})$ leaves k free variables $\bar{v}_1, \ldots, \bar{v}_k$ in the corresponding complete model, then the real estimation of M in these terms is

$$\begin{aligned}
\Pr(x|y) &= \hat{\Pr}(\hat{X} \models x|\hat{Y} \models y) \\
&= \frac{\hat{c}(\hat{X} \models x|\hat{Y} \models y)}{\hat{c}(\hat{X} \models *|\hat{Y} \models y)} \\
&= \frac{\hat{c}(\bar{x}_1 = x_1, \ldots, \bar{x}_n = x_n, \bar{y}_1 = y_1, \ldots, \bar{y}_m = y_m, \bar{v}_1 = *, \ldots, \bar{v}_k = *)}{\hat{c}(\bar{x}_1 = *, \ldots, \bar{x}_n = *, \bar{y}_1 = y_1, \ldots, \bar{y}_m = y_m, \bar{v}_1 = *, \ldots, \bar{v}_k = *)} \\
&= \frac{\sum_{v_1 \in \mathcal{D}_{v_1}} \cdots \sum_{v_k \in \mathcal{D}_{v_k}} c(x_1, \ldots, x_n, y_1, \ldots, y_m, v_1, \ldots, v_k)}{\sum_{x_1 \in \mathcal{D}_{x_1}} \cdots \sum_{x_n \in \mathcal{D}_{x_1}} \sum_{v_1 \in \mathcal{D}_{v_1}} \cdots \sum_{v_k \in \mathcal{D}_{v_k}} c(x_1, \ldots, x_n, y_1, \ldots, y_m, v_1, \ldots, v_k)}
\end{aligned} \tag{A.6}$$

where $\mathcal{D}_{v_j}$ is the set of possibles values for the variable $\bar{v}_j$.

For example, the model $\Pr(NC = a|NC_{-1} = b)$ will instantiate the name-class complete model (equation A.4) resulting in the following estimation

$$\begin{aligned}
\Pr(NC = a|NC_{-1} = b) &= \hat{\Pr}(\hat{X} \models a|\hat{Y} \models b) \\
&= \frac{\hat{c}(\hat{X} \models a|\hat{Y} \models b)}{\hat{c}(\hat{X} \models *|\hat{Y} \models b)} \\
&= \frac{\hat{c}(NC = a, NC_{-1} = b, w_{-1} = *)}{\hat{c}(NC = *, NC_{-1} = b, w_{-1} = *)} \\
&= \frac{\sum_{\bar{c} \in \mathcal{D}_{w_{-1}}} c(a, b, \bar{c})}{\sum_{\bar{a} \in \mathcal{D}_{NC}} \sum_{\bar{c} \in \mathcal{D}_{w_{-1}}} c(\bar{a}, b, \bar{c})}
\end{aligned}$$

similarly when siNymble requires $\Pr(<w,f> = <p,q>|NC = t)$, it is estimated as

$$= \frac{\sum_{\bar{r} \in \mathcal{D}_{w_{-1}}} \sum_{\bar{s} \in \mathcal{D}_{f_{-1}}} \sum_{\bar{u} \in \mathcal{D}_{NC_{-1}}} c(p, q, \bar{r}, \bar{s}, t, \bar{u})}{\sum_{\bar{p} \in \mathcal{D}_w} \sum_{\bar{q} \in \mathcal{D}_f} \sum_{\bar{r} \in \mathcal{D}_{w_{-1}}} \sum_{\bar{s} \in \mathcal{D}_{f_{-1}}} \sum_{\bar{u} \in \mathcal{D}_{NC_{-1}}} c(\bar{p}, \bar{q}, \bar{r}, \bar{s}, t, \bar{u})}$$

Note that $\mathcal{D}_{NC}$, $\mathcal{D}_{NC_{-1}}$ and $\mathcal{D}_{w_{-1}}$ are not the same sets in both complete models, as one is collecting statistics for changes of name-class (i.e. state transitions) while the other for the generation of words (i.e. emission probabilities).

This strategy of complete models works for all models used by siNymble —including the back-off and smoothing models discussed later— but two observations need to be considered: the meaning of the subscript *first* in equation A.2 and the values of $<w,f>_{-1}$ in equation A.3. These top-level components collect probabilities for first words in a name-class and subsequent word in a name-class respectively, and they must exclude counts for the other type. Therefore, the subscript *first* in the former implies that the value of the previous word in the complete model is restricted to $<+begin+,otw>$, and that the values of $<w,f>_{-1}$ in the latter cannot be instantiated with $<+begin+,otw>$. This last rectriction is automatic as subsequent words will never be seen after the magical word $<+begin+,otw>$.

With this discussion in mind, the equations for estimating the probabilities of the top-level model can be re-written as follows

$$\begin{aligned}
\Pr(NC = a | NC_{-1} = b, w_{-1} = c) &= \hat{\Pr}(\hat{X} \models a | \hat{Y} \models b, c) \\
\Pr(<w,f>_{first} = <p,q> | NC = t, NC_{-1} = u) &= \hat{\Pr}(\hat{X} \models p, q | \hat{Y} \models +begin+, otw, t, u) \\
\Pr(<w,f> = <p,q> | <w,f>_{-1} = <r,s>, NC = t) &= \hat{\Pr}(\hat{X} \models p, q | \hat{Y} \models r, s, t)
\end{aligned}$$

A.2 Training sequence

Given this top-level model, it can be determined how input training sentences are transformed into a sequence of training events for the system. Consider the following input sentences.

> Mr. <PERSON>**T. Jones**</PERSON> eats in <ORGANISATION>**Mcdoneld's**</ORGANISATION>. Mr. <PERSON>**Jones**</PERSON> is eating apples.

Table A.1 shows the training events which these two simple sentences produce. Note that actually shorter names are used in the implementation of siNymble. For example, Not-A-Name=NAN, first-word=fwd, Start-Of-Sent=SOS, etc. These shorter names will be used hereafter.

A.3 Decoding sequence

Now it can be determined how new sentences are transformed into a sequence of decoding events to be presented to siNymble's hidden Markov model. Consider the following input sentence.

Table A.1: Training events for siNymble's hidden Markov model.

NC	W	F	W_{-1}	F_{-1}	NC_{-1}
NAN	mr.	fwd	+*begin*+	otw	SOS
NAN	+*end*+	otw	mr.	fwd	NAN
PER	t.	cpp	+*begin*+	otw	NAN
PER	jones	icp	t.	cpp	PER
PER	+*end*+	otw	jones	icp	PER
NAN	eats	ucp	+*begin*+	otw	PER
NAN	in	ucp	eats	ucp	NAN
NAN	+*end*+	otw	in	ucp	NAN
ORG	mcdoneld	icp	+*begin*+	otw	NAN
ORG	's	otw	mcdoneld	icp	ORG
ORG	+*end*+	otw	's	otw	ORG
NAN	.	otw	+*begin*+	otw	ORG
NAN	+*end*+	otw	.	otw	NAN
NAN	mr.	fwd	+*begin*+	otw	SOS
NAN	+*end*+	otw	mr.	fwd	NAN
PER	jones	icp	+*begin*+	otw	NAN
PER	+*end*+	otw	jones	icp	PER
NAN	is	ucp	+*begin*+	otw	PER
NAN	eating	ucp	is	ucp	NAN
NAN	apples	ucp	eating	ucp	NAN
NAN	.	otw	apples	ucp	NAN
NAN	+*end*+	otw	.	otw	NAN

Table A.2: Decoding events for siNymble's hidden Markov model.

W	F	W_{-1}	F_{-1}
mr.	fwd	+*end*+	otw
jones	icp	mr.	fwd
eats	ucp	jones	icp
+UNK+	ucp	eats	ucp
in	ucp	+UNK+	ucp
+UNK+	icp	in	ucp
.	otw	+UNK+	icp

Mr. Jones eats bananas in Starebucks.

Table A.2 shows the decoding sequence for this sentence. The token *+UNK+* is an special token used by siNymble when managing words which are not seen during training. Section A.4 discusses this cases in more detail. Given the training events in table A.1, Nymble finds that the most likely word/name-class sequence for these events is

$$
\begin{aligned}
&\Pr(NAN|SOS, +end+) \cdot \Pr(<mr.,fwd>|NAN, SOS) \cdot \\
&\quad \Pr(<+end+,otw>|<mr.,fwd>, NAN) \cdot \\
&\Pr(PER|NAN, mr.) \cdot \Pr(<mr.,fwd>|PER, NAN) \cdot \\
&\quad \Pr(<+end+,otw>|<jones,icp>, PER) \cdot \\
&\Pr(NAN|PER, jones) \cdot \Pr(<eats,ucp>|NAN, PER) \cdot \\
&\quad \Pr(<+UNK+,ucp>|<eats,ucp>, NAN) \cdot \\
&\quad \Pr(<in,ucp>|<+UNK+,ucp>, NAN) \cdot \\
&\quad \Pr(<+end+,otw>|<in,ucp>, NAN) \cdot \\
&\Pr(ORG|NAN, in) \cdot \Pr(<+UNK+,icp>|ORG, NAN) \cdot \\
&\quad \Pr(<+end+,otw>|<+UNK+,icp>, ORG) \cdot \\
&\Pr(PER|NAN, mr.) \cdot \Pr(<jones,icp>|PER, NAN) \cdot \\
&\quad \Pr(<+end+,otw>|<.,otw>, NAN) \cdot \\
&\Pr(EOS|NAN, .)
\end{aligned}
$$

which corresponds to the correct sequence of named entity tags.

A.4 Unknown words

As any corpus-trained learner, siNymble will encounter unknown words —words which have not been seen in the training data— during decoding. The approach for handling unknown words is quite simple though the exact procedure is not very clear. Bikel et al. say

> "...we hold up 50% of our data to train the unknown word model (the vocabulary is built up on the first 50%), save these counts in training data file, then hold out the other 50% and concatenate these bigrams counts with the first unknown word-training file."

The interpretation of this approach given here is the following: using a vocabulary built up on the first half of the training data, counts are collected for *all* training bigram; when a bigram contains unknown words, they are replaced by the special lexeme *+UNK+* so that statistics for this token can be accumulated; then, the process is repeated but using a vocabulary built up from the other half of the training data. The frequencies obtained are then added together, which results in a model of "unknown words occurring in the midst of known words" (Bikel et al. 1997). Later when a bigram is found to contain unknown words during decoding, this model is used to estimated the equations which defines the top-level model.

A.5 Estimating top-level and back-off models

As with any n-gram language model, Nymble's top-level model also requires less informed models in which rely when a particular bigram has not been seen in the training

set. In this section, these schema is looked in detail and re-formulated in terms of the complete models. Table 3.4 presents these models as defined in Bikel et al. (1997).

The top-level model estimation for name-class generation can be re-written as

$$\begin{aligned} Pr(NC = a|NC_{-1} = b, w_{-1} = c) &= \hat{Pr}(\hat{X} \models a|\hat{Y} \models b, c) \\ &= \frac{\hat{c}(\hat{X} \models a|\hat{Y} \models b, c)}{\hat{c}(\hat{X} \models *|\hat{Y} \models b, c)} \end{aligned} \tag{A.7}$$

During decoding, if the event $e = (NC = a|NC_{-1} = b, w_{-1} = c)$ has not been seen during training, siNymble will resort to the event $e = (NC = a|NC_{-1} = b)$, whose estimation can be written as

$$\begin{aligned} \Pr(NC = a|NC_{-1} = b) &= \hat{\Pr}(\hat{X} \models a|\hat{Y} \models b) \\ &= \frac{\hat{c}(\hat{X} \models a|\hat{Y} \models b)}{\hat{c}(\hat{X} \models *|\hat{Y} \models b)} \end{aligned} \tag{A.8}$$

If this event has also not been seen during training, the system will back off further to the event $e = (NC = a)$, which is re-formulated to

$$\begin{aligned} \Pr(NC = a) &= \hat{\Pr}(\hat{X} \models a|\hat{Y} \models *) \\ &= \frac{\hat{c}(\hat{X} \models a|\hat{Y} \models *)}{\hat{c}(\hat{X} \models *|\hat{Y} \models *)} \end{aligned} \tag{A.9}$$

Finally, if this event is also unknown for the system, it will be backed off to a default constant probability given by

$$\frac{1}{|\mathcal{D}_{NC}|} \tag{A.10}$$

where $|\mathcal{D}_{NC}|$ is the number of name-class values seen for the variable NC in the name-class complete model (which is the same number as $|\mathcal{D}_{NC_{-1}}|$though the actual sets are different as the former contains the special state EOS in addition to the name-class regions, whereas the latter contains the special state SOS instead).

From section A.1, the top-level model estimation for first words of a name-class can be written as

$$\begin{aligned} \Pr(<w,f>_{first} = <p,q>|NC = t, NC_{-1} = u) &= \hat{\Pr}(\hat{X} \models p, q|\hat{Y} \models +begin+, otw, t, u) \\ &= \frac{\hat{c}(\hat{X} \models p, q|\hat{Y} \models +begin+, otw, t, u)}{\hat{c}(\hat{X} \models *|\hat{Y} \models +begin+, otw, t, u)} \end{aligned} \tag{A.11}$$

The back-off event for this top-level component is ($<w,f>=<p,q>|<w,f>_{-1}=<+begin+, otw>, NC=t$), whose sample probability is estimated by

$$\begin{aligned} \Pr(<w,f>=<p,q>|<w,f>_{-1}=<+begin+,otw>, NC=t) &= \hat{\Pr}(\hat{X} \models p, q|\hat{Y} \models +begin+, otw, t) \\ &= \frac{\hat{c}(\hat{X} \models p, q|\hat{Y} \models +begin+, otw, t)}{\hat{c}(\hat{X} \models *|\hat{Y} \models +begin+, otw, t)} \end{aligned} \tag{A.12}$$

When there has been no such event in the training data, siNymble resorts to the event ($<w,f> = <p,q>|NC = t$), which is estimated as

$$\begin{aligned} \Pr(<w,f> = <p,q>|NC = t) &= \hat{\Pr}(\hat{X} \models p,q|\hat{Y} \models t) \\ &= \frac{\hat{c}(\hat{X} \models p,q|\hat{Y} \models t)}{\hat{c}(\hat{X} \models *|\hat{Y} \models t)} \end{aligned} \tag{A.13}$$

If this event is also unknown, the system backs off to the multiplication of the probabilities of the events $(w = p|NC = t)$ and $(f = q|NC = t)$, thus

$$\begin{aligned} \Pr(w = p|NC = t) \cdot \Pr(f = q|NC = t) &= \hat{\Pr}(\hat{X} \models p|\hat{Y} \models t) \cdot \hat{\Pr}(\hat{X} \models q|\hat{Y} \models t) \\ &= \frac{\hat{c}(\hat{X} \models p|\hat{Y} \models t)}{\hat{c}(\hat{X} \models *|\hat{Y} \models t)} \cdot \frac{\hat{c}(\hat{X} \models q|\hat{Y} \models t)}{\hat{c}(\hat{X} \models *|\hat{Y} \models t)} \end{aligned} \tag{A.14}$$

If at least one of this events has not been seen during training, siNymble backs off to a default constant probability given by

$$\frac{1}{|\mathcal{D}_w||\mathcal{D}_f|} \tag{A.15}$$

where $|\mathcal{D}_w|$ is the number of different words seen for the variable w, $|\mathcal{D}_f|$ is the number of different lexical features seen for the variable f in the word complete model. Once more, these numbers are the same as $|\mathcal{D}_{w_{-1}}|$ and $|\mathcal{D}_{f_{-1}}|$(though $\mathcal{D}_w \neq \mathcal{D}_{w_{-1}}$, as the first domain contains the magic word *+end+* whereas the second one contains the magic word *+begin+* instead).

Finally, the estimation for the top-level component to generate subsequent words of a name-class can be written as

$$\begin{aligned} \Pr(<w,f> = <p,q>|<w,f>_{-1} = <r,s>, NC = t) &= \hat{\Pr}(\hat{X} \models p,q|\hat{Y} \models r,s,t) \\ &= \frac{\hat{c}(\hat{X} \models p,q|\hat{Y} \models r,s,t)}{\hat{c}(\hat{X} \models *|\hat{Y} \models r,s,t)} \end{aligned} \tag{A.16}$$

remembering that the previous word will never be instantiated with the magic word $<+begin+,otw>$.

When this estimation is not possible, siNymble follows the backing-off sequence to the models given in equations A.13, A.14 and A.15, until a result is obtained.

A.6 The backing-off and smoothing strategy

Another important aspect of n-gram language model is that they normarly require an smoothing procedure of the probabilities obtained from training. Bikel et al. (1997) defined an interesting smoothing strategy which automatically performs backing off when necessary.

This is achieved by assigning the appropriate *weights* to each model and its immediate back-off model. When both models are relevant, an smoothed estimation for the main

model is obtained. When there is no information available for a main event, a backed off estimation is attained by making the weight for the main model zero and one for the secondary one. The procedure to calculate a model's weight is again rather simple though important details that were missing had to be filled:

> "If computing $\Pr(X|Y)$, assign weight λ to the direct computation... and weight $(1-\lambda)$ to the back-off model, where
>
> $$\lambda = \left(1 - \frac{\text{old } c(Y)}{c(Y)}\right) \frac{1}{1 + \frac{\text{unique outcomes of } Y}{c(Y)}} \tag{A.17}$$
>
> where 'old $c(Y)$' is the sample size of the model from which we are backing off...
>
> This method... overcomes the problem when a back-off model has roughly the same amount of training as the current model, via the first factor... which essentially ignores the back-off model and puts all the weight on the primary model in such an equi-trained situation..." (Bikel et al. 1997)
>
> "The second factor... is based on the notion that the number of unique outcomes over the sample size is a crude measure of the certainty of the model..." (Bikel et al. 1999)

From this, it can be interpreted that the computation of the probability of a top-model M with back-off models $B_1, B_2, \ldots, B_k$ has the form

$$\begin{aligned}\Pr(M) = \lambda_M \Pr(M) + (1-\lambda_M) \cdot \\ (\lambda_{B_1} \Pr(B_1) + (1-\lambda_{B_1}) \cdot \\ (\lambda_{B_2} \Pr(B_2) + (1-\lambda_{B_2}) \cdot \\ \vdots \\ (\lambda_{k-1} \Pr(B_{k-1}) + (1-\lambda_{B_{k-1}}) \Pr(B_k)) \cdots))\end{aligned}$$

which is formulated by the recursive expression of equation A.18 in which m is a primary model and b is the immediate back-off model for m.

$$\tilde{\Pr}(m) = \begin{cases} \lambda_m \Pr(m) + (1-\lambda_m)\tilde{\Pr}(b) & \text{if a back-off model exists} \\ \Pr(m) & \text{otherwise} \end{cases} \tag{A.18}$$

The hard work here is to determine what exactly 'old $c(Y)$' means and how 'unique outcomes of Y' are counted for each model.

First it must be noted that $c(Y)$ in equation A.17 is equivalent to $\hat{c}(\hat{X} \models *|\hat{Y} \models y)$ in the complete model notation used here. So, 'old $c(Y)$' must correspond to $\hat{c}(\hat{X} \models *|\hat{Y'} \models y)$ for the model being backed off. Because top-level components are never secondary models, it can be assumed that 'old $c(Y) = 0$' for them, that is, the first term of λ is always one for top-level models.

For name-class models, 'old $c(Y)$' is straighforward as it can clearly be seen that each secondary model frees one variable with respect to the primary one. Thus, 'old $\hat{c}(\hat{X} \models *|\hat{Y} \models b)$' is $\hat{c}(\hat{X} \models *|\hat{Y} \models b, c)$ and 'old $\hat{c}(\hat{X} \models *|\hat{Y} \models *)$' is $\hat{c}(\hat{X} \models *|\hat{Y} \models b)$.

Word models are more tricky though, as some transformations are needed. At the beginning, things are quite straight as 'old $\hat{c}(\hat{X} \models p, q|\hat{Y} \models +begin+, otw, t)$' is the previous $\hat{c}(\hat{X} \models p, q|\hat{Y} \models +begin+, otw, t, u)$ and 'old $\hat{c}(\hat{X} \models p, q|\hat{Y} \models, t)$' is $\hat{c}(\hat{X} \models p, q|\hat{Y} \models r, s, t)$ whether <r,s> is the magical begin word (as for first word generation) or not (as for subsequent word generation) respectively.

Problems arise for the model $M = \hat{Pr}(\hat{X} \models p|\hat{Y} \models t) \cdot \hat{Pr}(\hat{X} \models q|\hat{Y} \models t)$. Each factor in this model is treated as an independent model, so that what it is multiplied are the estimations after smoothing. That is $M = \tilde{\hat{Pr}}(\hat{X} \models p|\hat{Y} \models t) \cdot \tilde{\hat{Pr}}(\hat{X} \models q|\hat{Y} \models t)$. Then, it must be solved the problem that free variables in the $\hat{Y}$ part in M has the same free variables than the $\hat{Y}$ part in the primary model $\hat{c}(\hat{X} \models p, q|\hat{Y} \models t)$, which would always make the first term of λ equals to zero for these submodels. This is done by defining 'old $\hat{c}(\hat{X} \models p|\hat{Y} \models t)$' and 'old $\hat{c}(\hat{X} \models q|\hat{Y} \models t)$' as $\hat{c}(\hat{X} \models p, q|\hat{Y} \models t)$, which makes perfect sense with the equi-trained situation explained in Bikel et al. (1997), though variables belonging to the $\hat{X}$ are used.

For determining 'unique outcomes of Y', Bikel et al. give further clues with the following example.

> "As an example—disregarding the first factor—if we saw the bigram 'come hither' once in training and we saw 'come here' three times, and nowhere else did we see the word 'come' in NOT-A-NAME class, when computing Pr(*'hither'*|*'come'*, *NOT-A-NAME*), we would back off to the unigram probability Pr(*'hither'*|*NOT-A-NAME*) with a weight of $\frac{1}{3}$, since the number of unique outcomes for the word-state for 'come' would be two, and the total number of times 'come' had been the preceding word in a bigram would be four (a $1/\left(1+\frac{2}{4}\right) = \frac{2}{3}$ weight for the bigram probability, a $1-\frac{2}{3} = \frac{1}{3}$ weight for the back-off model)."

Analising this example, it can be noticed that 'unique outcomes of Y' follows the same considerations than for 'old $c(Y)$' by replacing the function $\hat{c}(\hat{X} \models x|\hat{Y} \models y)$ by the function $\hat{u}(\hat{X} \models x|\hat{Y} \models y)$ define as

$$\hat{u}(\hat{X} \models x|\hat{Y} \models y) = \begin{cases} 1 & \text{if } \hat{c}(\hat{X} \models x|\hat{Y} \models y) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Table A.3 shows the formulation of the λ-weight for each model used by siNymble.

Table A.3: The λ-weight for each model used by siNymble.

Model	λ-weight
$\hat{\Pr}(\hat{X} \models a \mid \hat{Y} \models b, c)$	$\frac{1}{1+\frac{\hat{u}(\hat{X}\models a \mid \hat{Y}\models b,c)}{\hat{c}(\hat{X}\models * \mid \hat{Y}\models b,c)}}$
$\hat{\Pr}(\hat{X} \models a \mid \hat{Y} \models b)$	$\left(1 - \frac{\hat{c}(\hat{X}\models * \mid \hat{Y}\models b,c)}{\hat{c}(\hat{X}\models * \mid \hat{Y}\models b)}\right)\frac{1}{1+\frac{\hat{u}(\hat{X}\models * \mid \hat{Y}\models b)}{\hat{c}(\hat{X}\models * \mid \hat{Y}\models b)}}$
$\hat{\Pr}(\hat{X} \models a \mid \hat{Y} \models *)$	$\left(1 - \frac{\hat{c}(\hat{X}\models * \mid \hat{Y}\models b)}{\hat{c}(\hat{X}\models * \mid \hat{Y}\models *)}\right)\frac{1}{1+\frac{\hat{u}(\hat{X}\models * \mid \hat{Y}\models *)}{\hat{c}(\hat{X}\models * \mid \hat{Y}\models *)}}$
$\hat{\Pr}(\hat{X} \models p,q \mid \hat{Y} \models +begin+, otw, t, u)$	$\frac{1}{1+\frac{\hat{u}(\hat{X}\models p,q \mid \hat{Y}\models +begin+,otw,t,u)}{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models +begin+,otw,t,u)}}$
$\hat{\Pr}(\hat{X} \models p,q \mid \hat{Y} \models +begin+, otw, t)$	$\left(1 - \frac{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models +begin+,otw,t,u)}{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models +begin+,otw,t)}\right)\frac{1}{1+\frac{\hat{u}(\hat{X}\models p,q \mid \hat{Y}\models +begin+,otw,t)}{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models +begin+,otw,t)}}$
$\hat{\Pr}(\hat{X} \models p,q \mid \hat{Y} \models r,s,t)$	$\frac{1}{1+\frac{\hat{u}(\hat{X}\models p,q \mid \hat{Y}\models r,s,t)}{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models r,s,t)}}$
$\hat{\Pr}(\hat{X} \models p,q \mid \hat{Y} \models t)$	$\left(1 - \frac{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models r,s,t)}{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models t)}\right)\frac{1}{1+\frac{\hat{u}(\hat{X}\models p,q \mid \hat{Y}\models t)}{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models t)}}$
$\hat{\Pr}(\hat{X} \models p \mid \hat{Y} \models t)$	$\left(1 - \frac{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models t)}{\hat{c}(\hat{X}\models p \mid \hat{Y}\models t)}\right)\frac{1}{1+\frac{\hat{u}(\hat{X}\models p \mid \hat{Y}\models t)}{\hat{c}(\hat{X}\models p \mid \hat{Y}\models t)}}$
$\hat{\Pr}(\hat{X} \models q \mid \hat{Y} \models t)$	$\left(1 - \frac{\hat{c}(\hat{X}\models p,q \mid \hat{Y}\models t)}{\hat{c}(\hat{X}\models q \mid \hat{Y}\models t)}\right)\frac{1}{1+\frac{\hat{u}(\hat{X}\models q \mid \hat{Y}\models t)}{\hat{c}(\hat{X}\models q \mid \hat{Y}\models t)}}$

Table A.4: Training events for the name-class complete model.

NC	NC_{-1}	w_{-1}
NAN	SOS	+end+
PER	NAN	mr.
NAN	PER	jones
ORG	NAN	in
NAN	ORG	's
EOS	NAN	.
NAN	SOS	+end+
PER	NAN	mr.
NAN	PER	jones
EOS	NAN	.

A.7 Training: a walk-through example

The first step is to generate the actual training events for each of the complete models in use. Table A.4 shows the training events for the name-class complete model given the training input of table A.1 on page 183. Note how only transitions between name-classes

Table A.5: Training events for the word complete model.

w	f	w_{-1}	f_{-1}	NC	NC_{-1}
mr.	fwd	+begin+	otw	NAN	SOS
+end+	otw	mr.	fwd	NAN	NAN
t.	cpp	+begin+	otw	PER	NAN
jones	icp	t.	cpp	PER	PER
+end+	otw	jones	icp	PER	PER
eats	ucp	+begin+	otw	NAN	PER
in	ucp	eats	ucp	NAN	NAN
+end+	otw	in	ucp	NAN	NAN
mcdoneld	icp	+begin+	otw	ORG	NAN
's	otw	mcdonald	icp	ORG	ORG
+end+	otw	's	otw	ORG	ORG
.	otw	+begin+	otw	NAN	ORG
+end+	otw	.	otw	NAN	NAN
mr.	fwd	+begin+	otw	NAN	SOS
+end+	otw	mr.	fwd	NAN	NAN
jones	icp	+begin+	otw	PER	NAN
+end+	otw	jones	icp	PER	PER
is	ucp	+begin+	otw	NAN	PER
eating	ucp	is	ucp	NAN	NAN
apples	ucp	eating	ucp	NAN	NAN
.	otw	apples	ucp	NAN	NAN
+end+	otw	.	otw	NAN	NAN

are considered and that transitions to the special state *EOS* has been included.

Table A.5 presents the training events for the word complete model which closely follows the training input in table A.1.

Now, it is necessary to apply the procedure for generating statistics about unknown word as explained in section A.4. Tables A.6 and A.7 show the resulting training events for the example input. Note how these tables has twice the original amount of training.

The first step of the training is the counting of events. Tables A.8 and A.9 detail the resulting figures for all required counts.

Table A.6: Training events for the name-class complete model.

NC	NC_{-1}	w_{-1}
NAN	SOS	+end+
PER	NAN	mr.
NAN	PER	jones
ORG	NAN	in
NAN	ORG	's
EOS	NAN	+UNK+
NAN	SOS	+end+
PER	NAN	mr.
NAN	PER	jones
EOS	NAN	+UNK+
NAN	SOS	+end+
PER	NAN	mr.
NAN	PER	jones
ORG	NAN	+UNK+
NAN	ORG	+UNK+
EOS	NAN	.
NAN	SOS	+end+
PER	NAN	mr.
NAN	PER	jones
EOS	NAN	.

Table A.7: Training events for the word complete model.

w	f	w_{-1}	f_{-1}	NC	NC_{-1}
mr.	fwd	+begin+	otw	NAN	SOS
+end+	otw	mr.	fwd	NAN	NAN
t.	cpp	+begin+	otw	PER	NAN
jones	icp	t.	cpp	PER	PER
+end+	otw	jones	icp	PER	PER
eats	ucp	+begin+	otw	NAN	PER
in	ucp	eats	ucp	NAN	NAN
+end+	otw	in	ucp	NAN	NAN
mcdonald	icp	+begin+	otw	ORG	NAN
's	otw	mcdonald	icp	ORG	ORG
+end+	otw	's	otw	ORG	ORG
+UNK+	otw	+begin+	otw	NAN	ORG
+end+	otw	+UNK+	otw	NAN	NAN
mr.	fwd	+begin+	otw	NAN	SOS
+end+	otw	mr.	fwd	NAN	NAN
jones	icp	+begin+	otw	PER	NAN
+end+	otw	jones	icp	PER	PER
+UNK+	ucp	+begin+	otw	NAN	PER
+UNK+	ucp	+UNK+	ucp	NAN	NAN
+UNK+	ucp	+UNK+	ucp	NAN	NAN
+UNK+	otw	+UNK+	ucp	NAN	NAN
+end+	otw	+UNK+	otw	NAN	NAN
mr.	fwd	+begin+	otw	NAN	SOS
+end+	otw	mr.	fwd	NAN	NAN
+UNK+	cpp	+begin+	otw	PER	NAN
jones	icp	+UNK+	cpp	PER	PER
+end+	otw	jones	icp	PER	PER
+UNK+	ucp	+begin+	otw	NAN	PER
+UNK+	ucp	+UNK+	ucp	NAN	NAN
+end+	otw	+UNK+	ucp	NAN	NAN
+UNK+	icp	+begin+	otw	ORG	NAN
+UNK+	otw	+UNK+	icp	ORG	ORG
+end+	otw	+UNK+	otw	ORG	ORG
.	otw	+begin+	otw	NAN	ORG
+end+	otw	.	otw	NAN	NAN
mr.	fwd	+begin+	otw	NAN	SOS
+end+	otw	mr.	fwd	NAN	NAN
jones	icp	+begin+	otw	PER	NAN
+end+	otw	jones	icp	PER	PER
is	ucp	+begin+	otw	NAN	PER

Table A.8: The counting function applied to the events for the name-class complete model.

$\mathbf{NC = a}$	$\mathbf{NC_{-1} = b}$	$\mathbf{w_{-1} = c}$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{a} \vert \hat{\mathbf{Y}} \models \mathbf{b, c})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models * \vert \hat{\mathbf{Y}} \models \mathbf{b, c})$
EOS	NAN	.	2	2
NAN	ORG	's	1	1
NAN	PER	jones	2	2
NAN	SOS	+end+	2	2
ORG	NAN	in	1	1
PER	NAN	mr.	2	2

$\mathbf{NC = a}$	$\mathbf{NC_{-1} = b}$	$\mathbf{w_{-1} = c}$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{a} \vert \hat{\mathbf{Y}} \models \mathbf{b})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models * \vert \hat{\mathbf{Y}} \models \mathbf{b})$
EOS	NAN	.	2	5
NAN	ORG	's	1	1
NAN	PER	jones	2	2
NAN	SOS	+end+	2	2
ORG	NAN	in	1	5
PER	NAN	mr.	2	5

$\mathbf{NC = a}$	$\mathbf{NC_{-1} = b}$	$\mathbf{w_{-1}}$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{a} \vert \hat{\mathbf{Y}} \models *)$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models * \vert \hat{\mathbf{Y}} \models *)$
EOS	NAN	.	2	10
NAN	ORG	's	5	10
NAN	PER	jones	5	10
NAN	SOS	+end+	5	10
ORG	NAN	in	1	10
PER	NAN	mr.	2	10

Table A.9: The counting function applied to the events for the word complete model.

$\mathbf{w = p}$	$\mathbf{f = q}$	$\mathbf{NC = t}$	$\mathbf{NC_{-1} = u}$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{p,q}\vert \hat{\mathbf{Y}} \models \mathbf{+begin+, otw, t, u})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models *\vert \hat{\mathbf{Y}} \models \mathbf{+begin+, otw, t, u})$
.	otw	NAN	ORG	1	1
eats	ucp	NAN	PER	1	2
is	ucp	NAN	PER	1	2
mr.	fwd	NAN	SOS	2	2
mcdonald	icp	ORG	NAN	1	1
jones	icp	PER	NAN	1	2
t.	cpp	PER	NAN	1	2

$\mathbf{w = p}$	$\mathbf{f = q}$	$\mathbf{NC = t}$	$\mathbf{NC_{-1}}$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{p,q}\vert \hat{\mathbf{Y}} \models \mathbf{+begin+, otw, t})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models *\vert \hat{\mathbf{Y}} \models \mathbf{+begin+, otw, t})$
.	otw	NAN	ORG	1	5
eats	ucp	NAN	PER	1	5
is	ucp	NAN	PER	1	5
mr.	fwd	NAN	SOS	2	5
mcdonald	icp	ORG	NAN	1	1
jones	icp	PER	NAN	1	2
t.	cpp	PER	NAN	1	2

$\mathbf{w = p}$	$\mathbf{f = q}$	$\mathbf{w_{-1} = r}$	$\mathbf{f_{-1} = s}$	$\mathbf{NC = t}$	$\mathbf{NC_{-1}}$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{p,q}\vert \hat{\mathbf{Y}} \models \mathbf{r,s,t})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models *\vert \hat{\mathbf{Y}} \models \mathbf{r,s,t})$
's	otw	mcdonald	icp	ORG	ORG	1	1
+end+	otw	's	otw	ORG	ORG	1	1
+end+	otw	.	otw	NAN	NAN	2	2
+end+	otw	in	ucp	NAN	NAN	1	1
+end+	otw	jones	icp	PER	PER	2	2
+end+	otw	mr.	fwd	NAN	NAN	2	2
.	otw	apples	ucp	NAN	NAN	1	1
apples	ucp	eating	ucp	NAN	NAN	1	1
eating	ucp	is	ucp	NAN	NAN	1	1
in	ucp	eats	ucp	NAN	NAN	1	1
jones	icp	t.	cpp	PER	PER	1	1

$\mathbf{w = p}$	$\mathbf{f = q}$	$\mathbf{w_{-1}}$	$\mathbf{f_{-1}}$	$\mathbf{NC = t}$	$\mathbf{NC_{-1}}$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{p,q}\vert \hat{\mathbf{Y}} \models \mathbf{t})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{p}\vert \hat{\mathbf{Y}} \models \mathbf{t})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models \mathbf{q}\vert \hat{\mathbf{Y}} \models \mathbf{t})$	$\hat{\mathbf{c}}(\hat{\mathbf{X}} \models *\vert \hat{\mathbf{Y}} \models \mathbf{t})$
's	otw	mcdonald	icp	ORG	ORG	1	1	2	3
+end+	otw	's	otw	ORG	ORG	1	1	2	3
+end+	otw	.	otw	NAN	NAN	5	5	7	14
+end+	otw	in	ucp	NAN	NAN	5	5	7	14
+end+	otw	jones	icp	PER	PER	2	2	2	5
+end+	otw	mr.	fwd	NAN	NAN	5	5	7	14
.	otw	+begin+	otw	NAN	ORG	2	2	7	14
.	otw	apples	ucp	NAN	NAN	2	2	7	14
apples	ucp	eating	ucp	NAN	NAN	1	1	5	14
eating	ucp	is	ucp	NAN	NAN	1	1	5	14
eats	ucp	+begin+	otw	NAN	PER	1	1	5	14
in	ucp	eats	ucp	NAN	NAN	1	1	5	14
is	ucp	+begin+	otw	NAN	PER	1	1	5	14
jones	icp	+begin+	otw	PER	NAN	2	2	2	5
jones	icp	t.	cpp	PER	PER	2	2	2	5
mcdonald	icp	+begin+	otw	ORG	NAN	1	1	1	3
mr.	fwd	+begin+	otw	NAN	SOS	2	2	2	14
t.	cpp	+begin+	otw	PER	NAN	1	1	1	5

Table A.10: Sample probabilities for the name-class complete model.

$NC = a$	$NC_{-1} = b$	$w_{-1} = c$	$\hat{Pr}(\hat{X} \models a \mid \hat{Y} \models b, c)$	$\hat{Pr}(\hat{X} \models a \mid \hat{Y} \models b)$	$\hat{Pr}(\hat{X} \models a \mid \hat{Y} \models *)$	$\frac{1}{\lvert\mathcal{D}_{NC}\rvert}$
EOS	NAN	.	1.0	0.4	0.2	0.25
NAN	ORG	's	1.0	1.0	0.5	0.25
NAN	PER	jones	1.0	1.0	0.5	0.25
NAN	SOS	+end+	1.0	1.0	0.5	0.25
ORG	NAN	in	1.0	0.2	0.1	0.25
PER	NAN	mr.	1.0	0.4	0.2	0.25

Table A.11: Sample probabilities for the word complete model.

$w = p$	$f = q$	$NC = t$	$NC_{-1} = u$	$\hat{Pr}(\hat{X} \models p, q \mid \hat{Y} \models +begin+, otw, t, u)$	$\hat{Pr}(\hat{X} \models p, q \mid \hat{Y} \models +begin+, otw, t)$
.	otw	NAN	ORG	1.0	0.2
eats	ucp	NAN	PER	0.5	0.2
is	ucp	NAN	PER	0.5	0.2
mr.	fwd	NAN	SOS	0.5	0.5
mcdonald	icp	ORG	NAN	1.0	1.0
jones	icp	PER	NAN	1.0	0.4
t.	cpp	PER	NAN	0.5	0.5

$w = p$	$f = q$	$w_{-1} = r$	$f_{-1} = s$	$NC = t$	NC_{-1}	$\hat{Pr}(\hat{X} \models p, q \mid \hat{Y} \models r, s, t)$
's	otw	mcdonald	icp	ORG	ORG	1.0
+end+	otw	's	otw	ORG	ORG	1.0
+end+	otw	.	otw	NAN	NAN	1.0
+end+	otw	in	ucp	NAN	NAN	1.0
+end+	otw	jones	icp	PER	PER	1.0
+end+	otw	mr.	fwd	NAN	NAN	1.0
.	otw	apples	ucp	NAN	NAN	1.0
+end+	otw	.	otw	NAN	NAN	1.0
apples	ucp	eating	ucp	NAN	NAN	1.0
eating	ucp	is	ucp	NAN	NAN	1.0
in	ucp	eats	ucp	NAN	NAN	1.0
jones	icp	t.	cpp	PER	PER	1.0

$w = p$	$f = q$	w_{-1}	f_{-1}	NC = t	NC_{-1}	$\hat{Pr}(\hat{X} \models p, q \mid \hat{Y} \models t)$	$\hat{Pr}(\hat{X} \models p \mid \hat{Y} \models t)$	$\hat{Pr}(\hat{X} \models q \mid \hat{Y} \models t)$	$\frac{1}{\lvert\mathcal{D}_w\rvert\lvert\mathcal{D}_f\rvert}$
's	otw	mcdonald	icp	ORG	ORG	0.33333	0.33333	0.66667	0.01667
+end+	otw	's	otw	ORG	ORG	0.33333	0.33333	0.66667	0.01667
+end+	otw	.	otw	NAN	NAN	0.35714	0.35714	0.5	0.01667
+end+	otw	in	ucp	NAN	NAN	0.35714	0.35714	0.5	0.01667
+end+	otw	jones	icp	PER	PER	0.4	0.4	0.4	0.01667
+end+	otw	mr.	fwd	NAN	NAN	0.35714	0.35714	0.5	0.01667
.	otw	+begin+	otw	NAN	ORG	0.14286	0.14286	0.5	0.01667
.	otw	apples	ucp	NAN	NAN	0.14286	0.14286	0.5	0.01667
apples	ucp	eating	ucp	NAN	NAN	0.07143	0.07143	0.35714	0.01667
eating	ucp	is	ucp	NAN	NAN	0.07143	0.07143	0.35714	0.01667
eats	ucp	+begin+	otw	NAN	PER	0.07143	0.07143	0.35714	0.01667
in	ucp	eats	ucp	NAN	NAN	0.07143	0.07143	0.35714	0.01667
is	ucp	+begin+	otw	NAN	PER	0.07143	0.07143	0.35714	0.01667
jones	icp	+begin+	otw	PER	NAN	0.4	0.4	0.4	0.01667
jones	icp	t.	cpp	PER	PER	0.4	0.4	0.4	0.01667
mcdonald	icp	+begin+	otw	ORG	NAN	0.33333	0.33333	0.33333	0.01667
mr.	fwd	+begin+	otw	NAN	SOS	0.14286	0.14286	0.14286	0.01667
t.	cpp	+begin+	otw	PER	NAN	0.2	0.2	0.2	0.01667

The probability of each model for every event in the training can be calculated now. Tables A.10 and A.11 list the probabilities of each complete model for every training event. Note that these probabilities are not the final smoothed probabilities which are actually used.

Table A.12: The unique function applied to the events for the name-class complete model.

NC	$NC_{-1} = b$	$w_{-1} = c$	$\hat{u}(\hat{X} \models *\|\hat{Y} \models b, c)$	$\hat{u}(\hat{X} \models *\|\hat{Y} \models b)$	$\hat{u}(\hat{X} \models *\|\hat{Y} \models *)$
EOS	NAN	.	1	3	6
NAN	ORG	's	1	1	6
NAN	PER	jones	1	1	6
NAN	SOS	+end+	1	1	6
ORG	NAN	in	1	3	6
PER	NAN	mr.	1	3	6

Table A.13: The unique function applied to the events for the word complete model.

w	f	$NC = t$	$NC_{-1} = u$	$\hat{u}(\hat{X} \models *\|$ $\hat{Y} \models +begin+, otw, t, u)$	$\hat{u}(\hat{X} \models *\|$ $\hat{Y} \models +begin+, otw, t)$
.	otw	NAN	ORG	1	4
eats	ucp	NAN	PER	2	4
is	ucp	NAN	PER	2	4
mr.	fwd	NAN	SOS	1	4
mcdonald	icp	ORG	NAN	1	1
jones	icp	PER	NAN	2	2
t.	cpp	PER	NAN	2	2

w	f	$w_{-1} = r$	$f_{-1} = s$	$NC = t$	NC_{-1}	$\hat{u}(\hat{X} \models *\|\hat{Y} \models r, s, t)$
's	otw	mcdonald	icp	ORG	ORG	1
+end+	otw	's	otw	ORG	ORG	1
+end+	otw	.	otw	NAN	NAN	1
+end+	otw	in	ucp	NAN	NAN	1
+end+	otw	jones	icp	PER	PER	1
+end+	otw	mr.	fwd	NAN	NAN	1
.	otw	apples	ucp	NAN	NAN	1
apples	ucp	eating	ucp	NAN	NAN	1
eating	ucp	is	ucp	NAN	NAN	1
in	ucp	eats	ucp	NAN	NAN	1
jones	icp	t.	cpp	PER	PER	1

$w = p$	$f = q$	w_{-1}	f_{-1}	$NC = t$	NC_{-1}	$\hat{u}(\hat{X} \models p\|$ $\hat{Y} \models t)$	$\hat{u}(\hat{X} \models q\|$ $\hat{Y} \models t)$	$\hat{u}(\hat{X} \models *\|$ $\hat{Y} \models t)$
's	otw	mcdonald	icp	ORG	ORG	1	2	3
+end+	otw	's	otw	ORG	ORG	1	2	3
+end+	otw	.	otw	NAN	NAN	3	5	11
+end+	otw	in	ucp	NAN	NAN	3	5	11
+end+	otw	jones	icp	PER	PER	1	1	4
+end+	otw	mr.	fwd	NAN	NAN	3	5	11
.	otw	+begin+	otw	NAN	ORG	2	5	11
.	otw	apples	ucp	NAN	NAN	2	5	11
apples	ucp	eating	ucp	NAN	NAN	1	5	11
eating	ucp	is	ucp	NAN	NAN	1	5	11
eats	ucp	+begin+	otw	NAN	PER	1	5	11
in	ucp	eats	ucp	NAN	NAN	1	5	11
is	ucp	+begin+	otw	NAN	PER	1	5	11
jones	icp	+begin+	otw	PER	NAN	2	2	4
jones	icp	t.	cpp	PER	PER	2	2	4
mcdonald	icp	+begin+	otw	ORG	NAN	1	1	3
mr.	fwd	+begin+	otw	NAN	SOS	1	1	11
t.	cpp	+begin+	otw	PER	NAN	1	1	4

Therefore, the next step is to calculate the λ-weights for each model. For that, first it is needed the unique counts for each primary model. These are shown in Tables A.12 and A.13.

Table A.14: λ-weights for the name-class complete model.

$NC = a$	$NC_{-1} = b$	$w_{-1} = c$	$\hat{Pr}(\hat{X} \models a \vert \hat{Y} \models b, c)$	$\hat{Pr}(\hat{X} \models a \vert \hat{Y} \models b)$		$\hat{Pr}(\hat{X} \models a \vert \hat{Y} \models *)$	
			λ^0	λ^0	λ	λ^0	λ
EOS	NAN	.	0.66667	0.625	0.375	0.625	0.3125
NAN	ORG	's	0.5	0.5	0	0.625	0.5625
NAN	PER	jones	0.66667	0.66667	0	0.625	0.5
NAN	SOS	+end+	0.66667	0.66667	0	0.625	0.5
ORG	NAN	in	0.5	0.625	0.5	0.625	0.3125
PER	NAN	mr.	0.66667	0.625	0.375	0.625	0.3125

Now the λ-weights can be computed for each model, though there is an important observation that needs to be made here. For each model in the middle of the backing-off/smoothing strategy —i.e. not a top-level component nor a final constant, default model— two λ-weights are actually calculated: one for smoothing a higher-level model (which considers 'old $c(Y)$') and one for backing off a higher-level model (which fixes 'old $c(Y)$' to zero). It will be written λ^0 for the latter weight to differentiate it from the former. Moreover, the model $\Pr(<w,f>|NC)$ smoothes two different models —namely $\Pr(<w,f>|<+begin+,otw>, NC)$ and $\Pr(<w,f>|<w,f>_{-1}, NC)$— and consequently it has two smoothing λ-weights. For this model, λ_f will be written to denote the weight for the first word model and λ_s to denote the weight for the top-level component for subsequent words. Tables A.14 and A.15 present the resulting weights for each model/event.

Finally, the conditions to calculate the final sample probabilities are met. Here the word complete model is separated into the original two models for first and subsequent words in a name-class. These figures are compute by applying equation A.18, but considering that λ must be used when doing smoothing and λ^0 when estimating a back-off probability. Tables A.16, A.17 and A.18 present the final probabilities for the top-level model and their back-off models.

This same procedure is applied to the events with unknwon words. Tables A.19, A.20 and A.21 show the final probabilities for the top-level model and their back-off models considering unknown words.

Table A.15: λ-weights for the word complete model.

$\mathbf{w=p}$	$\mathbf{f=q}$	$\mathbf{NC=t}$	$\mathbf{NC_{-1}=u}$	$\hat{\Pr}(\hat{\mathbf{X}} \models \mathbf{p,q} \mid \hat{\mathbf{Y}} \models \mathbf{+begin+, otw, t, u})$	$\hat{\Pr}(\hat{\mathbf{X}} \models \mathbf{p.q} \mid \hat{\mathbf{Y}} \models \mathbf{+begin+, otw, t})$	
				λ^0	λ^0	λ
.	otw	NAN	ORG	0.5	0.55556	0.44444
eats	ucp	NAN	PER	0.5	0.55556	0.33333
is	ucp	NAN	PER	0.5	0.55556	0.33333
mr.	fwd	NAN	SOS	0.5	0.5	0
mcdonald	icp	ORG	NAN	0.5	0.5	0
jones	icp	PER	NAN	0.66667	0.55556	0.33333
t.	cpp	PER	NAN	0.5	0.5	0

$\mathbf{w=p}$	$\mathbf{f=q}$	$\mathbf{w_{-1}=r}$	$\mathbf{f_{-1}=s}$	$\mathbf{NC=t}$	$\mathbf{NC_{-1}}$	$\hat{\Pr}(\hat{\mathbf{X}} \models \mathbf{p,q} \mid \hat{\mathbf{Y}} \models \mathbf{r,s,t})$
						λ^0
's	otw	mcdonald	icp	ORG	ORG	0.5
+end+	otw	's	otw	ORG	ORG	0.5
+end+	otw	.	otw	NAN	NAN	0.66667
+end+	otw	in	ucp	NAN	NAN	0.5
+end+	otw	jones	icp	PER	PER	0.66667
+end+	otw	mr.	fwd	NAN	NAN	0.66667
.	otw	apples	ucp	NAN	NAN	0.5
apples	ucp	eating	ucp	NAN	NAN	0.5
eating	ucp	is	ucp	NAN	NAN	0.5
in	ucp	eats	ucp	NAN	NAN	0.5
jones	icp	t.	cpp	PER	PER	0.5

$\mathbf{w=p}$	$\mathbf{f=q}$	$\mathbf{w_{-1}}$	$\mathbf{f_{-1}}$	$\mathbf{NC=t}$	$\mathbf{NC_{-1}}$	$\hat{\Pr}(\hat{\mathbf{X}} \models \mathbf{p,q} \mid \hat{\mathbf{Y}} \models \mathbf{t})$		
						λ^0	$\lambda_\mathbf{f}$	$\lambda_\mathbf{s}$
's	otw	mcdonald	icp	ORG	ORG	0.5	0.5	0.33333
+end+	otw	's	otw	ORG	ORG	0.5	0.5	0.33333
+end+	otw	.	otw	NAN	NAN	0.56	0.56	0.48
+end+	otw	in	ucp	NAN	NAN	0.56	0.56	0.52
+end+	otw	jones	icp	PER	PER	0.55556	0.55556	0.33333
+end+	otw	mr.	fwd	NAN	NAN	0.56	0.56	0.48
.	otw	+begin+	otw	NAN	ORG	0.56	0.36	0.56
.	otw	apples	ucp	NAN	NAN	0.56	0.56	0.52
apples	ucp	eating	ucp	NAN	NAN	0.56	0.56	0.52
eating	ucp	is	ucp	NAN	NAN	0.56	0.56	0.52
eats	ucp	+begin+	otw	NAN	PER	0.56	0.36	0.56
in	ucp	eats	ucp	NAN	NAN	0.56	0.56	0.52
is	ucp	+begin+	otw	NAN	PER	0.56	0.36	0.56
jones	icp	+begin+	otw	PER	NAN	0.55556	0.33333	0.55556
jones	icp	t.	cpp	PER	PER	0.55556	0.55556	0.44444
mcdonald	icp	+begin+	otw	ORG	NAN	0.5	0.33333	0.5
mr.	fwd	+begin+	otw	NAN	SOS	0.56	0.36	0.56
t.	cpp	+begin+	otw	PER	NAN	0.55556	0.33333	0.55556

$\mathbf{w=p}$	$\mathbf{f=q}$	$\mathbf{w_{-1}}$	$\mathbf{f_{-1}}$	$\mathbf{NC=t}$	$\mathbf{NC_{-1}}$	$\hat{\Pr}(\hat{\mathbf{X}} \models \mathbf{p} \mid \hat{\mathbf{Y}} \models \mathbf{t})$		$\hat{\Pr}(\hat{\mathbf{X}} \models \mathbf{q} \mid \hat{\mathbf{Y}} \models \mathbf{t})$	
						λ^0	λ	λ^0	λ
's	otw	mcdonald	icp	ORG	ORG	0.5	0	0.5	0.25
+end+	otw	's	otw	ORG	ORG	0.5	0	0.5	0.25
+end+	otw	.	otw	NAN	NAN	0.625	0	0.58333	0.16667
+end+	otw	in	ucp	NAN	NAN	0.625	0	0.58333	0.16667
+end+	otw	jones	icp	PER	PER	0.66667	0	0.66667	0
+end+	otw	mr.	fwd	NAN	NAN	0.625	0	0.58333	0.16667
.	otw	+begin+	otw	NAN	ORG	0.5	0	0.58333	0.41667
.	otw	apples	ucp	NAN	NAN	0.5	0	0.58333	0.41667
apples	ucp	eating	ucp	NAN	NAN	0.5	0	0.5	0.4
eating	ucp	is	ucp	NAN	NAN	0.5	0	0.5	0.4
eats	ucp	+begin+	otw	NAN	PER	0.5	0	0.5	0.4
in	ucp	eats	ucp	NAN	NAN	0.5	0	0.5	0.4
is	ucp	+begin+	otw	NAN	PER	0.5	0	0.5	0.4
jones	icp	+begin+	otw	PER	NAN	0.5	0	0.5	0
jones	icp	t.	cpp	PER	PER	0.5	0	0.5	0
mcdonald	icp	+begin+	otw	ORG	NAN	0.5	0	0.5	0
mr.	fwd	+begin+	otw	NAN	SOS	0.66667	0	0.66667	0
t.	cpp	+begin+	otw	PER	NAN	0.5	0	0.5	0

Table A.16: Final sample probabilities for name-class generation.

NC	NC_{-1}	w_{-1}	$\hat{\Pr}(NC\|NC_{-1}, w_{-1})$	$\hat{\Pr}(NC\|NC_{-1})$	$\hat{\Pr}(NC)$	$\frac{1}{\|\mathcal{D}_{NC}\|}$
EOS	NAN	.	0.76549	0.33789	0.21875	0.25
NAN	ORG	's	0.69531	0.69531	0.40625	0.25
NAN	PER	jones	0.79167	0.79167	0.40625	0.25
NAN	SOS	+end+	0.79167	0.79167	0.40625	0.25
ORG	NAN	in	0.60078	0.20117	0.15625	0.25
PER	NAN	mr.	0.76549	0.33789	0.21875	0.25

Table A.17: Final sample probabilities for first words generation.

w	f	w_{-1}	f_{-1}	NC	NC_{-1}	$\tilde{\Pr}(<w,f>_{first} \mid NC, NC_{-1})$	$\tilde{\Pr}(<w,f> \mid <+begin+, otw>, NC)$	$\tilde{\Pr}(<w,f>\|NC)$	$\frac{\hat{\Pr}(w\|NC)}{\tilde{\Pr}(f\|NC)} \cdot \frac{1}{\|\mathcal{D}_w\|\|\mathcal{D}_f\|}$
's	otw	mcdonald	icp	ORG	ORG	0.16816	0.16816	0.16816	0.05979 0.01667
+end+	otw	's	otw	ORG	ORG	0.16816	0.16816	0.16816	0.05979 0.01667
+end+	otw	.	otw	NAN	NAN	0.20071	0.20071	0.20071	0.06852 0.01667
+end+	otw	in	ucp	NAN	NAN	0.22235	0.22235	0.22235	0.07410 0.01667
+end+	otw	jones	icp	PER	PER	0.55938	0.13500	0.0816	0.02382 0.01667
+end+	otw	mr.	fwd	NAN	NAN	0.0816	0.0816	0.0816	0.02382 0.01667
.	otw	+begin+	otw	NAN	ORG	0.04112	0.04112	0.04112	0.00823 0.01667
.	otw	apples	ucp	NAN	NAN	0.04112	0.04112	0.04112	0.00823 0.01667
apples	ucp	eating	ucp	NAN	NAN	0.29245	0.12326	0.04112	0.00823 0.01667
eating	ucp	is	ucp	NAN	NAN	0.04112	0.04112	0.04112	0.00823 0.01667
eats	ucp	+begin+	otw	NAN	PER	0.29245	0.12326	0.04112	0.00823 0.01667
in	ucp	eats	ucp	NAN	NAN	0.31676	0.31676	0.22235	0.04340 0.01667
is	ucp	+begin+	otw	NAN	PER	0.22235	0.22235	0.22235	0.04340 0.01667
jones	icp	+begin+	otw	PER	NAN	0.55565	0.55565	0.16681	0.03063 0.01667
jones	icp	t.	cpp	PER	PER	0.72258	0.24516	0.08012	0.01016 0.01667
mcdonald	icp	+begin+	otw	ORG	NAN	0.28343	0.28343	0.11123	0.01174 0.01667
mr.	fwd	+begin+	otw	NAN	SOS	0.08012	0.08012	0.10079	0.10079 0.01667
t.	cpp	+begin+	otw	PER	NAN	0.11123	0.11123	0.10833	0.10833 0.01667

Table A.18: Final sample probabilities for subsequent words generation.

w	f	w_{-1}	f_{-1}	NC	NC_{-1}	$\tilde{\Pr}(<w,f> \mid <w,f>_{-1}, NC)$	$\tilde{\Pr}(<w,f>\|NC)$	$\frac{\hat{\Pr}(w\|NC)}{\tilde{\Pr}(f\|NC)} \cdot \frac{1}{\|\mathcal{D}_w\|\|\mathcal{D}_f\|}$
's	otw	mcdonald	icp	ORG	ORG	0.55655	0.16816	0.05979 0.01667
+end+	otw	's	otw	ORG	ORG	0.55655	0.16816	0.05979 0.01667
+end+	otw	.	otw	NAN	NAN	0.72409	0.20071	0.06852 0.01667
+end+	otw	in	ucp	NAN	NAN	0.59325	0.20071	0.06852 0.01667
+end+	otw	jones	icp	PER	PER	0.71117	0.22235	0.07410 0.01667
+end+	otw	mr.	fwd	NAN	NAN	0.72409	0.20071	0.06852 0.01667
.	otw	+begin+	otw	NAN	ORG	0.0816	0.0816	0.02382 0.01667
.	otw	apples	ucp	NAN	NAN	0.53802	0.0816	0.02382 0.01667
apples	ucp	eating	ucp	NAN	NAN	0.51918	0.04112	0.00823 0.01667
eating	ucp	is	ucp	NAN	NAN	0.51918	0.04112	0.00823 0.01667
eats	ucp	+begin+	otw	NAN	PER	0.04112	0.04112	0.00823 0.01667
in	ucp	eats	ucp	NAN	NAN	0.51918	0.04112	0.00823 0.01667
is	ucp	+begin+	otw	NAN	PER	0.04112	0.04112	0.00823 0.01667
jones	icp	+begin+	otw	PER	NAN	0.22235	0.22235	0.04340 0.01667
jones	icp	t.	cpp	PER	PER	0.58897	0.22235	0.04340 0.01667
mcdonald	icp	+begin+	otw	ORG	NAN	0.16681	0.16681	0.03063 0.01667
mr.	fwd	+begin+	otw	NAN	SOS	0.08012	0.08012	0.01016 0.01667
t.	cpp	+begin+	otw	PER	NAN	0.11123	0.11123	0.01174 0.01667

Table A.19: Final sample probabilities for name-class generation considering unknown words.

NC	NC_{-1}	w_{-1}	$\tilde{Pr}(NC\|NC_{-1}, w_{-1})$	$\tilde{Pr}(NC\|NC_{-1})$	$\tilde{Pr}(NC)$	$\frac{1}{\|\mathcal{D}_{NC}\|}$
EOS	NAN	+UNK+	0.52432	0.34425	0.21552	0.25
EOS	NAN	.	0.77398	0.34425	0.21552	0.25
NAN	ORG	's	0.77694	0.70259	0.42241	0.25
NAN	ORG	+UNK+	0.77694	0.70259	0.42241	0.25
NAN	PER	jones	0.87759	0.87759	0.42241	0.25
NAN	SOS	+end+	0.87759	0.87759	0.42241	0.25
ORG	NAN	+UNK+	0.27963	0.19943	0.14655	0.25
ORG	NAN	in	0.59966	0.19943	0.14655	0.25
PER	NAN	mr.	0.85993	0.34425	0.21552	0.25

Table A.20: Final sample probabilities for first words generation considering unknown words.

w	f	NC	NC_{-1}	$\tilde{Pr}(<w,f>_{first} \mid NC, NC_{-1})$	$\tilde{Pr}(<w,f> \mid <+begin+, otw>, NC)$	$\tilde{Pr}(<w,f>\|NC)$	$\tilde{Pr}(w\|NC)\cdot \tilde{Pr}(f\|NC)$	$\frac{1}{\|\mathcal{D}_w\|\|\mathcal{D}_f\|}$
's	otw	ORG	ORG	0.08533	0.08533	0.08533	0.03104	0.01538
+end+	otw	ORG	ORG	0.16804	0.16804	0.16804	0.05946	0.01538
+end+	otw	NAN	NAN	0.22280	0.22280	0.22280	0.07549	0.01538
+end+	otw	PER	PER	0.25009	0.25009	0.25009	0.10438	0.01538
+UNK+	cpp	PER	NAN	0.15918	0.15918	0.06286	0.00621	0.01538
+UNK+	icp	ORG	NAN	0.28078	0.28078	0.08783	0.03040	0.01538
+UNK+	otw	NAN	ORG	0.28705	0.08058	0.05681	0.05111	0.01538
+UNK+	otw	ORG	ORG	0.09565	0.09565	0.09565	0.05946	0.01538
+UNK+	otw	NAN	NAN	0.05681	0.05681	0.05681	0.05111	0.01538
+UNK+	ucp	NAN	PER	0.33807	0.1533	0.11365	0.03564	0.01538
+UNK+	ucp	NAN	NAN	0.11365	0.11365	0.11365	0.03564	0.01538
.	otw	NAN	ORG	0.28276	0.07414	0.04600	0.01347	0.01538
.	otw	NAN	NAN	0.046	0.04600	0.04600	0.01347	0.01538
apples	ucp	NAN	NAN	0.02336	0.02336	0.02336	0.00553	0.01538
eating	ucp	NAN	NAN	0.02336	0.02336	0.02336	0.00553	0.01538
eats	ucp	NAN	PER	0.16324	0.06854	0.02336	0.00553	0.01538
in	ucp	NAN	NAN	0.02336	0.02336	0.02336	0.00553	0.01538
is	ucp	NAN	PER	0.16324	0.06854	0.02336	0.00553	0.01538
jones	icp	PER	NAN	0.35006	0.35006	0.25009	0.05530	0.01538
jones	icp	PER	PER	0.25009	0.25009	0.25008	0.05530	0.01538
mcdonald	icp	ORG	NAN	0.27826	0.27826	0.08406	0.01587	0.01538
mr.	fwd	NAN	SOS	0.83716	0.27148	0.08898	0.01377	0.01538
t.	cpp	PER	NAN	0.15918	0.15918	0.06285	0.00621	0.01538

Table A.21: Final sample probabilities for subsequent words generation considering unknown words.

w	f	w_{-1}	f_{-1}	NC	$\tilde{Pr}(<w,f> \mid <w,f>_{-1}, NC)$	$\tilde{Pr}(<w,f>\|NC)$	$\tilde{Pr}(w\|NC)\cdot \tilde{Pr}(f\|NC)$	$\frac{1}{\|\mathcal{D}_w\|\|\mathcal{D}_f\|}$
's	otw	mcdonald	icp	ORG	0.53589	0.08533	0.03104	0.01538
+end+	otw	's	otw	ORG	0.57024	0.16804	0.05946	0.01538
+end+	otw	+UNK+	otw	NAN	0.73567	0.2228	0.07549	0.01538
+end+	otw	+UNK+	otw	ORG	0.57024	0.16804	0.05946	0.01538
+end+	otw	+UNK+	ucp	NAN	0.19373	0.2228	0.07549	0.01538
+end+	otw	.	otw	NAN	0.73567	0.2228	0.07549	0.01538
+end+	otw	in	ucp	NAN	0.60745	0.2228	0.07549	0.01538
+end+	otw	jones	icp	PER	0.83003	0.25009	0.10438	0.01538
+end+	otw	mr.	fwd	NAN	0.83824	0.2228	0.07549	0.01538
+UNK+	cpp	+begin+	otw	PER	0.06286	0.06286	0.00621	0.01538
+UNK+	icp	+begin+	otw	ORG	0.08783	0.08783	0.0304	0.01538
+UNK+	otw	+begin+	otw	NAN	0.05681	0.05681	0.05111	0.01538
+UNK+	otw	+UNK+	icp	ORG	0.54191	0.09565	0.05946	0.01538
+UNK+	otw	+UNK+	ucp	NAN	0.14469	0.05681	0.05111	0.01538
+UNK+	ucp	+begin+	otw	NAN	0.11365	0.11365	0.03564	0.01538
+UNK+	ucp	+UNK+	ucp	NAN	0.41046	0.11365	0.03564	0.01538
.	otw	+begin+	otw	NAN	0.046	0.046	0.01347	0.01538
.	otw	apples	ucp	NAN	0.52225	0.046	0.01347	0.01538
apples	ucp	eating	ucp	NAN	0.51132	0.02336	0.00553	0.01538
eating	ucp	is	ucp	NAN	0.51132	0.02336	0.00553	0.01538
eats	ucp	+begin+	otw	NAN	0.02336	0.02336	0.00553	0.01538
in	ucp	eats	ucp	NAN	0.51132	0.02336	0.00553	0.01538
is	ucp	+begin+	otw	NAN	0.02336	0.02336	0.00553	0.01538
jones	icp	+begin+	otw	PER	0.25009	0.25009	0.0553	0.01538
jones	icp	+UNK+	cpp	PER	0.61255	0.25009	0.0553	0.01538
jones	icp	t.	cpp	PER	0.61255	0.25009	0.0553	0.01538
mcdonald	icp	+begin+	otw	ORG	0.08406	0.08406	0.01587	0.01538
mr.	fwd	+begin+	otw	NAN	0.08898	0.08898	0.01377	0.01538
t.	cpp	+begin+	otw	PER	0.06286	0.06286	0.00621	0.01538

A.8 Decoding: a walk-thorugh example

The first step for applying siNymble is to prepare the sequences of decoding events —representing sentences— to be presented to the HMM. The example sequence is shown in table A.2 on page 183.

Bikel et al. (1997) state they use the Viterbi algorithm (Viterbi 1967, Rabiner 1989, Durbin, Eddy, Krogh and Mitchison 1998) for decoding, though they do not provide any detail of how this was done. It is evident that the standard Viterbi algorithm is not applicable as probabilities for transitions and emissions are mixed in the n-gram language model.

An appropriate version of this algorithm has been created for siNymble. Basically, the program keeps one possible path which finalises in a given state NC after the emission of the k-th event $<w,f>_k$ in the input sentence. That is, $|\mathcal{D}_{NC} - \{EOS\}|$ —the number of name-classes— different paths are kept at any step. Each of them has associated a probability $\delta_k(j)$ where j iterates over the name-classes. When all the words in the sequence has been emitted, the highest δ determines the most likely path for that sequence. Algorithm A.1 illustrates this procedure in detail. For a complete understanding of this algorithm, the following clarifications are needed:

- ▷ the special states for the beginning and end of sentences are not considered as name-classes because it is known *a priori* that the probability of making a transition to these states in the middle of the sentence is zero; they are consequently considered during the initialisation and termination of the algorithm
- ▷ w_k denotes the word only —ignoring the lexical feature f— of the k-th event; thus w_{-1} in siNymble models becomes $w_{(k-1)}$ for this event
- ▷ the function model(w, w_{-1}) determines whether the algorithm should use the unknown words model or the model from normal training; that is

$$\text{model}(w, w_{-1}) = \begin{cases} \textbf{normal training top-level model} & \textbf{if } w = +UNK+ \textbf{ or } w_{-1} = +UNK+ \\ \textbf{unknown words top-level model} & \textbf{if both words are known} \end{cases}$$

- ▷ $\hat{Pr}_M^C$ denotes the final sample probabilities of a top-level component C given the modality M determined by the function model(w, w_{-1})

The following is the ouput produced by algorithm A.1 for the sample decoding sentence.

INITIALISATION

<mr.,fwd>

Algorithm A.1: The Viterbi algorithm used by siNymble.

Input: $E = \{<w, f>_0, <w, f>_1, \ldots, <w, f>_n\}$, the events for a sentence
Output: q^*, the indices for the most likely name-class path $\{NC_{q_1^*}, NC_{q_2^*}, \ldots, NC_{q_n^*}\}$

procedure VITERBI(E)
  Initialisation:
    $M \leftarrow \text{model}(w_0, +begin+)$
    for each name-class $j = 1, 2, \ldots, |\mathcal{D}_{NC} - \{EOS\}|$ **do**
      $\delta_0(j) \leftarrow \hat{\Pr}_M^{NC}(NC_j|SOS, +begin+) \cdot \hat{\Pr}_M^{FW}(<w, f>_0 | NC_j, SOS)$
      $\psi_0(j) \leftarrow j$
    end for each
  Recursion:
    for each event $<w, f>_k \in E$, $k = 1, 2, \ldots, n$ **do**
      $M \leftarrow \text{model}(w_k, w_{(k-1)})$
      for each name-class $j = 1, 2, \ldots, |\mathcal{D}_{NC} - \{EOS\}|$ **do**
        for each name-class $i = 1, 2, \ldots, |\mathcal{D}_{NC} - \{EOS\}|$ **do**
          if $j = i$ **then**
            $p \leftarrow \hat{\Pr}_M^{SW}(<w, f>_k \mid <w, f>_{(k-1)}, NC_j)$
            $p' \leftarrow \hat{\Pr}_M^{SW}(<+end+, otw> \mid <w, f>_{(k-1)}, NC_j) \cdot \hat{\Pr}_M^{NC}(NC_j|NC_j, w_{(k-1)}) \cdot \hat{\Pr}_M^{FW}(<w, f>_k | NC_j, NC_j)$
            $P_i \leftarrow \max(p, p')$
          else
            $P_i \leftarrow \hat{\Pr}_M^{SW}(<+end+, otw> \mid <w, f>_{(k-1)}, NC_i) \cdot \hat{\Pr}_M^{NC}(NC_j|NC_i, w_{(k-1)}) \cdot \hat{\Pr}_M^{FW}(<w, f>_k | NC_j, NC_i)$
          end if
        end for each
        $\psi_k(j) \leftarrow \underset{i}{\text{argmax}}\, P_i$
        $\delta_k(j) \leftarrow \delta_{(k-1)}(j) \cdot P_{\psi_k(j)}$
      end for each
    end for each
  Termination:
    $M \leftarrow \text{model}(+end+, w_n)$
    for each name-class $j = 1, 2, \ldots, |\mathcal{D}_{NC} - \{EOS\}|$ **do**
      $P_f(j) \leftarrow \hat{\Pr}_M^{SW}(<+end+, otw> \mid <w, f>_n, NC_j) \cdot \hat{\Pr}_M^{NC}(EOS|NC_j, w_n)$
    end for each
    $q_n^* \leftarrow \underset{j}{\text{argmax}}\, P_f(j)$
  Best sequence:
    $q_k^* \leftarrow \psi_{k+1}(q_{(k+1)}^*), \quad k = n-1, n-2, \ldots, 1$
end procedure

[ncModel] Pr(NC=NAN|pre-NC=SOS,pre-w=+end+)=0.791666666666667
[fwModel] Pr(<w,f>=<mr.,fwd>|NC=NAN,pre-NC=SOS)=0.722579188712522
log Delta(0, SOS, NAN)=0.5720418577307468
[ncModel] Pr(NC=ORG|pre-NC=SOS,pre-w=+end+) ~ Pr(NC=ORG)=0.15625
[fwModel] Pr(<w,f>=<mr.,fwd>|NC=ORG,pre-NC=SOS) ~ Pr(default)=0.0166666666666667
log Delta(0, SOS, ORG)=0.0026041666666666713
[ncModel] Pr(NC=PER|pre-NC=SOS,pre-w=+end+) ~ Pr(NC=PER)=0.21875
[fwModel] Pr(<w,f>=<mr.,fwd>|NC=PER,pre-NC=SOS) ~ Pr(default)=0.0166666666666667
log Delta(0, SOS, PER)=0.00364583333333334

log Delta(0, NAN)=0.5720418577307468
log Delta(0, ORG)=0.0026041666666666713
log Delta(0, PER)=0.00364583333333334

ITERATION

<jones,icp>

[swModel] Pr(<w,f>=<jones,icp>|pre-<w,f>=<mr.,fwd>,NC=NAN) ~ Pr(default)=0.0166666666666667
[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=NAN)=0.724090388007055
[ncModel] Pr(NC=NAN|pre-NC=NAN,pre-w=mr.) ~ Pr(NC=NAN)=0.40625
[fwModel] Pr(<w,f>=<jones,icp>|NC=NAN,pre-NC=NAN) ~ Pr(default)=0.0166666666666667
log Delta(1,NAN,NAN)=0.00953403096217913

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168159722222222
[ncModel] Pr(NC=NAN|pre-NC=ORG,pre-w=mr.) ~ Pr(NC=NAN|pre-NC=ORG)=0.6953125
[fwModel] Pr(<w,f>=<jones,icp>|NC=NAN,pre-NC=ORG) ~ Pr(default)=0.0166666666666667
log Delta(1,ORG,NAN)=5.0748071552794776E-6

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=PER) ~ Pr(<w,f>=<+end+,otw>|
NC=PER)=0.222345679012346
[ncModel] Pr(NC=NAN|pre-NC=PER,pre-w=mr.) ~ Pr(NC=NAN|pre-NC=PER)=0.791666666666667
[fwModel] Pr(<w,f>=<jones,icp>|NC=NAN,pre-NC=PER) ~ Pr(default)=0.0166666666666667
log Delta(1,PER,NAN)=1.0695882273091053E-5

log Delta(1,NAN) = log Delta(1,NAN,NAN) = 0.00953403096217913

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=NAN)=0.724090388007055
[ncModel] Pr(NC=ORG|pre-NC=NAN,pre-w=mr.) ~ Pr(NC=ORG|pre-NC=NAN)=0.201171875
[fwModel] Pr(<w,f>=<jones,icp>|NC=ORG,pre-NC=NAN) ~ Pr(default)=0.0166666666666667
log Delta(1,NAN,ORG)=0.0013887900750069976

[swModel] Pr(<w,f>=<jones,icp>|pre-<w,f>=<mr.,fwd>,NC=ORG) ~ Pr(default)=0.0166666666666667
[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168159722222222
[ncModel] Pr(NC=ORG|pre-NC=ORG,pre-w=mr.) ~ Pr(NC=ORG)=0.15625
[fwModel] Pr(<w,f>=<jones,icp>|NC=ORG,pre-NC=ORG) ~ Pr(default)=0.0166666666666667
log Delta(1,ORG,ORG)=4.340277777777779E-5 (I tag)

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=PER) ~ Pr(<w,f>=<+end+,otw>|
NC=PER)=0.222345679012346
[ncModel] Pr(NC=ORG|pre-NC=PER,pre-w=mr.) ~ Pr(NC=ORG)=0.15625
[fwModel] Pr(<w,f>=<jones,icp>|NC=ORG,pre-NC=PER) ~ Pr(default)=0.0166666666666667
log Delta(1,PER,ORG)=2.1110293960048104E-6

log Delta(1,ORG) = log Delta(1,NAN,ORG) = 0.0013887900750069976

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=NAN)=0.724090388007055
[ncModel] Pr(NC=PER|pre-NC=NAN,pre-w=mr.)=0.765494791666667
[fwModel] Pr(<w,f>=<jones,icp>|NC=PER,pre-NC=NAN)=0.316759259259259
log Delta(1,NAN,PER)=0.10043663404226938

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168159722222222
[ncModel] Pr(NC=PER|pre-NC=ORG,pre-w=mr.) ~ Pr(NC=PER)=0.21875
[fwModel] Pr(<w,f>=<jones,icp>|NC=PER,pre-NC=ORG) ~ Pr(<w,f>=<jones,icp>|

pre-<w,f>=<+begin+,otw>,NC=PER)=0.316759259259259
log Delta(1,ORG,PER)=3.0343672146685987E-5

[swModel] Pr(<w,f>=<jones,icp>|pre-<w,f>=<mr.,fwd>,NC=PER) ~ Pr(<w,f>=<jones,icp>|
NC=PER)=0.222345679012346
[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<mr.,fwd>,NC=PER) ~ Pr(<w,f>=<+end+,otw>|
NC=PER)=0.222345679012346
[ncModel] Pr(NC=PER|pre-NC=PER,pre-w=mr.) ~ Pr(NC=PER)=0.21875
[fwModel] Pr(<w,f>=<jones,icp>|NC=PER,pre-NC=PER)=0.222345679012346
log Delta(1,PER,PER)=8.106352880658458E-4 (I tag)

log Delta(1,PER) = log Delta(1,NAN,PER) = 0.10043663404226938

log Delta(1, NAN)=0.00953403096217913
log Delta(1, ORG)=0.0013887900750069976
log Delta(1, PER)=0.10043663404226938

<eats,ucp>

[swModel] Pr(<w,f>=<eats,ucp>|pre-<w,f>=<jones,icp>,NC=NAN) ~ Pr(<w,f>=<eats,ucp>|
NC=NAN)=0.0411209523809524
[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=NAN) ~ Pr(<w,f>=<+end+,otw>|
NC=NAN)=0.200712962962963
[ncModel] Pr(NC=NAN|pre-NC=NAN,pre-w=jones) ~ Pr(NC=NAN)=0.40625
[fwModel] Pr(<w,f>=<eats,ucp>|NC=NAN,pre-NC=NAN) ~ Pr(<w,f>=<eats,ucp>|
pre-<w,f>=<+begin+,otw>,NC=NAN)=0.123264338624339
log Delta(2,NAN,NAN)=3.920484331942936E-4

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168159722222222
[ncModel] Pr(NC=NAN|pre-NC=ORG,pre-w=jones) ~ Pr(NC=NAN|pre-NC=ORG)=0.6953125
[fwModel] Pr(<w,f>=<eats,ucp>|NC=NAN,pre-NC=ORG) ~ Pr(<w,f>=<eats,ucp>|
pre-<w,f>=<+begin+,otw>,NC=NAN)=0.123264338624339
log Delta(2,ORG,NAN)=2.001594376897324E-5

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=PER)=0.711172839506173
[ncModel] Pr(NC=NAN|pre-NC=PER,pre-w=jones)=0.791666666666667
[fwModel] Pr(<w,f>=<eats,ucp>|NC=NAN,pre-NC=PER)=0.292448253968254
log Delta(2,PER,NAN)=0.01653707529480418

log Delta(2,NAN) = log Delta(2,PER,NAN) = 0.01653707529480418

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=NAN) ~ Pr(<w,f>=<+end+,otw>|
NC=NAN)=0.200712962962963
[ncModel] Pr(NC=ORG|pre-NC=NAN,pre-w=jones) ~ Pr(NC=ORG|pre-NC=NAN)=0.201171875
[fwModel] Pr(<w,f>=<eats,ucp>|NC=ORG,pre-NC=NAN) ~ Pr(default)=0.0166666666666667
log Delta(2,NAN,ORG)=6.416053748377586E-6

[swModel] Pr(<w,f>=<eats,ucp>|pre-<w,f>=<jones,icp>,NC=ORG) ~ Pr(default)=0.0166666666666667
[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168159722222222
[ncModel] Pr(NC=ORG|pre-NC=ORG,pre-w=jones) ~ Pr(NC=ORG)=0.15625

[fwModel] Pr(<w,f>=<eats,ucp>|NC=ORG,pre-NC=ORG) ~ Pr(default)=0.0166666666666667
log Delta(2,ORG,ORG)=2.3146501250116655E-5 (I tag)

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=PER)=0.711172839506173
[ncModel] Pr(NC=ORG|pre-NC=PER,pre-w=jones) ~ Pr(NC=ORG)=0.15625
[fwModel] Pr(<w,f>=<eats,ucp>|NC=ORG,pre-NC=PER) ~ Pr(default)=0.0166666666666667
log Delta(2,PER,ORG)=1.8600991203719593E-4

log Delta(2,ORG) = log Delta(2,PER,ORG) = 1.8600991203719593E-4

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=NAN) ~ Pr(<w,f>=<+end+,otw>|
NC=NAN)=0.200712962962963
[ncModel] Pr(NC=PER|pre-NC=NAN,pre-w=jones) ~ Pr(NC=PER|pre-NC=NAN)=0.337890625
[fwModel] Pr(<w,f>=<eats,ucp>|NC=PER,pre-NC=NAN) ~ Pr(default)=0.0166666666666667
log Delta(2,NAN,PER)=1.0776478625915744E-5

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168159722222222
[ncModel] Pr(NC=PER|pre-NC=ORG,pre-w=jones) ~ Pr(NC=PER)=0.21875
[fwModel] Pr(<w,f>=<eats,ucp>|NC=PER,pre-NC=ORG) ~ Pr(default)=0.0166666666666667
log Delta(2,ORG,PER)=8.514426420141109E-7

[swModel] Pr(<w,f>=<eats,ucp>|pre-<w,f>=<jones,icp>,NC=PER) ~ Pr(default)=0.0166666666666667
[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<jones,icp>,NC=PER)=0.711172839506173
[ncModel] Pr(NC=PER|pre-NC=PER,pre-w=jones) ~ Pr(NC=PER)=0.21875
[fwModel] Pr(<w,f>=<eats,ucp>|NC=PER,pre-NC=PER) ~ Pr(default)=0.0166666666666667
log Delta(2,PER,PER)=0.0016739439007044932 (I tag)

log Delta(2,PER) = log Delta(2,PER,PER) = 0.0016739439007044932

log Delta(2, NAN)=0.01653707529480418
log Delta(2, ORG)=1.8600991203719593E-4
log Delta(2, PER)=0.0016739439007044932

<+UNK+,ucp> (bananas)

[unk_swModel] Pr(<w,f>=<+UNK+,ucp>|pre-<w,f>=<eats,ucp>,NC=NAN) ~
Pr(<w,f>=<+UNK+,ucp>|NC=NAN)=0.113652484267869
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=NAN) ~
Pr(<w,f>=<+end+,otw>|NC=NAN)=0.222801475002144
[unk_ncModel] Pr(NC=NAN|pre-NC=NAN,pre-w=eats) ~ Pr(NC=NAN)=0.422413793103448
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=NAN,pre-NC=NAN)=0.113652484267869
log Delta(3,NAN,NAN)=0.001879479689779297

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=NAN|pre-NC=ORG,pre-w=eats) ~ Pr(NC=NAN|pre-NC=ORG)=0.702586206896552
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=NAN,pre-NC=ORG) ~ Pr(<w,f>=<+UNK+,ucp>|pre-
<w,f>=<+begin+,otw>,NC=NAN)=0.153299326239371
log Delta(3,ORG,NAN)=3.3665269922629046E-6

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645

[unk_ncModel] Pr(NC=NAN|pre-NC=PER,pre-w=eats) ~ Pr(NC=NAN|pre-NC=PER)=0.877586206896552
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=NAN,pre-NC=PER)=0.338070947313837
log Delta(3,PER,NAN)=1.2420317771032666E-4

log Delta(3,NAN) = log Delta(3,NAN,NAN) = 0.001879479689779297

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=NAN) ~
Pr(<w,f>=<+end+,otw>|NC=NAN)=0.222801475002144
[unk_ncModel] Pr(NC=ORG|pre-NC=NAN,pre-w=eats) ~ Pr(NC=ORG|pre-NC=NAN)=0.199425287356322
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=ORG,pre-NC=NAN) ~ Pr(default)=0.0153846153846154
log Delta(3,NAN,ORG)=1.1304298978449625E-5

[unk_swModel] Pr(<w,f>=<+UNK+,ucp>|pre-<w,f>=<eats,ucp>,NC=ORG) ~
Pr(default)=0.0153846153846154
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=ORG|pre-NC=ORG,pre-w=eats) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=ORG,pre-NC=ORG) ~ Pr(default)=0.0153846153846154
log Delta(3,ORG,ORG)=2.861690954418402E-6 (I tag)

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=ORG|pre-NC=PER,pre-w=eats) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=ORG,pre-NC=PER) ~ Pr(default)=0.0153846153846154
log Delta(3,PER,ORG)=9.438710015033703E-7

log Delta(3,ORG) = log Delta(3,NAN,ORG) = 1.1304298978449625E-5

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=NAN) ~
Pr(<w,f>=<+end+,otw>|NC=NAN)=0.222801475002144
[unk_ncModel] Pr(NC=PER|pre-NC=NAN,pre-w=eats) ~ Pr(NC=PER|pre-NC=NAN)=0.344252873563218
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=PER,pre-NC=NAN) ~ Pr(default)=0.0153846153846154
log Delta(3,NAN,PER)=1.9513761060781844E-5

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=PER|pre-NC=ORG,pre-w=eats) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=PER,pre-NC=ORG) ~ Pr(default)=0.0153846153846154
log Delta(3,ORG,PER)=1.0363606103417495E-7

[unk_swModel] Pr(<w,f>=<+UNK+,ucp>|pre-<w,f>=<eats,ucp>,NC=PER) ~
Pr(default)=0.0153846153846154
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<eats,ucp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=PER|pre-NC=PER,pre-w=eats) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,ucp>|NC=PER,pre-NC=PER) ~ Pr(default)=0.0153846153846154
log Delta(3,PER,PER)=2.5752983087761442E-5 (I tag)

log Delta(3,PER) = log Delta(3,PER,PER) = 2.5752983087761442E-5

log Delta(3, NAN)=0.001879479689779297
log Delta(3, ORG)=1.1304298978449625E-5
log Delta(3, PER)=2.5752983087761442E-5

<in,ucp>

[unk_swModel] Pr(<w,f>=<in,ucp>|pre-<w,f>=<+UNK+,ucp>,NC=NAN) ~
Pr(<w,f>=<in,ucp>|NC=NAN)=0.0233632009016624
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=NAN)=0.193733488919108
[unk_ncModel] Pr(NC=NAN|pre-NC=NAN,pre-w=+UNK+) ~ Pr(NC=NAN)=0.422413793103448
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=NAN,pre-NC=NAN)=0.0233632009016624
log Delta(4,NAN,NAN)=4.391066158290782E-5

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=NAN|pre-NC=ORG,pre-w=+UNK+)=0.776939655172414
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=NAN,pre-NC=ORG) ~ Pr(<w,f>=<in,ucp>|
NC=NAN)=0.0233632009016624
log Delta(4,ORG,NAN)=3.448017133816848E-8

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=NAN|pre-NC=PER,pre-w=+UNK+) ~ Pr(NC=NAN|
pre-NC=PER)=0.877586206896552
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=NAN,pre-NC=PER) ~
Pr(<w,f>=<in,ucp>|NC=NAN)=0.0233632009016624
log Delta(4,PER,NAN)=1.32051653496337E-7

log Delta(4,NAN) = log Delta(4,NAN,NAN) = 4.391066158290782E-5

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=NAN)=0.193733488919108
[unk_ncModel] Pr(NC=ORG|pre-NC=NAN,pre-w=+UNK+)=0.279632183908046
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=ORG,pre-NC=NAN) ~ Pr(default)=0.0153846153846154
log Delta(4,NAN,ORG)=1.566448548080543E-6

[unk_swModel] Pr(<w,f>=<in,ucp>|pre-<w,f>=<+UNK+,ucp>,NC=ORG) ~
Pr(default)=0.0153846153846154
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=ORG|pre-NC=ORG,pre-w=+UNK+) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=ORG,pre-NC=ORG) ~ Pr(default)=0.0153846153846154
log Delta(4,ORG,ORG)=1.739122919761483E-7 (I tag)

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=ORG|pre-NC=PER,pre-w=+UNK+) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=ORG,pre-NC=PER) ~ Pr(default)=0.0153846153846154
log Delta(4,PER,ORG)=1.4521092330821073E-8

log Delta(4,ORG) = log Delta(4,NAN,ORG) = 1.566448548080543E-6

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=NAN)=0.193733488919108
[unk_ncModel] Pr(NC=PER|pre-NC=NAN,pre-w=+UNK+) ~ Pr(NC=PER|
pre-NC=NAN)=0.344252873563218
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=PER,pre-NC=NAN) ~ Pr(default)=0.0153846153846154
log Delta(4,NAN,PER)=1.9284418782888972E-6

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=PER|pre-NC=ORG,pre-w=+UNK+) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=PER,pre-NC=ORG) ~ Pr(default)=0.0153846153846154
log Delta(4,ORG,PER)=6.2982289817163E-9

[unk_swModel] Pr(<w,f>=<in,ucp>|pre-<w,f>=<+UNK+,ucp>,NC=PER) ~
Pr(default)=0.0153846153846154
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,ucp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=PER|pre-NC=PER,pre-w=+UNK+) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<in,ucp>|NC=PER,pre-NC=PER) ~ Pr(default)=0.0153846153846154
log Delta(4,PER,PER)=3.96199739811715E-7 (I tag)

log Delta(4,PER) = log Delta(4,NAN,PER) = 1.9284418782888972E-6

log Delta(4, NAN)=4.391066158290782E-5
log Delta(4, ORG)=1.566448548080543E-6
log Delta(4, PER)=1.9284418782888972E-6

<+UNK+,icp> (starebucks)

[unk_swModel] Pr(<w,f>=<+UNK+,icp>|pre-<w,f>=<in,ucp>,NC=NAN) ~
Pr(default)=0.0153846153846154
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=NAN)=0.607449520379286
[unk_ncModel] Pr(NC=NAN|pre-NC=NAN,pre-w=in) ~ Pr(NC=NAN)=0.422413793103448
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=NAN,pre-NC=NAN) ~ Pr(default)=0.0153846153846154
log Delta(5,NAN,NAN)=6.755486397370441E-7

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=NAN|pre-NC=ORG,pre-w=in) ~ Pr(NC=NAN|pre-NC=ORG)=0.702586206896552
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=NAN,pre-NC=ORG) ~ Pr(default)=0.0153846153846154
log Delta(5,ORG,NAN)=2.8451721261433814E-9

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=NAN|pre-NC=PER,pre-w=in) ~ Pr(NC=NAN|pre-NC=PER)=0.877586206896552
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=NAN,pre-NC=PER) ~ Pr(default)=0.0153846153846154
log Delta(5,PER,NAN)=6.511441677022464E-9

log Delta(5,NAN) = log Delta(5,NAN,NAN) = 6.755486397370441E-7

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=NAN)=0.607449520379286
[unk_ncModel] Pr(NC=ORG|pre-NC=NAN,pre-w=in)=0.599655172413793
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=ORG,pre-NC=NAN)=0.280777996931843
log Delta(5,NAN,ORG)=4.491018349709104E-6

[unk_swModel] Pr(<w,f>=<+UNK+,icp>|pre-<w,f>=<in,ucp>,NC=ORG) ~ Pr(<w,f>=<+UNK+,icp>|
NC=ORG)=0.0878336620644313
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168037475345168

[unk_ncModel] Pr(NC=ORG|pre-NC=ORG,pre-w=in) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=ORG,pre-NC=ORG) ~ Pr(<w,f>=<+UNK+,icp>|
pre-<w,f>=<+begin+,otw>,NC=ORG)=0.280777996931843
log Delta(5,ORG,ORG)=1.375869124134256E-7 (I tag)

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=PER) ~ Pr(<w,f>=<+end+,otw>|
NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=ORG|pre-NC=PER,pre-w=in) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=ORG,pre-NC=PER) ~ Pr(<w,f>=<+UNK+,icp>|
pre-<w,f>=<+begin+,otw>,NC=ORG)=0.280777996931843
log Delta(5,PER,ORG)=1.98451655608078E-8

log Delta(5,ORG) = log Delta(5,NAN,ORG) = 4.491018349709104E-6

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=NAN)=0.607449520379286
[unk_ncModel] Pr(NC=PER|pre-NC=NAN,pre-w=in) ~ Pr(NC=PER|pre-NC=NAN)=0.344252873563218
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=PER,pre-NC=NAN) ~ Pr(w=+UNK+|NC=PER)*Pr(f=icp|
NC=PER)=0.013567202028740483
log Delta(5,NAN,PER)=1.2457991786051997E-7

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=PER|pre-NC=ORG,pre-w=in) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=PER,pre-NC=ORG) ~ Pr(w=+UNK+|NC=PER)*
Pr(f=icp|NC=PER)=0.013567202028740483
log Delta(5,ORG,PER)=7.696523397930446E-10

[unk_swModel] Pr(<w,f>=<+UNK+,icp>|pre-<w,f>=<in,ucp>,NC=PER) ~ Pr(w=+UNK+|NC=PER)*
Pr(f=icp|NC=PER)=0.013567202028740483
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<in,ucp>,NC=PER) ~ Pr(<w,f>=<+end+,otw>|
NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=PER|pre-NC=PER,pre-w=in) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<+UNK+,icp>|NC=PER,pre-NC=PER) ~ Pr(w=+UNK+|NC=PER)*
Pr(f=icp|NC=PER)=0.013567202028740483
log Delta(5,PER,PER)=2.6163560563429185E-8 (I tag)

log Delta(5,PER) = log Delta(5,NAN,PER) = 1.2457991786051997E-7

log Delta(5, NAN)=6.755486397370441E-7
log Delta(5, ORG)=4.491018349709104E-6
log Delta(5, PER)=1.2457991786051997E-7

<.,otw>

[unk_swModel] Pr(<w,f>=<.,otw>|pre-<w,f>=<+UNK+,icp>,NC=NAN) ~ Pr(<w,f>=<.,otw>|
NC=NAN)=0.0460033730669182
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=NAN) ~
Pr(<w,f>=<+end+,otw>|NC=NAN)=0.222801475002144
[unk_ncModel] Pr(NC=NAN|pre-NC=NAN,pre-w=+UNK+) ~ Pr(NC=NAN)=0.422413793103448
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=NAN,pre-NC=NAN)=0.0460033730669182
log Delta(6,NAN,NAN)=3.107751609867233E-8

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=NAN|pre-NC=ORG,pre-w=+UNK+)=0.776939655172414
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=NAN,pre-NC=ORG)=0.282761843507663
log Delta(6,ORG,NAN)=1.657902820550323E-7

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=NAN|
pre-NC=PER,pre-w=+UNK+) ~ Pr(NC=NAN|pre-NC=PER)=0.877586206896552
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=NAN,pre-NC=PER) ~ Pr(<w,f>=<.,otw>|
pre-<w,f>=<+begin+,otw>,NC=NAN)=0.0741427652614944
log Delta(6,PER,NAN)=2.0272195103891604E-9

log Delta(6,NAN) = log Delta(6,ORG,NAN) = 1.657902820550323E-7

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=NAN) ~
Pr(<w,f>=<+end+,otw>|NC=NAN)=0.222801475002144
[unk_ncModel] Pr(NC=ORG|pre-NC=NAN,pre-w=+UNK+)=0.279632183908046
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=ORG,pre-NC=NAN) ~ Pr(default)=0.0153846153846154
log Delta(6,NAN,ORG)=6.475129869856067E-10

[unk_swModel] Pr(<w,f>=<.,otw>|pre-<w,f>=<+UNK+,icp>,NC=ORG) ~
Pr(default)=0.0153846153846154
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=ORG|pre-NC=ORG,pre-w=+UNK+) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=ORG,pre-NC=ORG) ~ Pr(default)=0.0153846153846154
log Delta(6,ORG,ORG)=6.909258999552476E-8 (I tag)

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=ORG|pre-NC=PER,pre-w=+UNK+) ~ Pr(NC=ORG)=0.146551724137931
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=ORG,pre-NC=PER) ~ Pr(default)=0.0153846153846154
log Delta(6,PER,ORG)=7.024570643540017E-11

log Delta(6,ORG) = log Delta(6,ORG,ORG) = 6.909258999552476E-8

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=NAN) ~
Pr(<w,f>=<+end+,otw>|NC=NAN)=0.222801475002144
[unk_ncModel] Pr(NC=PER|pre-NC=NAN,pre-w=+UNK+) ~ Pr(NC=PER|
pre-NC=NAN)=0.344252873563218
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=PER,pre-NC=NAN) ~ Pr(default)=0.0153846153846154
log Delta(6,NAN,PER)=7.971478937939354E-10

[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=ORG) ~
Pr(<w,f>=<+end+,otw>|NC=ORG)=0.168037475345168
[unk_ncModel] Pr(NC=PER|pre-NC=ORG,pre-w=+UNK+) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=PER,pre-NC=ORG) ~ Pr(default)=0.0153846153846154
log Delta(6,ORG,PER)=2.5021862904971494E-9

[unk_swModel] Pr(<w,f>=<.,otw>|pre-<w,f>=<+UNK+,icp>,NC=PER) ~
Pr(default)=0.0153846153846154
[unk_swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<+UNK+,icp>,NC=PER) ~
Pr(<w,f>=<+end+,otw>|NC=PER)=0.25008875739645
[unk_ncModel] Pr(NC=PER|pre-NC=PER,pre-w=+UNK+) ~ Pr(NC=PER)=0.21551724137931
[unk_fwModel] Pr(<w,f>=<.,otw>|NC=PER,pre-NC=PER) ~ Pr(default)=0.0153846153846154
log Delta(6,PER,PER)=1.916614120931079E-9 (I tag)

log Delta(6,PER) = log Delta(6,ORG,PER) = 2.5021862904971494E-9

log Delta(6, NAN)=1.657902820550323E-7
log Delta(6, ORG)=6.909258999552476E-8
log Delta(6, PER)=2.5021862904971494E-9

FINALISATION

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<.,otw>,NC=NAN)=0.724090388007055
[ncModel] Pr(NC=EOS|pre-NC=NAN,pre-w=.)=0.765494791666667
log Delta(7,EOS,NAN)=9.189546781994543E-8

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<.,otw>,NC=ORG) ~ Pr(<w,f>=<+end+,otw>|
NC=ORG)=0.168159722222222
[ncModel] Pr(NC=EOS|pre-NC=ORG,pre-w=.) ~ Pr(NC=EOS)=0.21875
log Delta(7,EOS,ORG)=2.541566724650912E-9

[swModel] Pr(<w,f>=<+end+,otw>|pre-<w,f>=<.,otw>,NC=PER) ~ Pr(<w,f>=<+end+,otw>|
NC=PER)=0.222345679012346
[ncModel] Pr(NC=EOS|pre-NC=PER,pre-w=.) ~ Pr(NC=EOS)=0.21875
log Delta(7,EOS,PER)=1.217016302634939E-10

log P(SOS NAN PER NAN NAN NAN ORG NAN EOS)=9.189546781994543E-8 [*BEST*]
log P(SOS NAN PER NAN NAN NAN ORG ORG EOS)=2.541566724650912E-9
log P(SOS NAN PER NAN NAN NAN PER PER EOS)=1.217016302634939E-10

A.9 Implementation

The baseline system presented in this appendix is fully implemented in Java™. SiNymble models are stored in a relational database which allows an efficient way of searching probabilities during decoding —with full or partial information— given that the appropriate indexes on the tables have been set. Moreover, a relational database provides elegant implementations for the $\hat{c}()$ and $\hat{u}()$ function as queries of the form SELECT COUNT ... GROUP BY from a table that contains all training events or DISTINCT events only.

Appendix B

A walk-through example for LexMENE

In this appendix, a walk-through example is presented to show how LexMENE is trained and applied on unseen documents. This examples are taken from the MUC-7 training and dryrun test corpora.

B.1 Training

Consider the following sentence which is part of the training input for LexMENE.

> <TIME>Last night</TIME>'s crash came just hours after <LOCATION>St. Louis</LOCATION>-based <ORGANISATION>TWA</ORGANISATION> reported a fivefold increase in <DATE>second-quarter</DATE> profit.

The sentence must be tokenized first. This tokenization needs to be fine-grain because words can partially be part of a named entity. In this scheme, a *word* contains *tokens* which may atomically belong to a particular named entity class. In the text above, the words *tonight's* and *Louis-based* are example of multi-token words. In both cases, only the first token has a named entity class associated (namely time and organisation respectively). Consequently, each token is marked *%FT*, *%MT* and *%LT* when they are the first, a middle and the last token in a multi-token word respectively, and *%UT* when the token is actually a (unique token) word in the terms used here. Note that possessives are considerer a completely new token, whereas tokens linked by hyphens are considered as part of a multi-token word.

During this procedure, paragraphs and sentences are also identified and marked in the text. The beginning of a paragraph is marked with a label *%BP* followed by the paragraph identifier and the text zone in which it is found. Valid text zones are *p*

(paragraph of text), *SLUG*, *DATE*, *NWORDS*, *PREAMBLE* and *TRAILER*, in accordance with the sections contained in the New York Time documents compiled for the corpora. This text zone is also made explicit in tokens by adding a feature (labelled *%ZN*) with the corresponding value to each one of them. The labels *%%SOS* and *%%EOS* are used to mark the start and the end of sentences respectively.

The second step is submitting these tokenized sentences to the MBSP parser (Daelemans, Veenstra and Buchholz 1999). This parser assigns a part-of-speech tag to each token and identifies the chunks in the sentence. This information is marked by adding a label *%PoS* to each token and indicating the beginning of each phrase with the label *%CT*.

In the third step, orthographic features are associated to each token by adding one or more *word features* (labelled *%WF*) that indicate the orthographic features presented by the word to which the token belongs. Tokens in multi-token words also have *token features* (labelled *%TF*) which give information about the orthographic features of the particular token.

The following is the output of these pre-processing steps applied to the example sentence. The mark %NE identifies the named entity class in BIO notation.

```
% BP 16 p
%% SOS
%CT=NP
%UT=Last %WF=icp %NE=Itime %PoS=JJ %ZN=p
%UT=night %WF=ucp %NE=Itime %PoS=NN %ZN=p
%UT='s %WF=mix %WF=ucp %NE=O %PoS=POS %ZN=p
%UT=crash %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=VP
%UT=came %WF=ucp %NE=O %PoS=VBD %ZN=p
%UT=just %WF=ucp %NE=O %PoS=RB %ZN=p
%UT=hours %WF=ucp %NE=O %PoS=VBZ %HD=true %ZN=p
%CT=P
%UT=after %WF=ucp %NE=O %PoS=IN %ZN=p
%CT=NP
%UT=St. %WF=icp %WF=abb %NE=Ilocation %PoS=NNP %ZN=p
%FT=Louis %WF=icp %TF=icp %NE=Ilocation %PoS=JJ %ZN=p
%MT=- %WF=icp %TF=ncp %NE=O %PoS=JJ %ZN=p
%LT=based %WF=icp %TF=ucp %NE=O %PoS=JJ %ZN=p
%UT=TWA %WF=acp %WF=icp %NE=Iorganisation %PoS=NNP %HD=true %ZN=p
%CT=VP
%UT=reported %WF=ucp %NE=O %PoS=VBD %HD=true %ZN=p
%CT=NP
%UT=a %WF=ucp %NE=O %PoS=DT %ZN=p
%UT=fivefold %WF=ucp %NE=O %PoS=JJ %ZN=p
%UT=increase %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=P
%UT=in %WF=ucp %NE=O %PoS=IN %ZN=p
%CT=NP
%FT=second %WF=ucp %TF=ucp %NE=Idate %PoS=JJ %ZN=p
%MT=- %WF=ucp %TF=ncp %NE=Idate %PoS=JJ %ZN=p
%LT=quarter %WF=ucp %TF=ucp %NE=Idate %PoS=JJ %ZN=p
```

```
%UT=profit %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=CONJ
%UT=. %WF=ncp %NE=O %PoS=. %ZN=p
%% EOS
%% BP 17 p
...
```

The text is then processed further to obtain the input for the training algorithm of the maximum entropy model. In the case of LexMENE, the information provided by the parser is completely ignored. In the resulting text, the first line indicates the features —all atomic in this case— given in each line. The feature *tok0* represents the (token) lexical feature of the focus token, and *tkf0* its orthographic feature. Note how some orthographic features identified in the text are actually not provided to the maximum entropy model, as they were not defined in MENE's configuration. In addition, named entity tags are translated into the FMLU notation. In this notation, tokens are marked as the first, a middle or the last token of a multi-token named entity or the unique token in a one-token named entity. This information is kept by the prefix of the tag: *F-*, *M-*, *L-* or *U-* respectively. The second part of a named entity tag indicates the actual named entity class of the token. When a token is not part of any target named entity, the tag *O* is assigned. The following is the example text after these processing, in which every line —except the first one— represent a training event for the GIS algorithm in used implemented in the maxent package version 2.1.0.

```
%% FEATURES tkf+1 tkf+2 tkf-1 tkf-2 tkf0 tok+1 tok+2 tok-1 tok-2 tok0 znf FMLU
...
ucp ucp * * icp night 's NONE NONE last p F-time
ucp ucp icp * ucp 's crash last NONE night p L-time
ucp ucp ucp icp ucp crash came night last 's p O
ucp ucp ucp ucp ucp came just 's night crash p O
ucp ucp ucp ucp ucp just hours crash 's came p O
ucp ucp ucp ucp ucp hours after came crash just p O
ucp icp ucp ucp ucp after st. just came hours p O
icp icp ucp ucp ucp st. louis hours just after p O
icp * ucp ucp icp louis - after hours st. p F-location
* ucp icp ucp icp - based st. after louis p L-location
ucp acp#icp icp icp * based twa louis st. - p O
acp#icp ucp * icp ucp twa reported - louis based p O
ucp ucp ucp * acp#icp reported a based - twa p U-organisation
ucp ucp acp#icp ucp ucp a fivefold twa based reported p O
ucp ucp ucp acp#icp ucp fivefold increase reported twa a p O
ucp ucp ucp ucp ucp increase in a reported fivefold p O
ucp ucp ucp ucp ucp in second fivefold a increase p O
ucp * ucp ucp ucp second - increase fivefold in p O
* ucp ucp ucp ucp - quarter in increase second p F-date
ucp ucp ucp ucp * quarter profit second in - p M-date
ucp * * ucp ucp profit . - second quarter p L-date
* * ucp * ucp . NONE quarter - profit p O
* * ucp ucp * NONE NONE profit quarter . p O
...
```

Note also that none of the tokens in this sentence is considered unknown for the training process. This is because all of them have been seen more than three times in the training documents, according to Borthwick's (1999) definitions.

B.2 Decoding

For decoding, input texts have a similar pre-processing than training documents: the text is first tokenized; then subjected to the parser; then each token is associated with lexical, orthographic features, the zone feature and its named entity class; and finally the features of context tokens are gathered. Considerer the following (key) input text from the dryrun corpus.

> <ORGANISATION>Valujet Airlines</ORGANISATION> stock dropped sharply <DATE>Monday</DATE>, the first day of trading since the crash and also the first day of intensified federal scrutiny of the <LOCATION>Atlanta</LOCATION>-based carrier.

The following is the text resulting of pre-processing this sentence.

```
%% BP 23 p
%% SOS
%CT=NP
%UT=ValuJet %WF=icp %WF=mcp %NE=Iorganisation %PoS=NNP %ZN=p
%UT=Airlines %WF=icp %NE=Iorganisation %PoS=NNPS %ZN=p
%UT=stock %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=VP
%UT=dropped %WF=ucp %NE=O %PoS=VBD %HD=true %ZN=p
%CT=ADVP
%UT=sharply %WF=ucp %NE=O %PoS=RB %ZN=p
%CT=NP
%UT=Monday %WF=icp %NE=Idate %PoS=NNP %HD=true %ZN=p
%CT=CONJ
%UT=, %WF=ncp %NE=O %PoS=, %ZN=p
%CT=NP
%UT=the %WF=ucp %NE=O %PoS=DT %ZN=p
%UT=first %WF=ucp %NE=O %PoS=JJ %ZN=p
%UT=day %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=P
%UT=of %WF=ucp %NE=O %PoS=IN %ZN=p
%CT=VP
%UT=trading %WF=ucp %NE=O %PoS=VBG %HD=true %ZN=p
%CT=P
%UT=since %WF=ucp %NE=O %PoS=IN %ZN=p
%CT=NP
%UT=the %WF=ucp %NE=O %PoS=DT %ZN=p
%UT=crash %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=CONJ
%UT=and %WF=ucp %NE=O %PoS=CC %ZN=p
%CT=ADVP
```

```
%UT=also %WF=ucp %NE=O %PoS=RB %ZN=p
%CT=NP
%UT=the %WF=ucp %NE=O %PoS=DT %ZN=p
%UT=first %WF=ucp %NE=O %PoS=JJ %ZN=p
%UT=day %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=P
%UT=of %WF=ucp %NE=O %PoS=IN %ZN=p
%CT=NP
%UT=intensified %WF=ucp %NE=O %PoS=JJ %ZN=p
%UT=federal %WF=ucp %NE=O %PoS=JJ %ZN=p
%UT=scrutiny %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=P
%UT=of %WF=ucp %NE=O %PoS=IN %ZN=p
%CT=NP
%UT=the %WF=ucp %NE=O %PoS=DT %ZN=p
%FT=Atlanta %WF=icp %TF=icp %NE=llocation %PoS=JJ %ZN=p
%MT=- %WF=icp %TF=ncp %NE=O %PoS=JJ %ZN=p
%LT=based %WF=icp %TF=ucp %NE=O %PoS=JJ %ZN=p
%UT=carrier %WF=ucp %NE=O %PoS=NN %HD=true %ZN=p
%CT=CONJ
%UT=. %WF=ncp %NE=O %PoS=. %ZN=p
%% EOS
%% SOS
...
```

The following text is the piece of the decoding events file provided to the maximum entropy model for classification corresponding to the example sentence. Although is basicaly identical to the training events file, there are three considerations which should be noticed. Firstly, the decoding process ignores the annotated named entities at the end of each line. These annotations are included in order to performed the comparison with the predicted classes by the scoring algorithm. Secondly, sentence boundaries are kept because they are necessary to applied the Viterbi algorithm, which estimates the most probable sequence of labels (named entity classes) for the whole sentence, and ultimately to translate the annotation and predictions back to the BIO notation, which is the notation used by the scoring program. Finally, each line starts with the token producing the event. This is also ignored during the maximum entropy application, and is kept in this file to make more human readable the report of the scoring program.

```
%% FEATURES tkf+1 tkf+2 tkf-1 tkf-2 tkf0 tok+1 tok+2 tok-1 tok-2 tok0 znf FMLU
...
%% SOS
valujet icp ucp * * icp#mcp airlines stock NONE NONE valujet p F-organisation
airlines ucp ucp icp#mcp * icp stock dropped valujet NONE airlines p L-organisation
stock ucp ucp icp icp#mcp ucp dropped UNK airlines valujet stock p O
dropped ucp icp ucp icp ucp UNK monday stock airlines dropped p O
sharply icp * ucp ucp ucp monday , dropped stock UNK p O
monday * ucp ucp ucp icp , the UNK dropped monday p U-date
, ucp ucp icp ucp * the first monday UNK , p O
the ucp ucp * icp ucp first day , monday the p O
first ucp ucp ucp * ucp day of the , first p O
```

```
day ucp ucp ucp ucp ucp of trading first the day p O
of ucp ucp ucp ucp ucp trading since day first of p O
trading ucp ucp ucp ucp ucp since the of day trading p O
since ucp ucp ucp ucp ucp the crash trading of since p O
the ucp ucp ucp ucp ucp crash and since trading the p O
crash ucp ucp ucp ucp ucp and also the since crash p O
and ucp ucp ucp ucp ucp also the crash the and p O
also ucp ucp ucp ucp ucp the first and crash also p O
the ucp ucp ucp ucp ucp first day also and the p O
first ucp ucp ucp ucp ucp day of the also first p O
day ucp ucp ucp ucp ucp of UNK first the day p O
of ucp ucp ucp ucp ucp UNK federal day first of p O
intensified ucp ucp ucp ucp ucp federal scrutiny of day UNK p O
federal ucp ucp ucp ucp ucp scrutiny of UNK of federal p O
scrutiny ucp ucp ucp ucp ucp of the federal UNK scrutiny p O
of ucp icp ucp ucp ucp the atlanta scrutiny federal of p O
the icp * ucp ucp ucp atlanta - of scrutiny the p O
atlanta * ucp ucp ucp icp - based the of atlanta p U-location
- ucp ucp icp ucp * based carrier atlanta the - p O
based ucp * * icp ucp carrier . - atlanta based p O
carrier * * ucp * ucp . NONE based - carrier p O
. * * ucp ucp * NONE NONE carrier based . p O
%% EOS
...
```

The output to this input is the probability of every token of being associated with all 29 named entity tags. For example, the following is the output for the word *Atlanta-based*.

```
atlanta U-location
  F-date 2.3582254497589891E-4
  F-location 0.0056284907665606985
  F-money 5.8084742966748794E-5
  F-organisation 0.003972190320722765
  F-percent 3.8034777646051736E-6
  F-person 2.749348552523396E-5
  F-time 1.732437548339911E-5
  L-date 4.789126177176374E-6
  L-location 8.880591606821247E-4
  L-money 7.464585779427727E-9
  L-organisation 3.538716311993472E-5
  L-percent 1.1149902005758877E-5
  L-person 3.919547376509764E-7
  L-time 5.908821429276911E-5
  M-date 5.674433920287156E-6
  M-location 1.572481606027267E-5
  M-money 3.0283646902420904E-5
  M-organisation 1.020023235271163E-4
  M-percent 3.338873085558022E-8
  M-person 5.601148139433899E-6
  M-time 4.29918584804938E-6
  O 0.03622207086611734
  U-date 2.523912498607799E-4
  U-location 0.9454951129329978
  U-organisation 0.006762640356509325
```

U-person 1.6120473375125032E-4
U-time 8.782180344493859E-7

- O

F-date 2.2427269407620365E-4
F-location 9.761312465862E-7
F-money 1.98000802195393E-6
F-organisation 1.0099160338869072E-5
F-percent 6.436769989325952E-7
F-person 1.5565573843853617E-7
F-time 0.001454113467014768
L-date 1.5315163103567881E-4
L-location 0.0029026916121269344
L-money 1.0655820638683854E-5
L-organisation 0.0035994135656187585
L-percent 2.895073508629643E-6
L-person 1.3034007607261184E-4
L-time 3.636429566865979E-6
M-date 1.0473197292275692E-4
M-location 8.115132644627172E-5
M-money 1.0214220922467269E-4
M-organisation 6.306883913024113E-4
M-percent 3.9878955612484467E-8
M-person 4.4994337643814996E-7
M-time 2.452453201441192E-5
O 0.9902344229398096
U-date 7.953225159614118E-5
U-location 6.121161821338814E-6
U-organisation 1.9101024534620337E-4
U-person 1.7384016768339187E-5
U-time 3.277612841174649E-5

based O
F-date 0.0010599138066777003
F-location 2.3339245958880563E-5
F-money 1.579078437197117E-6
F-organisation 1.2458725322157061E-6
F-percent 5.902656497025372E-6
F-person 2.7884670045230972E-5
F-time 2.4540518125800207E-4
L-date 6.232198488898867E-5
L-location 9.21150184764415E-6
L-money 4.305207280045616E-6
L-organisation 2.6005054949159107E-4
L-percent 2.568766574996694E-6
L-person 2.0963634223038765E-6
L-time 2.394891056400395E-5
M-date 1.5180629034018936E-5
M-location 4.098018079674603E-5
M-money 5.0642669272570075E-6
M-organisation 0.0011839996710339387
M-percent 7.884049038543913E-7
M-person 4.062421773683823E-6
M-time 0.0030214750319969372
O 0.993220056089924
U-date 6.5426657607175796E-6
U-location 5.350868941303706E-4

U-organisation 1.3260809718281386E-6
U-person 1.8431244086216143E-4
U-time 5.135142640852881E-5

These probabilities correspond to the input for the Viterbi search explained in section 3.3.2. The output of this search is a file in which each token is associated with a unique named entity class. The following is the output resulting for the example input. It can be noted that LexMENE misclassifies the token *Monday* in this example.

%% SOS
valujet F-organisation F-organisation
airlines L-organisation L-organisation
stock O O
dropped O O
sharply O O
monday U-date O
, O O
the O O
first O O
day O O
of O O
trading O O
since O O
the O O
crash O O
and O O
also O O
the O O
first O O
day O O
of O O
intensified O O
federal O O
scrutiny O O
of O O
the O O
atlanta U-location U-location
- O O
based O O
carrier O O
. O O
%% EOS

In the final step, sentence delimiters are replaced by empty lines and the familiarity types are added to enable the scoring program to produce the performance report.

Appendix C

Decision lists

In this appendix, the hypotheses built by the Ripper algorithm are presented. In each MOLI MENE version that utilises this algorithm, the real name of the features are replaced by the simpler names f1, f2, etc., so that the process that checks which rules are fired by each example can do so very efficiently by just looking up into a table. Consequently, each decision list presented here is preceded by a table with the interpretation of each feature reported in the final hypothesis outputted by the Ripper algorithm.

C.1 MOLI MENE V6

Recall that MOLI MENE V6 introduces features which inform the algorithm of the chunk tags and the head words of these chunks in a window of sizes [4,4]. Therefore, there are 18 new features per token, whose relation with the names as used by Ripper can be seen in table C.1.

It should be remarked that the atomic features are considered *set-valued*, that is their value is a set of strings (Cohen 1996). This is not really necessary for the features employed here, as they could be transformed into nominal features — though it would be a bit unnatural for the head word feature. Nevertheless, this kind of features are necessary for the next versions of MOLI MENE and consequently also used in this version for the sake of standarisation.

The following is the outputted final hypothesis given by the Ripper algorithm. In the format utilised in this implementation by Cohen (1995), each line contains the class predicted followed by the conditions of the rule. The symbols ~ and !~ represent the symbols $\in$ and $\notin$ respectively, so that the condition is met if the value of the feature —a set— contains or not a given string. At the end of each rule, the Ripper algorithm reports the number of positive and negative examples covered by the rule.

Table C.1: Relation between the names of the features as used by Ripper and the more linguistically informed features in MOLI MENE V3/V6/V9.

Ripper's name	MOLI MENE V3/V6/V9 features
f1	$chunk_{-4}$'s tag
f2	$chunk_{-4}$'s head word
f3	$chunk_{-3}$'s tag
f4	$chunk_{-3}$'s head word
f5	$chunk_{-2}$'s tag
f6	$chunk_{-2}$'s head word
f7	$chunk_{-1}$'s tag
f8	$chunk_{-1}$'s head word
f9	focus chunk's tag
f10	focus chunk's head word
f11	$chunk_{+1}$'s tag
f12	$chunk_{+1}$'s head word
f13	$chunk_{+2}$'s tag
f14	$chunk_{+2}$'s head word
f15	$chunk_{+3}$'s tag
f16	$chunk_{+3}$'s head word
f17	$chunk_{+4}$'s tag
f18	$chunk_{+4}$'s head word

Utime :- f10 ~ night, f5 ~ NP, f15 ~ VP (5/4).
Mmoney :- f10 ~ million, f13 ~ CONJ, f1 ~ CONJ (5/2).
Mdate :- f10 ~ years, f11 ~ ADVP, f7 ~ NP (12/0).
Mdate :- f10 ~ years, f6 ~ UNK, f15 ~ CONJ (6/5).
Fdate :- f9 ~ NP, f10 ~ last (36/6).
Ldate :- f11 ~ CONJ, f10 ~ ago (20/1).
Ldate :- f11 ~ CONJ, f8 ~ last (19/4).
Uperson :- f12 ~ said, f8 ~ P_COMMA, f10 !~ he, f13 ~ CONJ, f3 ~ VP, f10 !~ official, f10 !~ she (33/14).
Uperson :- f12 ~ said, f8 ~ P_COMMA, f13 ~ CONJ, f10 !~ he, f10 ~ UNK (10/1).
Ulocation :- f11 ~ CONJ, f3 ~ P, f8 ~ in, f12 ~ P_COMMA, f15 ~ CONJ, f1 !~ CONJ, f18 !~ UNK, f2 !~ bombed (16/0).
Ulocation :- f9 ~ NP, f11 ~ CONJ, f8 ~ P_COMMA, f3 ~ P, f15 ~ NP, f10 !~ UNK (36/28).
Ulocation :- f11 ~ CONJ, f8 ~ in, f3 ~ P, f10 ~ atlanta (8/0).
Ulocation :- f9 ~ NP, f11 ~ CONJ, f10 ~ washington (37/0).
Uorganisation :- f9 ~ NP, f10 ~ valujet (109/30).
Uorganisation :- f9 ~ NP, f10 ~ faa (105/103).
Uorganisation :- f9 ~ NP, f10 ~ twa, f8 !~ after (54/14).
default O (80755/10148).

========== summary ==========

Train error rate: 11.31% +/- 0.10% (91626 datapoints) <<
Hypothesis size: 16 rules, 74 conditions
Learning time: 247.31 sec

In this example, the first rule states that if the focus head word is *night* and the third and the eighth chunks in the context are a noun phrase and a verb phrase respectively, then the token must be classified as a one-token time expression. This rules covers nine

examples: five of them are classified correctly and four incorrectly.

C.2 MOLI MENE V7

MOLI MENE V7 cosideres features which include the lemmas of the tokens in a window of sizes [1,1], their part-of-speech tags and the WordNet® synsets of their close synonyms. In other words, there are nine new features per token. Because there is not any kind of word sense disambiguation applied, all this features are of type *set* (Cohen 1996). In this way, a word can be —for example— both a noun and an adjective having both part-of-speech tags, the corresponding lemmas and all the close synonyms synsets for every sense in each lexical category.

Table C.2: Relation between the names of the features as used by Ripper and the more linguistically informed features in MOLI MENE V4/V7/V10.

Ripper's name	MOLI MENE V4/V7/V10 features
f1	$token_{-1}$'s lemmas
f2	$token_{-1}$'s PoS tag
f3	$token_{-1}$'s close synonyms synsets
f4	focus token's tag
f5	focus token's PoS tag
f6	focus token's close synonyms synsets
f7	$token_{+1}$'s lemmas
f8	$token_{+1}$'s PoS tag
f9	$token_{+1}$'s close synonyms synsets

Table C.2 presents the names used by Ripper in building the following decision list hypothesis with the more informed features of MOLI MENE V7. As an example, the fourth rule indicates that if the token that follows the focus token is a synonym of the meaning *n12793745* (e.g. day, space-age), and the token that preceeds the focus token is a form of the word *late*, then the focus token must be classified a token in the middle of a time expression. This rules correctly classifies three examples.

```
Lpercent :- f4 ~ percent (34/0).
Fpercent :- f7 ~ percent (33/1).
Mtime :- f7 ~ local_time (16/0).
Mtime :- f9 ~ n12793745, f1 ~ late (3/0).
Mtime :- f9 ~ n12896567, f2 ~ CC, f7 ~ second (4/0).
Utime :- f6 ~ n12832221, f3 ~ n12831744 (41/4).
Utime :- f6 ~ n12836480, f1 ~ sunday (2/0).
Utime :- f6 ~ n12836480, f8 ~ NN, f6 !~ n12823670 (4/0).
Utime :- f6 ~ n12833627, f4 !~ day (11/4).
Utime :- f4 ~ tonight (2/0).
Mmoney :- f2 ~ '$', f8 ~ CD, f2 !~ PRP (64/0).
Mmoney :- f9 ~ n11302011, f9 !~ n11293334 (6/1).
Mlocation :- f8 ~ NNP, f9 ~ n7498246, f2 ~ NNP (35/6).
```

Mlocation :- f8 ~ NNP, f9 ~ n2336754 (11/3).
Mlocation :- f8 ~ NNP, f2 ~ NNP, f7 ~ international (6/0).
Mlocation :- f8 ~ NNP, f1 ~ new_york_city (6/0).
Mlocation :- f8 ~ NNP, f2 ~ NNP, f3 ~ n7574159 (4/3).
Lmoney :- f2 ~ CD, f5 ~ CD, f8 ~ IN (19/7).
Lmoney :- f5 ~ CD, f2 ~ CD, f8 ~ DT (8/0).
Lmoney :- f5 ~ CD, f2 ~ '$', f2 !~ PRP (23/2).
Lmoney :- f2 ~ CD, f5 ~ CD, f8 ~ P_COMMA (10/6).
Lmoney :- f2 ~ CD, f6 ~ n11524771 (14/0).
Lmoney :- f2 ~ CD, f5 ~ CD, f8 ~ NN, f7 !~ flight, f7 !~ plane (7/1).
Lmoney :- f2 ~ CD, f5 ~ CD, f8 ~ P_COLON (3/0).
Fmoney :- f5 ~ '$', f8 ~ CD, f5 !~ PRP (85/5).
Fmoney :- f7 ~ cent (12/0).
Fmoney :- f1 ~ donate (1/0).
Mperson :- f6 ~ n5710086, f2 ~ NNP, f8 ~ NNP, f1 !~ bc, f3 !~ n6790797, f3 !~ n11594993 (49/4).
Mperson :- f2 ~ NNP, f3 ~ n5408511, f8 ~ NNP (5/0).
Mperson :- f5 ~ NNP, f3 ~ n8049747, f6 ~ n5938672 (12/0).
Mperson :- f2 ~ NNP, f8 ~ NNP, f7 ~ jr (4/0).
Mperson :- f2 ~ NNP, f8 ~ NNP, f1 ~ hillary (3/0).
Ftime :- f5 ~ CD, f8 ~ CD, f2 ~ NNP (101/1).
Ftime :- f9 ~ n5294032, f4 !~ pan (23/0).
Ftime :- f7 ~ pP_PERIODmP_PERIOD (9/0).
Ftime :- f9 ~ n12836480, f4 ~ last (9/1).
Ftime :- f7 ~ aP_PERIODmP_PERIOD (8/0).
Ltime :- f2 ~ CD, f5 ~ CD, f8 !~ CD, f8 !~ NNS, f8 !~ P_PERIOD, f8 !~ TO, f8 !~ CC, f8 !~ NNP, f8 !~ JJ, f8 !~ NN (101/8).
Ltime :- f6 ~ n12824116, f4 ~ night, f2 ~ JJ (11/3).
Ltime :- f6 ~ n5294032, f1 !~ pan, f8 !~ NNP (20/0).
Ltime :- f6 ~ n12803990 (9/0).
Ltime :- f4 ~ pP_PERIODmP_PERIOD (6/0).
Ltime :- f4 ~ aP_PERIODmP_PERIOD (4/0).
Mdate :- f9 ~ n12811110, f2 ~ JJ, f5 ~ CD (21/1).
Mdate :- f6 ~ n12811110, f8 ~ RB (33/14).
Mdate :- f9 ~ n12846772, f1 ~ earlier (9/0).
Mdate :- f3 ~ n12877042, f8 ~ P_COMMA, f1 ~ sept (4/0).
Mdate :- f2 ~ CD, f8 ~ CD, f5 ~ P_COMMA (23/9).
Fdate :- f6 ~ n12873180, f8 ~ CD (127/3).
Fdate :- f9 ~ n12896721, f4 ~ last (54/2).
Fdate :- f9 ~ n12811110, f7 ~ week, f8 ~ NN (18/7).
Fdate :- f9 ~ n12811110, f7 ~ year, f5 ~ DT, f2 !~ IN (14/11).
Fdate :- f6 ~ a1370871, f8 ~ NNP (22/0).
Fdate :- f9 ~ n12811110, f5 ~ CD, f2 !~ IN, f7 ~ years, f1 !~ be, f2 !~ RB, f2 !~ VBG, f2 !~ TO, f2 !~ CC (15/5).
Ldate :- f6 ~ n12896721, f1 ~ last (54/2).
Ldate :- f3 ~ n12872860, f5 ~ CD (104/21).
Ldate :- f6 ~ n12811110, f4 ~ week, f5 ~ NN (18/6).
Ldate :- f6 ~ n12811110, f4 ~ year, f2 ~ DT, f8 !~ IN, f7 !~ earlier (13/9).
Ldate :- f6 ~ n12811110, f4 ~ month, f5 ~ NN (9/2).
Ldate :- f8 ~ P_COMMA, f3 ~ n12811110, f4 ~ ago (13/0).
Ldate :- f6 ~ n12811110, f6 ~ n11598583, f8 ~ P_COMMA (4/1).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 ~ IN, f9 ~ n7533497, f9 !~ n7430143 (43/1).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f9 ~ n7130825, f2 !~ IN (70/21).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f9 ~ n7653466 (47/9).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f9 ~ n976162, f2 !~ IN, f2 !~ DT, f2 !~ CD, f6 ~ n976162 (13/1).

Flocation :- f5 ~ NNP, f8 ~ NNP, f2 ~ IN, f4 ~ pearl_harbor (6/0).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 ~ IN, f6 ~ n7430143, f4 !~ new_york (13/0).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f9 ~ n7700297, f4 !~ uP_PERIODsP_PERIOD (17/3).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f6 ~ n7430143, f9 ~ n6993235 (8/1).
Flocation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f7 ~ airport (14/2).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n7170582, f3 ~ n7170582 (102/1).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n7130825, f6 !~ n12345618 (50/11).
Llocation :- f2 ~ NNP, f5 ~ NNP, f6 ~ n7653466, f3 ~ n7172103 (32/1).
Llocation :- f2 ~ NNP, f6 ~ n2342336 (50/18).
Llocation :- f2 ~ NNP, f6 ~ n7158145, f3 ~ n7144938 (21/0).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n7700297 (17/4).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n7710579 (12/0).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n7745910 (12/1).
Llocation :- f2 ~ NNP, f5 ~ NNP, f6 ~ n7034213, f4 ~ county (11/3).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n7164229, f3 ~ n7164229 (8/0).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f4 ~ pearl_harbor (10/0).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n3573798 (6/1).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f1 ~ persian_gulf (6/0).
Llocation :- f2 ~ NNP, f5 ~ NNP, f4 ~ york (5/1).
Llocation :- f2 ~ NNP, f4 ~ everglades (9/0).
Llocation :- f2 ~ NNP, f5 ~ NNP, f6 ~ n7552184 (5/2).
Llocation :- f5 ~ NNP, f2 ~ NNP, f6 ~ n7457534 (9/8).
Llocation :- f2 ~ NNP, f5 ~ NNP, f6 ~ n7740659 (5/1).
Llocation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f3 ~ n7003112 (7/6).
Llocation :- f2 ~ NNP, f6 ~ n7130825, f5 ~ NNPS (4/0).
Llocation :- f2 ~ NNP, f5 ~ JJ, f8 ~ JJ (9/6).
Llocation :- f5 ~ NNP, f2 ~ NNP, f3 ~ n7533173, f6 ~ n7533173 (5/1).
Llocation :- f5 ~ NNP, f6 ~ n11834397 (4/1).
Llocation :- f5 ~ NNP, f6 ~ n7758560, f2 ~ NNP (3/0).
Llocation :- f5 ~ NNP, f1 ~ camp (3/0).
Fperson :- f5 ~ NNP, f8 ~ NNP, f2 ~ NN, f3 ~ n8041449 (33/1).
Fperson :- f5 ~ NNP, f8 ~ NNP, f2 ~ VBD, f1 ~ say (68/17).
Fperson :- f5 ~ NNP, f8 ~ NNP, f6 ~ n9160375 (41/7).
Fperson :- f5 ~ NNP, f8 ~ NNP, f9 ~ n5710086, f6 !~ n11594993, f9 !~ n6737514 (41/15).
Fperson :- f5 ~ NNP, f8 ~ NNP, f3 ~ n8655657, f6 !~ n8788729 (30/1).
Fperson :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 !~ DT, f9 ~ n9184552, f7 !~ service, f9 !~ n8307222 (18/2).
Fperson :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 !~ DT, f4 ~ robert (11/0).
Fperson :- f5 ~ NNP, f8 ~ NNP, f2 !~ DT, f2 !~ NNP, f6 !~ n7130825, f4 !~ bc, f6 ~ n8923881 (10/1).
Lperson :- f2 ~ NNP, f8 ~ P_COMMA, f5 ~ NNP, f3 ~ n5708379 (32/2).
Lperson :- f2 ~ NNP, f8 ~ P_COMMA, f5 ~ NNP, f3 !~ n2771586 (239/132).
Lperson :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n9296454, f4 !~ service (45/2).
Lperson :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f8 ~ VBD, f9 ~ v648722, f3 !~ n3977588, f6 !~ n2847188, f3 !~ n6748862, f1 !~ air_force 26/12).
Uperson :- f5 ~ NNP, f8 ~ VBD, f9 ~ v648722, f2 ~ P_SINGLEQUOTEP_SINGLEQUOTE (70/0).
Uperson :- f5 ~ NNP, f8 ~ VBD, f9 ~ v734519, f2 !~ NNP, f2 !~ DT, f2 !~ IN (99/9).
Uperson :- f5 ~ NNP, f8 ~ VBD, f2 ~ P_COMMA (31/14).
Uperson :- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP, f6 ~ n8908571 (32/0).
Uperson :- f5 ~ NNP, f8 !~ NNP, f8 ~ VBD, f6 ~ n9360060, f6 !~ n9067642 (11/0).
Uperson :- f5 ~ NNP, f8 !~ NNP, f3 ~ v734519, f6 !~ n12831744, f8 ~ P_PERIOD (12/4).
Uperson :- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP, f6 ~ n8696598, f4 !~ houston (33/3).

Uperson :- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP, f6 ~ n9351365 (18/6).
Uperson :- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP, f3 ~ v600025, f8 ~ P_COMMA (9/2).
Udate :- f5 ~ CD, f1 ~ c (90/0).
Udate :- f6 ~ n12831744, f2 !~ JJ, f4 !~ th (200/1).
Udate :- f5 ~ CD, f8 !~ NNS, f2 ~ IN, f8 ~ P_COMMA (43/8).
Udate :- f5 ~ CD, f8 !~ NNS, f2 !~ NNP, f8 !~ IN, f2 !~ P_COMMA, f8 !~ JJ, f8 !~ CD, f8 !~ NN, f2 !~ NN, f3 !~ v50921, f2 !~ CC, f2 !~ CD, f8 !~ CC, f8 !~ P_COMMA, f8 !~ NNP, f8 !~ P_PERIOD, f8 !~ TO, f2 !~ NNPS, f2 !~ P_COLON, f8 !~ JJR, f8 !~ VBN, f9 !~ v50921 (111/83).
Udate :- f5 ~ NNP, f8 ~ CD, f2 ~ NNP, f6 !~ n6790295, f1 !~ bound, f3 !~ n12425532, f4 !~ world, f1 !~ honeymoon, f1 !~ delta, f1 !~ district, f4 !~ bound (101/6).
Udate :- f5 ~ CD, f2 ~ IN, f8 ~ P_PERIOD (25/24).
Udate :- f5 ~ CD, f2 ~ DT, f8 ~ NN (21/7).
Udate :- f6 ~ n12824724, f6 !~ n12846772 (30/3).
Udate :- f6 ~ n12877042, f2 ~ IN (28/5).
Udate :- f4 ~ sunday (28/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f5 ~ NNP, f3 ~ n7130825, f4 ~ times (84/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f5 ~ NNP, f6 ~ n5590116, f1 !~ crash (95/0).
Morganisation :- f5 ~ NNP, f2 ~ NNP, f8 ~ NNP, f9 ~ n2847188, f1 ~ national (41/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f6 ~ n976162 (73/9).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n6882991 (61/1).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n6769589 (44/2).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f3 ~ n5272695 (25/10).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f6 ~ n3977588, f9 !~ n7528713, f1 !~ crash (18/2).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n6748862 (14/2).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f1 ~ uP_PERIODsP_PERIOD (12/5).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n12475555, f7 ~ co (8/0).
Morganisation :- f5 ~ NNP, f2 ~ NNP, f8 ~ NNPS (26/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n6895143 (8/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f6 ~ n7593134 (5/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n3085921, f1 !~ world, f7 !~ hall (9/0).
Morganisation :- f5 ~ NNP, f8 ~ NNP, f9 ~ n3725783 (16/2).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f4 ~ rent (8/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n6693458 (7/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f3 ~ n7593134, f9 !~ n6987833 (6/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n717997 (5/0).
Morganisation :- f5 ~ NNP, f2 ~ NNP, f3 ~ n7909591, f8 !~ NNP (8/1).
Morganisation :- f5 ~ NNP, f2 ~ NNP, f6 ~ n5975532 (7/1).
Morganisation :- f5 ~ NNP, f8 ~ NNP, f9 ~ n438352, f9 ~ n207743 (8/1).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f7 ~ police_department (4/0).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f3 ~ n6879976 (5/2).
Morganisation :- f2 ~ NNP, f8 ~ NNP, f9 ~ n9287812 (5/2).
Morganisation :- f5 ~ NNP, f8 ~ NNP, f9 ~ n5282591 (10/8).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 ~ DT, f9 ~ n6757947 (60/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f9 ~ n226730 (32/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f9 ~ n6788854 (34/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f6 !~ n5708379, f6 ~ n7676970 (17/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ CD, f4 ~ ny (28/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ IN, f6 ~ n7431482, f4 !~ la (65/25).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f6 ~ n7474210, f2 ~ CD (14/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f6 !~ n5938672, f9 ~ n3756403 (12/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f6 !~ n5938672, f9 ~

n8083876 (12/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f9 ~ n5592130 (25/1).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ IN, f6 ~ n2792013 (12/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f6 !~ n5708379, f9 !~ n7041023, f9 ~ n6686439 (10/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f6 !~ n5938672, f4 ~ uP_PERIODsP_PERIOD (10/2).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ IN, f9 ~ n6643594, f7 !~ europe (20/2).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f9 ~ n6895143 (9/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f6 ~ n7902212 (19/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f9 ~ n218158 (17/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f6 ~ n6757947 (28/3).
Forganisation :- f5 ~ NNP, f7 ~ airline, f8 ~ NNPS (32/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ IN, f9 ~ n6788854 (7/0).
Forganisation :- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f9 ~ n6637680 (6/0).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n6790797 (128/1).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n6882991, f4 !~ bureau (62/1).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n6769589, f1 !~ army (41/0).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f3 ~ n6757947 (41/1).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n3756403 (26/0).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f3 ~ n7457534 (25/4).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f8 ~ NN, f9 ~ n2341562 (10/5).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f3 ~ n2692952 (17/2).
Lorganisation :- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP, f6 ~ n549894, f1 !~ trade (20/0).
Lorganisation :- f2 ~ NNP, f5 ~ NNPS, f6 ~ n3898137 (46/1).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP, f6 ~ n5592130 (17/2).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP, f3 ~ n7170582 (13/2).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP, f6 ~ n5279815 (11/2).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP, f3 ~ n5272695 (6/1).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP, f6 ~ n6686439 (11/6).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP, f1 ~ national_guard (6/0).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f9 ~ n6790295 (15/6).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP, f6 ~ n12475555 (6/1).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f7 ~ dc (6/1).
Lorganisation :- f5 ~ NNP, f2 ~ NNPS, f8 !~ NNP, f8 !~ CD (12/3).
Lorganisation :- f2 ~ NNP, f1 ~ pan (16/1).
Lorganisation :- f5 ~ NNP, f6 ~ n6751988, f2 ~ IN (10/2).
Lorganisation :- f5 ~ NNP, f2 ~ NNP, f6 ~ n6788854, f1 ~ air_force (6/0).
Lorganisation :- f2 ~ NNP, f3 ~ n6789709, f1 ~ squadron (9/0).
Ulocation :- f5 ~ NNP, f2 ~ IN, f8 !~ NNP, f8 ~ P_COMMA (153/49).
Ulocation :- f5 ~ NNP, f6 ~ n7424046 (175/7).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f2 ~ IN, f6 ~ n7172103 (31/2).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f2 ~ P_COMMA, f6 ~ n7431482 (48/0).
Ulocation :- f5 ~ NNP, f6 ~ n7168879, f8 !~ NNP (96/5).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f2 ~ IN, f6 ~ n7474210 (27/3).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f2 ~ P_COMMA, f6 ~ n7165212 (30/0).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f6 ~ n7474210, f4 !~ independence, f2 !~ NNP, f4 !~ troy (35/2).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP, f6 ~ n7168503 (33/0).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f6 ~ n7529345 (52/1).
Ulocation :- f5 ~ NNP, f6 ~ n7166963, f9 !~ n7166963, f4 !~ jordan (40/5).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f2 ~ IN, f6 ~ n7130825 (15/2).
Ulocation :- f5 ~ NNP, f8 !~ NNP, f6 ~ n7165019 (22/1).
Uorganisation :- f5 ~ NNP, f2 ~ DT, f8 !~ NNP (412/228).
organisation :- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP, f8 ~ POS (71/53).

```
Uorganisation :- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f8 ~ NNS, f9 ~ n8686103 (27/5).
Uorganisation :- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP, f8 ~ VBZ (46/40).
Uorganisation :- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f6 !~ n12345618, f8 ~ NNS, f2 ~
JJ (10/2).
Uorganisation :- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f8 ~ NN, f9 ~ n231407 (10/0).
Uorganisation :- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f8 ~ NN, f2 ~ JJ (23/10).
Uorganisation :- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f6 !~ n12345618, f8 ~ ')', f2 ~ '('
(18/7).
Uorganisation :- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f6 !~ n12345618, f2 ~ CD, f8 !~
NNS (11/4).
Uorganisation :- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f6 !~ n12345618, f6 ~ n6788854
(11/2).
default O (80388/3411).

================ summary ================
Train error rate: 5.01% +/- 0.07% (91626 datapoints) <<
Hypothesis size: 221 rules, 1065 conditions
Learning time: 2533.86 sec
```

C.3 MOLI MENE V8

MOLI MENE V8 uses trigger synsets as linguistically informed features. As there are three lists of trigger synsets (the next-is list, the this-is list and the previous-is list), there are three new features per token with the synsets hit by the tokens in a window of sizes [1,1]. Table C.3 shows the relation between these features and the names that Ripper utilises.

Table C.3: Relation between the names of the features as used by Ripper and the more linguistically informed features in MOLI MENE V5/V8/V11.

Ripper's name	MOLI MENE V5/V8/V11 features
f1	synsets in the *next-is* list hit by token$_{-1}$
f2	synsets in the *this-is* list hit by the focus token
f3	synsets in the *previous-is* list by token$_{+1}$

The following is the final hypothesis as given by the Ripper algorithm.

```
Lpercent :- f2 ~ n11658514 (34/1).
Fpercent :- f3 ~ n11658514 (33/2).
Mtime :- f3 ~ n19843, f2 ~ n16993, f3 !~ n24936 (10/5).
Mtime :- f3 ~ n19843, f2 ~ r250909 (4/0).
Utime :- f2 ~ n12889670, f1 ~ n24936 (30/2).
Utime :- f2 ~ n12832221, f2 !~ n8944134 (24/6).
Utime :- f2 ~ n12889670, f2 ~ n12832754 (4/0).
Mmoney :- f3 ~ n11302011 (6/1).
Mlocation :- f3 ~ n2888347, f1 ~ n1742, f2 ~ n22634 (21/4).
Mlocation :- f3 ~ n2888347, f3 ~ n2336754 (10/4).
Mlocation :- f3 ~ n2342336, f1 ~ n1742, f2 ~ n13067 (11/2).
```

Mlocation :- f1 ~ n19046, f3 ~ n6992023, f2 ~ n19046 (7/0).
Mlocation :- f3 ~ n6905978, f2 ~ n7147136, f2 !~ n7937305 (4/0).
Lmoney :- f2 ~ n11295091, f2 ~ n11474780 (17/1).
Fmoney :- f3 ~ n11562705 (12/0).
Mperson :- f2 ~ n5708379, f1 ~ n13067 (38/7).
Mperson :- f2 ~ n5708379, f2 ~ n11603699 (4/1).
Mperson :- f2 ~ n5708379, f3 ~ n5303, f1 !~ n11455258 (6/2).
Mperson :- f2 ~ n5708379, f2 ~ n12366980 (2/0).
Mperson :- f1 ~ n5303, f1 ~ n8152966 (3/0).
Mperson :- f3 ~ n8388313 (4/0).
Mperson :- f2 ~ n11667742, f2 ~ n11586299, f1 !~ n24936 (3/0).
Mperson :- f1 ~ n8391286, f2 ~ n8338222 (2/0).
Ftime :- f3 ~ n5293770, f2 !~ n7856852 (23/0).
Ftime :- f3 ~ n12889670, f2 ~ a1370871 (9/2).
Ftime :- f3 ~ r250909 (8/0).
Ltime :- f3 ~ NoWordKey, f1 !~ n1742, f1 !~ n16993, f2 ~ n5293770 (4/0).
Mdate :- f3 ~ n12786206, f1 ~ a1370871 (11/1).
Mdate :- f3 ~ n12786206, f1 ~ r60367 (9/2).
Mdate :- f2 ~ n12786206, f3 ~ r73272 (23/1).
Mdate :- f1 ~ n12786206, f1 ~ n12872860, f1 ~ n6729321 (4/1).
Mdate :- f3 ~ n12786206, f3 ~ n12811259, f1 ~ a421470 (5/1).
Mdate :- f3 ~ n12786206, f1 ~ a2922382 (5/2).
Mdate :- f3 ~ n12786206, f1 ~ n12786206, f2 ~ n24936 (4/2).
Fdate :- f2 ~ n12872860, f1 !~ a1370871 (132/46).
Fdate :- f3 ~ n12786206, f2 ~ a1370871, f3 !~ n6067926 (76/2).
Fdate :- f3 ~ n12786206, f3 ~ n12811259, f1 ~ n1742 (9/6).
Fdate :- f3 ~ n12786206, f3 ~ n12806554, f1 ~ n1742 (7/0).
Fdate :- f3 ~ n12786206, f2 ~ n12826002 (8/3).
Fdate :- f3 ~ n12786206, f2 ~ a790152 (7/2).
Ldate :- f2 ~ n12786206, f1 ~ a1370871, f2 !~ n16840 (76/2).
Ldate :- f1 ~ n12872860, f3 !~ NoWordKey (106/68).
Flocation :- f3 ~ n7111224, f2 ~ n7111224, f2 !~ n7492354, f2 ~ n7058546 (104/23).
Flocation :- f3 ~ n1742, f2 ~ n1742, f3 !~ n13067, f2 !~ n13067, f3 !~ n14223, f2 ~ n7582157 (27/9).
Flocation :- f3 ~ n1742, f2 ~ n19046, f3 ~ n19046, f2 !~ n13067, f2 ~ n7117259 (13/0).
Flocation :- f3 ~ n7111224, f2 ~ n7111224, f3 !~ n7058546, f2 !~ n22634, f2 !~ n24503 (25/5).
Flocation :- f3 ~ n1742, f3 ~ n7130102, f3 ~ n9377530, f2 !~ n1742, f1 !~ n13067 (29/10).
Flocation :- f3 ~ n7111224, f2 ~ n7062038, f1 ~ NoWordKey (12/2).
Flocation :- f3 ~ n1742, f2 ~ n1742, f3 ~ n7667106, f2 ~ n7497888 (31/0).
Flocation :- f3 ~ n1742, f2 ~ n7130825, f3 ~ n7623806 (9/0).
Flocation :- f3 ~ n1742, f2 ~ n7130825, f3 ~ n7101501, f2 ~ n7758560 (6/0).
Llocation :- f2 ~ n7111224, f1 ~ n7111224, f3 !~ n22113, f2 !~ n16840, f2 ~ n7130102 (72/2).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 !~ n13067, f1 !~ n13067, f3 !~ n16993, f1 ~ n7058546 (77/12).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n7667106, f1 ~ n7497888 (30/1).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 !~ n13067, f2 ~ n7582157, f1 ~ n7582157 (24/3).
Llocation :- f2 ~ n1742, f2 ~ n2338751, f1 ~ n6961162 (16/0).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 !~ n13067, f1 ~ n7034213, f1 !~ n22634, f2 !~ n11800524, f1 !~ n7184130 (21/2).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n2338751, f1 ~ n5303 (16/0).
Llocation :- f2 ~ n1742, f1 ~ n19046, f2 ~ n7459488 (13/0).

Llocation :- f2 ~ n1742, f2 ~ n6778253, f1 !~ n1742 (28/15).
Llocation :- f2 ~ n1742, f1 ~ n1742, f1 !~ n13067, f2 ~ n7623806 (9/0).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n7058546, f2 ~ n7653466 (6/0).
Llocation :- f2 ~ n1742, f1 ~ n19046, f1 ~ n7505383 (6/1).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n2342336, f1 !~ n7111224 (6/2).
Llocation :- f2 ~ n1742, f2 ~ n7036073 (11/5).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n12897329 (6/0).
Llocation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n7582157, f1 ~ n7653466 (4/0).
Llocation :- f2 ~ n1742, f1 ~ n1742, f1 ~ n3277837 (7/6).
Llocation :- f2 ~ n1742, f2 ~ n7575544 (6/5).
Llocation :- f2 ~ n1742, f2 ~ n7582157, f2 ~ n3783785 (4/3).
Fperson :- f2 ~ n2956, f2 ~ n7953417, f2 !~ n8032285 (77/20).
Fperson :- f2 ~ n2956, f3 ~ n5708379, f3 !~ n12599031 (21/1).
Fperson :- f2 ~ n3135, f2 ~ n8523465 (19/0).
Fperson :- f1 ~ n5303, f1 ~ n8337045, f1 ~ n8655657, f2 !~ n8655657, f1 ~ n22113 (25/3).
Fperson :- f2 ~ n2956, f2 ~ n7801312 (18/4).
Fperson :- f2 ~ n2956, f2 ~ n8536242, f3 ~ n6992023 (7/0).
Fperson :- f1 ~ n5303, f1 ~ n8712722, f3 ~ n13067 (21/5).
Fperson :- f2 ~ n3135, f2 ~ n8056193, f2 ~ n8580113 (10/0).
Fperson :- f2 ~ n2956, f2 ~ n8045071, f2 ~ n8528433 (7/0).
Fperson :- f2 ~ n3135, f2 ~ n7956863 (10/1).
Fperson :- f3 ~ n3135, f1 ~ v2154903, f3 ~ n7902627 (8/0).
Fperson :- f2 ~ n3135, f2 ~ n8536242, f2 ~ n7904081, f2 !~ n8477015 (8/4).
Fperson :- f3 ~ n3135, f3 ~ n8536242, f2 ~ n2847188 (4/0).
Fperson :- f1 ~ v730155, f3 ~ n7893547 (7/4).
Fperson :- f3 ~ n2956, f3 ~ n8648356, f3 !~ n8619573, f2 ~ n5294998 (5/0).
Fperson :- f2 ~ n7832025 (16/1).
Fperson :- f2 ~ n2956, f2 ~ n7960124, f2 ~ n7961997 (6/0).
Fperson :- f2 ~ n3135, f2 ~ n8853746 (6/0).
Fperson :- f2 ~ n3135, f2 ~ n3272277 (6/0).
Fperson :- f1 ~ n5303, f1 ~ n7904081, f1 ~ n2883498 (5/0).
Fperson :- f1 ~ v730155, f2 ~ n8163608 (3/0).
Fperson :- f1 ~ n5555437 (11/0).
Lperson :- f1 ~ n2956, f1 ~ n7953417, f1 !~ n8032285 (55/14).
Lperson :- f2 ~ n3135, f2 ~ n7902627, f2 !~ n7899836, f1 ~ n23704 (16/2).
Lperson :- f1 ~ n2956, f1 ~ n8523465 (19/0).
Lperson :- f1 ~ n2956, f1 ~ n8704783, f1 ~ n16840 (14/1).
Lperson :- f1 ~ n2956, f2 ~ n2956, f2 ~ n8563620 (10/4).
Lperson :- f2 ~ n3135, f2 ~ n8833984, f2 !~ n25413, f2 !~ n24936, f2 ~ n2656025 (11/0).
Lperson :- f1 ~ n5708379, f3 ~ NoWordKey (15/3).
Lperson :- f1 ~ n2956, f1 ~ n8056193, f1 ~ n8580113 (8/0).
Lperson :- f2 ~ n2956, f1 ~ n1742, f2 ~ n8833984, f2 !~ n22634 (18/8).
Lperson :- f1 ~ n2956, f1 !~ n2664, f1 ~ n8391286 (6/2).
Lperson :- f1 ~ n2956, f1 ~ n8536242, f1 ~ n8629644, f1 !~ n8024371 (7/1).
Lperson :- f1 ~ n5708379, f1 !~ n11478129, f1 ~ n11455258, f1 ~ n11474597 (6/2).
Lperson :- f2 ~ n3135, f2 ~ n7902627, f2 !~ n8619573, f2 !~ n8105326, f1 ~ n1742, f2 !~ n11413 (8/2).
Lperson :- f2 ~ n3135, f2 ~ n8536242, f2 ~ n8200333, f3 !~ v1454310 (5/0).
Lperson :- f1 ~ n7832025 (15/2).
Lperson :- f1 ~ n3135, f1 ~ n3272277 (6/0).
Uperson :- f3 ~ v599108, f1 ~ NoWordKey, f2 !~ n7899836, f2 ~ n1742, f2 !~ n7911996 (17/1).
Uperson :- f3 ~ v594545, f2 ~ n7893547, f2 !~ n16993 (25/8).
Uperson :- f3 ~ v730155, f1 ~ NoWordKey, f2 !~ n13067 (32/26).

Uperson :- f3 ~ v1868026, f3 ~ v802014, f1 !~ n1742, f2 ~ n7898963 (6/1).
Uperson :- f2 ~ n2956, f2 ~ n8094623 (51/8).
Udate :- f2 ~ n12830667, f2 !~ n1742, f1 !~ a790152 (233/3).
Udate :- f1 ~ n11497797 (90/6).
Udate :- f1 ~ NoWordKey, f3 ~ NoWordKey (100/80).
Udate :- f2 ~ n12824116, f2 !~ n12786206 (30/3).
Morganisation :- f3 ~ n6643140, f1 ~ n22113, f2 ~ n21905 (77/1).
Morganisation :- f1 ~ n1742, f2 ~ n22113, f3 ~ n16993, f1 ~ n7491601 (80/0).
Morganisation :- f1 ~ n1742, f3 ~ n22113, f2 ~ n1742, f1 !~ n2664, f1 ~ n7116686 (67/3).
Morganisation :- f3 ~ n6643140, f1 ~ n1742, f2 ~ n27447 (88/6).
Morganisation :- f1 ~ n1742, f2 ~ n22634, f2 ~ n241678, f1 ~ n7905970 (41/0).
Morganisation :- f3 ~ n6643140, f1 ~ n1742, f2 ~ n1742, f3 !~ n148506, f1 ~ n19046, f1 !~ n7058546 (15/2).
Morganisation :- f3 ~ n22634, f1 ~ n1742, f2 ~ n16993, f3 !~ n6636576 (18/7).
Morganisation :- f3 ~ n6643140, f2 ~ n22634, f1 ~ n6643140, f1 ~ n6893418 (7/0).
Morganisation :- f3 ~ n22634, f2 ~ n22634, f2 ~ n7782241, f3 ~ n6696044 (17/1).
Morganisation :- f3 ~ n22634, f1 ~ n1742, f2 ~ n6790797 (6/0).
Morganisation :- f3 ~ n22634, f1 ~ n13067, f2 ~ n6791040 (8/2).
Morganisation :- f3 ~ n22634, f1 ~ n13067, f3 ~ n6686439, f3 !~ n8395191 (7/3).
Morganisation :- f1 ~ n1742, f2 ~ n19046, f2 ~ n5270807, f1 ~ n7062038 (7/0).
Morganisation :- f3 ~ n22634, f1 ~ n22634, f3 ~ n6746043 (19/5).
Morganisation :- f2 ~ n1742, f1 ~ n19046, f1 ~ n7516129 (6/1).
Morganisation :- f3 ~ n22634, f2 ~ n1742, f3 ~ n6694803 (6/1).
Morganisation :- f3 ~ n16993, f3 ~ n7593134, f2 ~ n1742 (17/1).
Morganisation :- f1 ~ n8501671, f2 ~ n22634 (3/0).
Morganisation :- f3 ~ n22634, f3 ~ n310023, f2 ~ n4099891, f3 ~ n11340514 (5/0).
Forganisation :- f3 ~ n6683510, f2 ~ n1742, f3 ~ n6741150, f3 !~ n16840 (62/1).
Forganisation :- f3 ~ n22634, f2 ~ n6683510, f2 ~ n6748862 (32/5).
Forganisation :- f3 ~ n22634, f2 ~ n1742, f3 ~ n3197306, f2 !~ n22634 (36/5).
Forganisation :- f3 ~ n22634, f2 ~ n1742, f3 ~ n8416837 (21/2).
Forganisation :- f3 ~ n22634, f2 ~ n6683510, f3 ~ n6790797, f1 !~ a2650604 (50/3).
Forganisation :- f2 ~ n1742, f3 ~ n1742, f3 !~ n13067, f2 ~ n6671151 (23/1).
Forganisation :- f2 ~ n7491601, f3 ~ n12792491 (31/0).
Forganisation :- f3 ~ n1742, f2 ~ n6992023, f3 ~ n19046, f2 !~ n7130102, f3 ~ n22634 (10/0).
Forganisation :- f2 ~ n1742, f3 ~ n19046, f2 ~ n7430143 (64/43).
Forganisation :- f3 ~ n22634, f2 ~ n3135, f3 ~ n6683510, f3 !~ n25413, f3 !~ n16993, f1 !~ n5303, f2 !~ n7895781 (61/12).
Forganisation :- f3 ~ n1742, f2 ~ n1742, f3 ~ n12543232, f1 !~ n24936 (11/4).
Lorganisation :- f2 ~ n6643140, f1 ~ n22634, f3 ~ n1742 (68/18).
Lorganisation :- f2 ~ n6643140, f1 ~ n1742, f2 ~ n6710350, f1 ~ n6802831 (24/2).
Lorganisation :- f2 ~ n6643140, f1 ~ n1742, f2 ~ n2771586, f2 ~ n8575577 (16/0).
Lorganisation :- f2 ~ n6643140, f1 ~ n1742, f2 ~ n6882991 (49/24).
Lorganisation :- f2 ~ n6643140, f1 ~ n1742, f2 ~ n3197306, f1 !~ n22634, f2 !~ n11246282 (36/5).
Lorganisation :- f2 ~ n22634, f1 ~ n6683510, f1 !~ n1742, f2 ~ n12786206 (26/0).
Lorganisation :- f2 ~ n22634, f1 ~ n5540028 (86/2).
Lorganisation :- f2 ~ n22634, f1 ~ n6683510, f2 ~ n6790797 (27/7).
Lorganisation :- f2 ~ n6643140, f2 ~ n6746043, f1 ~ n6710350 (15/1).
Lorganisation :- f2 ~ n22634, f1 ~ n1742, f2 ~ n6850179 (10/2).
Lorganisation :- f2 ~ n6683510, f2 ~ n6746043, f2 ~ n833292 (11/5).
Lorganisation :- f2 ~ n22634, f1 ~ n1742, f2 ~ n2533363 (16/4).
Lorganisation :- f2 ~ n22634, f2 ~ n6696044, f1 ~ n13067, f2 !~ n2956, f1 !~ n15787 (13/6).
Lorganisation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n12345618, f3 !~ n310023 (26/2).

Lorganisation :- f2 ~ n22634, f1 ~ n16993, f2 ~ n310023, f1 ~ n4099891 (8/4).
Lorganisation :- f2 ~ n22634, f2 ~ n6694803 (8/0).
Lorganisation :- f2 ~ n1742, f1 ~ n1742, f1 ~ n7162964, f2 ~ n15787 (19/1).
Lorganisation :- f2 ~ n1742, f1 ~ n1742, f2 ~ n1290467 (8/0).
Lorganisation :- f2 ~ n22634, f2 ~ n85621 (7/4).
Lorganisation :- f2 ~ n22634, f1 ~ n22113, f2 ~ n6686439 (6/2).
Ulocation :- f2 ~ n7111224, f2 ~ n6992023, f2 ~ n7159610 (103/3).
Ulocation :- f2 ~ n7111224, f2 ~ n7013143, f3 !~ n22634, f3 !~ n3252432, f1 !~ n13067 (130/5).
Ulocation :- f2 ~ n6992023, f2 ~ n6899919, f3 !~ n7582157, f3 !~ n16840 (64/3).
Ulocation :- f2 ~ n6992023, f2 ~ n7062038, f3 !~ n1742 (63/5).
Ulocation :- f2 ~ n6992023, f2 ~ n7130825, f2 !~ n14223 (110/24).
Ulocation :- f2 ~ n6992023, f2 ~ n15787 (41/8).
Ulocation :- f2 ~ n6992023, f2 ~ n7017569, f1 !~ n1742, f2 !~ n7030738, f2 !~ n20595, f2 !~ n7524388, f2 !~ n16993, f2 !~ n7341405 (54/11).
Ulocation :- f2 ~ n1742, f2 ~ n6992023, f2 ~ n7034213, f2 !~ n6772247, f3 !~ n1742, f1 !~ n1742, f3 !~ n11800524, f2 !~ n7758560 (189/13).
Ulocation :- f2 ~ n7667106, f2 ~ n7188766 (20/3).
Ulocation :- f2 ~ n1742, f2 !~ n2664, f2 !~ n4911, f2 ~ n7623806 (18/1).
Ulocation :- f2 ~ n7040158, f2 ~ n7138810, f2 !~ n5303, f2 !~ n7143957 (13/4).
Ulocation :- f2 ~ n1742, f2 !~ n13067, f2 ~ n7700297, f2 !~ n11450705 (18/2).
Ulocation :- f2 ~ n1742, f2 ~ n7673764, f3 !~ n6961162, f3 !~ n2338751 (18/0).
Ulocation :- f2 ~ n7040158, f2 ~ n12363793 (4/0).
Ulocation :- f2 ~ n1742, f2 ~ n7667106, f2 ~ n7622212 (5/4).
Ulocation :- f2 ~ n9100610, f3 !~ n13067 (6/0).
Ulocation :- f2 ~ n793354 (5/1).
Ulocation :- f1 !~ n1742, f2 ~ n1818808 (3/0).
Ulocation :- f2 ~ n6899919, f3 ~ n22634 (6/0).
Ulocation :- f2 ~ n3176620 (4/0).
Ulocation :- f2 ~ a2796035 (4/0).
Ulocation :- f1 !~ n1742, f2 ~ n8120742 (3/0).
Ulocation :- f2 ~ n7667106, f2 ~ n7552184 (3/1).
Uorganisation :- f2 ~ n6789273 (80/4).
Uorganisation :- f1 ~ n12829061, f3 ~ n21905, f2 !~ n13067 (43/2).
Uorganisation :- f3 ~ n13067, f3 ~ n8337045, f2 ~ n8655657 (23/5).
default 0 (80480/5404).

========== summary ==========

Train error rate: 6.77% +/- 0.08% (91626 datapoints) <<
Hypothesis size: 193 rules, 793 conditions
Learning time: 2710.72 sec

C.4 MOLI MENE V9

Recall that the complex features used for the maximum entropy model of MOLI MENE V9 are the same than V6 uses, plus the generalisation of these features into less complex features according to the procedure presented in Mikheev (1998). Therefore, this new "rules" do not predict a class. Table C.1 on page 220 can be consulted for the meaning of the features mentioned in the following short extension.

Rule 17:- f9 ~ NP.
Rule 18:- f11 ~ CONJ.

C.5 MOLI MENE V10

The generalisation of the complex features of MOLI MENE V7 produces the features considered by MOLI MENE V10. Table C.2 on page 221 shows the relation between the actual features and the names used in the following added features.

Rule 222:- f2 ~ $, f2 !~ PRP.
Rule 223:- f8 ~ NNP, f2 ~ NNP.
Rule 224:- f8 ~ NNP.
Rule 225:- f2 ~ CD, f5 ~ CD.
Rule 226:- f2 ~ NNP, f8 ~ NNP.
Rule 227:- f4 ~ last.
Rule 228:- f5 ~ CD.
Rule 229:- f6 ~ n12811110, f5 ~ NN.
Rule 230:- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP.
Rule 231:- f2 ~ NNP.
Rule 232:- f2 ~ NNP, f5 ~ NNP, f8 !~ NNP.
Rule 233:- f2 ~ NNP, f5 ~ NNP.
Rule 234:- f5 ~ NNP, f2 ~ NNP.
Rule 235:- f2 ~ NNP, f5 ~ NNPS.
Rule 236:- f5 ~ NNP.
Rule 237:- f5 ~ NNP, f8 ~ NNP.
Rule 238:- f2 ~ NNP, f8 ~ P_COMMA, f5 ~ NNP.
Rule 239:- f5 ~ NNP, f8 !~ NNP, f2 !~ NNP.
Rule 240:- f5 ~ NNP, f8 ~ NNPS.
Rule 241:- f5 ~ NNP, f2 ~ NNP, f8 !~ NNP.
Rule 242:- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT.
Rule 243:- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f9 ~ n6788854.
Rule 244:- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ CD.
Rule 245:- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ DT, f6 !~ n5938672.
Rule 246:- f5 ~ NNP, f8 ~ NNP, f2 !~ NNP, f2 ~ IN.
Rule 247:- f5 ~ NNP, f8 !~ NNP, f2 ~ IN.
Rule 248:- f5 ~ NNP, f8 !~ NNP, f2 ~ P_COMMA.
Rule 249:- f5 ~ NNP, f8 !~ NNP.
Rule 250:- f5 ~ NNP, f2 !~ NNP, f8 !~ NNP, f8 ~ NN.
Rule 251:- f5 ~ NNP, f2 !~ NNP.
Rule 252:- f2 ~ NNP, f8 !~ NNP.

C.6 MOLI MENE V11

Applying Mikheev's (1998) generalisation to MOLI MENE V8 features, V11 is obtained. Table C.3 on page 226 presents the names used by the Ripper algorithm for each atomic feature mentioned in the following list of complex features added.

Rule 194:- f2 ~ n12889670.
Rule 195:- f2 ~ n5708379.
Rule 196:- f3 ~ n12786206.
Rule 197:- f3 ~ n12786206, f3 ~ n12811259.
Rule 198:- f3 ~ n12786206, f1 ~ n1742.
Rule 199:- f2 ~ n1742, f1 ~ n19046.

Rule 200:- f2 ~ n1742.
Rule 201:- f2 ~ n1742, f1 ~ n1742.
Rule 202:- f2 ~ n3135.
Rule 203:- f2 ~ n2956.
Rule 204:- f1 ~ v730155.
Rule 205:- f3 ~ NoWordKey.
Rule 206:- f3 ~ n22634, f2 ~ n1742.
Rule 207:- f2 ~ n22634.
Rule 208:- f2 ~ n22634, f1 ~ n1742.
Rule 209:- f2 ~ n7667106.
Rule 210:- f1 !~ n1742.
Rule 211:- f1 ~ n1742.

Appendix D

Biasing LexMENE: details

D.1 The constituent pattern information entity

The weight on a similarity arc that connects two constituent patterns corresponds to the sum of the estimated similarity of each chunk tag in the best alignment that the Needleman-Wunsch-Sellers (NWS) algorithm (Needleman and Wunsch 1970, Sellers 1974) can obtain. Algorithm D.1 presents the similarity function for calculating the similarity between two chunk tags in the patterns.

Basically, the similarity is zero if the chunk tags are different. Otherwise, the position of the closest constituent to the focus is determined and a similarity is assigned according to this distance. BiLexMENE consider two patterns more similar if they share a chunk tag closer to the focus. Accordingly, the score 5.50 is assigned to the position next to the focus, then 3.25, then 2.25 and finally 1.00 to the position farthest from the focus on either extremity. These weights have been selected so that the maximum similarity between two constituent patterns is 12.

If the chunk tags being scored are not in the same position, a penalty (Penalty 1) is applied in function of the distance between the different positions. In addition to this, a second penalty is applied (Penalty 2) to acknowledge this difference. Thus, a match between constituents in the same position gets a higher score than a match between constituents in contiguous positions, which in turn is higher than the score assigned to a match between —for example— the closest and farthest constituents from the focus.

In addition to this, if the matched constituents are optional, the score is further penalised (Penalty 3) as this can be thought as a less important match. BiLexMENE considers as optional constituents marked as adjectival phrases (*ADJP*), adverbial phrases (*ADVP*) and verb particles (*PTR*).

There is one observation that should be commented here. When one of the constituents is null —at a sentence's boundary— it matches constituents tagged as *CONJ*, a tag used

Algorithm D.1: Calculation of the similarity between two constituents for the alignment of two constituent patterns. The initial weights are: 5.50, 3.25, 2.25 and 1.00, according to the distance from the focus constituent. By default, *CPPenalty1* = 1.0, *CPPenalty2* = 0.5, *CPPenalty3* = 0.5 and *CPPenalty4* = 0.5.

Input: $const_1$ and $const_2$, the two constituents to be compared
Output: a score of the similarity between the two constituents

```
1: procedure GET-SIMILARITY-TWO-CONSTITUENT-LABELS(const1, const2)
2:     if the constituents' labels are the same then
3:         sim = the weight corresponding to the closer position to the focus
4:         if the constituents are not in the same position then
5:             sim = max(0, sim − CPPenalty1 · distance)
6:             sim = sim · CPPenalty2
7:         end if
8:         if the constituents are optional then        ▷ i.e. ADJP, ADVP or PTR
9:             sim = sim · CPPenalty3
10:        end if
11:        return sim
12:    end if
13:    if one of the constituents is null and the other one is labelled CONJ then
14:        sim = the weight corresponding to the closer position to the focus
15:        sim = sim · CPPenalty4
16:        if the constituents are not in the same position then
17:            sim = max(0, sim − CPPenalty1 · distance)
18:            sim = sim · CPPenalty2
19:        end if
20:        return sim
21:    end if
22:    return 0.0
23: end procedure
```

by the MBSP parser to recognise conjunctions —such as the words *and*, *but* and *or*— but that has been extended here to manage also punctuation marks —such as periods, commas, semicolons, quotes, etc.— that link two pieces of text. In this case however, the score is penalised (Penalty 4) in addition to the penalties mentioned above — except Penalty 3, of course.

Although the penalty values used by biLexMENE could be free parameters, in the implementation discussed here they have been fixed to the values 1.0, 0.5, 0.5 and 0.5 respectively on both left and right constituent patterns. These values showed to produced the desired effects when tested with a small number of examples.

D.2 The binary lexical pattern information entity

The similarity weight that labels an arc between two binary lexical patterns corresponds to the sum of the similarity of each binary lexical feature in the best alignment of those patterns (obtained by the NWS algorithm). The similarity function to obtain the individual similarity scores between two binary lexical features is presented in algorithm D.2. In few words, the algorithm gets the similarity of the features on the basis of the corresponding value in one of three tables of similarity weights, according to the type of both binary lexical features.

Algorithm D.2: Calculation of the similarity between two orthographic features for the alignment of two binary lexical patterns. By default, $BLPPenalty1 = \frac{1}{8}$.

Input: blf_1 and blf_2, the two binary lexical features to be compared
Output: a score of the similarity between the two features

procedure GET-SIMILARITY-TWO-BINARY-LEXICAL-FEATURES(blf_1, blf_2)
  if one feature is left context **and** the other is right context **then**
    return 0.0
  end if
  weight ← 1.0
  if either blf_1 **or** blf_2 is a context feature **then**
    weight ← *BLPPenalty1*
  end if
  if both blf_1 **and** blf_2 are number-type **then**
    sim ← the corresponding similarity score from the following table

	,dg	-dg	.dg	/dg	2dg	4dg	:dg	adg	odg	vdg
,dg	12.0	4.0	8.0	4.0	8.0	8.0	4.0	2.0	8.0	3.0
-dg	4.0	12.0	4.0	8.0	4.0	4.0	4.0	2.0	4.0	3.0
.dg	8.0	4.0	12.0	4.0	4.0	4.0	4.0	2.0	4.0	3.0
/dg	4.0	8.0	4.0	12.0	4.0	4.0	4.0	2.0	4.0	3.0
2dg	8.0	4.0	4.0	4.0	12.0	8.0	4.0	2.0	8.0	3.0
4dg	8.0	4.0	4.0	4.0	8.0	12.0	4.0	2.0	8.0	3.0
:dg	4.0	4.0	4.0	4.0	4.0	4.0	12.0	2.0	4.0	3.0
adg	2.0	2.0	2.0	2.0	2.0	2.0	2.0	12.0	2.0	2.0
odg	8.0	4.0	4.0	4.0	8.0	8.0	4.0	2.0	12.0	3.0
vdg	3.0	3.0	3.0	3.0	3.0	3.0	3.0	2.0	3.0	12.0

    return *weight* · *sim*
  end if
  if both blf_1 **and** blf_2 are word-type **then**
    sim ← the corresponding similarity score from the following table

	acp	adg	cpp	icp	mcp	ucp
acp	12.0	2.0	6.0	4.0	8.0	4.0
adg	2.0	12.0	2.0	2.0	2.0	2.0
cpp	6.0	2.0	12.0	6.0	6.0	6.0
icp	4.0	8.0	4.0	12.0	4.0	4.0
mcp	8.0	4.0	4.0	4.0	12.0	8.0
ucp	8.0	4.0	4.0	4.0	8.0	12.0

    return *weight* · *sim*
  end if
  if both blf_1 **and** blf_2 are other-type **then**
    sim ← the corresponding similarity score from the following table

	null	adg	mix	ncp
null	12.0	0.0	0.0	6.0
adg	0.0	12.0	8.0	0.0
mix	0.0	8.0	12.0	4.0
ncp	6.0	0.0	4.0	12.0

    return *weight* · *sim*
  end if
  return 0.0
end procedure

For this effect, three types have been defined: the number-type, which gathers binary lexical features shown by numeric tokens, such as *digits with commas* (*,dg*), *two-digits number* (*2dg*), etc.; the word-type, which groups binary lexical features that are presented by words, such as *all capitals* (*acp*), *mixed capitalisation* (*mcp*) or *uncapitalised*

(*ucp*); and the other-type for any other binary lexical feature in use, such as *symbols* (*ncp*) and *mixed characters* (*mix*).

Algorithm D.2 changes the score retrieved from the similarity tables in two ways. First, if at least one of the binary lexical features is not from a focus token, then the similarity weight is penalised. And secondly, if both binary lexical features are contextual, and one is from the left of the focus and the other is from the right side of the focus constituent, then the similarity is reduced to zero.

As earlier, the penalty applied was determined on the basis of testing different values on a selected set of examples until the scores obtained were considered sensible. In this case, this penalty was fixed to one eighth of the value in the tables.

D.3 The lexical pattern information entity

The similarity weight associated with each arc between lexical patterns corresponds to the sum of the similarity of the lexical features —i.e. words or tokens— in the best alignment that the NWS algorithm can obtain for the patterns. The similarity function to obtain the similarity score between two lexical features in the patterns is presented in algorithms D.3 through D.7.

Algorithm D.3: Calculation of the similarity between two lexical features (words/tokens). By default, *LPPenalty1* $= \frac{1}{8}$.

Input: lf_1 and lf_2, the two lexical features to be compared
Output: a score of the similarity between the two features

```
1: procedure GET-SIMILARITY-LEXICAL-FEATURES(lf1, lf2)
2:     if one feature is left context and the other is right context then
3:         return 0.0
4:     end if
5:     weight ← 1.0
6:     if either lf1 or lf2 is a context feature then
7:         weight ← LPPenalty1
8:     end if
9:     if either lf1 or lf2 is null then
10:        return weight· GET-SIMILARITY-NULL-LEXICAL-FEATURES(lf1, lf2)
11:    end if
12:    if both lf1 and lf2 have known meanings then
13:        return weight· GET-SIMILARITY-LEXICAL-FEATURES-WITH MEANINGS(lf1, lf2)
14:    end if
15:    if both lf1 and lf2 are tokens without known meaning then
16:        return weight· GET-SIMILARITY-TOKENS-WITHOUT-MEANINGS(lf1, lf2)
17:    end if
18:    return 0.0
19: end procedure
```

In few words, the main source of information for comparing two lexical features is their *meaning*. However, sometimes a comparison based on meanings is not possible: when

(at least) one of the lexical features is from the context of the beginning or ending of a sentence, in which case the feature is null; or when the features are tokens without known meanings.

This last case occurs because WordNet manages information on open class lexemes only (i.e. nouns, verbs, adjectives and adverbs), leaving automatically out determiners, prepositions, conjunctions, etc. In addition to this, it is unrealistic to expect that WordNet's lexical database is complete, specially nouns, and in particular proper nouns. When meanings are not available, the similarity function makes use of less informed, more lexically-oriented characteristics to estimates their similarity.

Algorithm D.3 changes this normal calculation in two occasions: if the two lexical features are from the context of a focus constituent, but one is from the left context and the other is from the right context, then their similarity is reduced to zero; and when the matching is occurring between a context lexical feature and a focus one, their similarity is penalised. As earlier, this penalty has been fixed on the basis of trying different values in a small set of selected examples and choosing the value that produces the desired effect best. In this case, this penalty has been fixed to 0.125.

Algorithm D.4: Calculation of the similarity between two lexical features (words/tokens) when at least one of them is null. By default, *LPBothNullFeatureSim* = 12.0, *LPNullFeatureStopSymbolSim* = $\frac{1}{2}$ *LPBothNullFeatureSim* and *LPNullFeatureSemistopSymbolSim* = $\frac{1}{2}$ *LPBothNullFeatureSim*.

Input: lf_1 and lf_2, the two lexical features to be compared,
of which at least one is null
Output: a score of the similarity between the two features

```
1: procedure GET-SIMILARITY-NULL-LEXICAL-FEATURES(lf1, lf2)
2:     if both lf1 and lf2 are null then
3:         return LPBothNullFeaturesSim
4:     end if
5:     if the not-null feature occurs in a constituent labelled CONJ then
6:         if the not-null feature is a stop symbol then        ▷ i.e. . , ; : ? !
7:             return LPNullFeatureStopSymbolSim
8:         end if
9:         if the not-null feature is a semi-stop symbol then   ▷ i.e. " " ' ' ( )
10:            return LPNullFeatureSemistopSymbolSim
11:        end if
12:    end if
13:    return 0.0
14: end procedure
```

When one or both of the lexical features are null, their similarity is estimated as presented in algorithm D.4. Essentially, three cases are considered as valid matches and each one of them is associated with a fixed similarity value. The matches considered include: both lexical features are null, which at this stage are known to come from the same contextual side; one of the features is null and the other is a punctuation mark that indicates the beginning of a new idea in the text; and one of the features is null and

the other is a token that (more weakly) suggests a possible change in the discussion.

In the version of biLexMENE presented here, the symbols considered as strong indicators of change in the discourse are periods, commas, semicolons, colons, question marks and exclamation marks. These symbols are referred to as *stop symbols*. Similarly, simple and double quotes and parentheses are considered as weak indicators of change in the discourse and, consequently, called *semi-stop symbols*. Note that these symbols are normally found in constituents labelled *CONJ*.

The similarity values has been fixed as previously. The best combination of values found is to consider the match of two null lexical features as a perfect match —i.e *LPBothNullFeatureSim* = 12.0— and matches with stop and semi-stop symbols as half-good matches, that is setting *LPNullFeatureStopSymbolSim* = 6.0 and *LPNullFeatureSemistopSymbolSim* = 6.0.

Algorithm D.5: Calculation of the similarity between two lexical features (words/tokens) with known meanings. By default, *WordNetSameLemmaSim* = 12.0, *WordNetSynonymsSim* = $\frac{3}{4}$ *WordNetSameLemmaSim*, *WordNetSiblingsSim* = $\frac{1}{2}$ *WordNetSameLemmaSim* and *WordNetParentChildSim* = $\frac{1}{4}$ *WordNetSameLemmaSim*.

Input: lf_1 and lf_2, the two lexical features with known meanings to be compared
Output: a score of the similarity between the two features

```
1: procedure GET-SIMILARITY-LEXICAL-FEATURES-WITH-MEANINGS(lf1, lf2)
       ▷ Searches in this algorithm are for every sense in the same lexical category
2:     if lf1 or lf2 does not have a WordNet entry then
3:         get special meanings for the features with unknown meaning
                                ▷ e.g. % means 'percent', 's in a VP means 'is' or 'has'
4:     end if
5:     if lf1 and lf2 have the same WordNet lemma then
6:         return WordNetSameLemmaSim
7:     end if
8:     if lf1 is a WordNet synonym of lf2 then
9:         return WordNetSynonymsSim
10:    end if
11:    sim1 ← 0.0
12:    if lf1 is a WordNet coordinate term of lf2 then
13:        sim1 ← WordNetSiblingsSim
14:    end if
15:    sim2 ← 0.0
16:    if one of the features is a WordNet direct hypernym of the other then
17:        sim2 ← WordNetParentChildSim
18:    end if
19:    return max(sim1, sim2)
20: end procedure
```

Algorithm D.5 shows the procedure to estimate the similarity between lexical features with known meanings, which considers four cases: the features have the same lemma — i.e. they are the same word or differ in their inflections, the features are synonyms, the features are coordinate terms, and the features have a hyponym/hypernym relationship.

As above, each case is associated with fixed values that represent the strength of these semantic relations. These values were selected following a sensible order. Thus, if

two lexical features are essentially the same word, their match is considered perfect (*WordNetSameLemmaSim* = 12.0). Then, the slightly less strong relation of synonymity is set accordingly to a slightly lower value (*WordNetSynonymsSim* = 9.0). The next relation in strength is the sibling relation, which in WordNet's terminology corresponds to coordinate terms. The similarity value for this relation has been defined has half of the perfect match (*WordNetSiblingsSim* = 6.0). Finally, the weakest semantic relations are the superordination and subordination, which in WordNet are called the hypernym and hyponym relations respectively, for which the lowest similarity value is defined (*WordNetParentChildSim* = 3.0).

Two observations about algorithm D.5 should be noticed. First, the similarity function always tries to determine the highest similarity value. Thus, before checking whether the lexical features match in a weaker semantic relationship, it exhausts the possibilities for a stronger relationship with every sense of each feature. And secondly, some extension of WordNet is performed before the comparison starts. This extension allows lexemes that do not have explicit entries in the lexical database to be associated with known meaning. For example, the symbol *%* is associated with the meanings that the word *percent* has in WordNet. This extension can manage several conditions, such as that the lexical feature occurs in a particular type of constituent. The list of lexemes added to lexical features with known meaning is mainly language dependent. Although, task specific entries might be included in this list, this has not been necessary in the version of biLexMENE presented here.

The last possibility for a match is that both lexical features are tokens with no known meanings. Words are discarded here because biLexMENE maintains as words only lexemes that consists of more than one token and have an entry in WordNet's lexical database. Thus, there are no words without known meaning. Example of words are *prime_minister*, *executive_officer*, *pearl_harbor*, etc. — for which valuable semantic information would be lost if analysed from the component tokens individually.

Algorithm D.6 describes the estimation of the similarity between tokens without known meanings. The first step is an attempt at making strings that contain numbers less variable. This process consists of replacing each digit in the token for the character 'D', and then replacing ordinal endings for the generic string 'TH'. For example, the strings *530*, *10/31/97*, *221*, *21st* and *8/12/98* will become *DDD*, *DD/DD/DD*, *DDD*, *DDTH* and *D/DD/DD* respectively. The resulting strings, called *texts* in the algorithm, are the ones considered in the following comparisons.

The highest match possible is that the two texts are the same. If this happens, algorithm D.6 returns the constant *LPSameTextSim*, which has been set to the maximum value of 12 in this initial version of biLexMENE. Otherwise, the algorithm checks whether there are defined *special matches* for the texts. Special matches include both tokens that are related to tokens that do not have an entry in WordNet's lexical database

Algorithm D.6: Calculation of the similarity between two lexical features that are tokens without meanings (words always have meanings). By default, *LPSameTextSim* = 12.0, *LPBothStopSymbolsSim* = $\frac{1}{2}$ *LPSameTextSim*, *LPBothSemistopSymbolsSim* = $\frac{1}{2}$ *LPSameTextSim* and *LPStopAndSemistopSymbolsSim* = 0.0.

Input: $token_1$ and $token_2$, the two tokens without known meanings to be compared
Output: a score of the similarity between the two tokens

```
1: procedure GET-SIMILARITY-TOKENS-WITHOUT-MEANINGS(token_1, token_2)
2:     text_1 ← GET-TEXT-NUMBER-PATTERN(t_1)
3:     text_2 ← GET-TEXT-NUMBER-PATTERN(t_2)
                          ▷ e.g. 'nuts4nuts' ⇒ 'nutsDnuts', '3-11-04' ⇒ 'D-DD-DD'
4:     if text_1 = text_2 then
5:         return LPSameTextSim
6:     end if
7:     if text_1 and text_2 have an special match then
8:         return the similarity defined for the special match
                                          ▷ e.g. '&' ~ 'and', 'DD-DD' ~ 'DD/DD'
9:     end if
10:    sim ← GET-MORPHOLOGICAL-SIMILARITY(text_1, text_2)
11:    if sim > 0.0 then
12:        return sim
                                  ▷ e.g. 'Jackson' ~ 'Johnson', 'D,DDD' ~ 'DD,DDD'
13:    end if
14:    if both tokens are stop symbols then
15:        return LPBothStopSymbolsSim
16:    end if
17:    if both tokens are semi-stop symbols then
18:        return LPBothSemistopSymbolsSim
19:    end if
20:    if one token is a stop symbol and the other is a semi-stop symbol then
21:        return LPStopAndSemistopSymbolsSim
22:    end if
23:    return 0.0
24: end procedure
```

—e.g. the word *and* and the *&* symbol— and language-dependent/domain-dependent relations which might be missed or incorrectly estimated by a morphological analyser — such as the texts *DD-DD-DD*, *DD/DD/DD* and *DD:DD:DD*, which would be considered to have the same morphological similarity even though the former two patterns could be thought closer as both are used to express dates, whereas the latter is used to express time. These special matches —which are introduced in biLexMENE as a list that can be easily adapted to other languages and domains— include the similarity score in their definition, which are the scores returned by the algorithm in these cases.

When the texts being compared are not equal and algorithm D.6 cannot find special matches for them, their similarity is estimated by applying a morphological analysis as presented in algorithm D.7. This algorithm starts by ensuring that the morphological analysis has a high probability of being significant. This involves checking that the texts are long enough to show some meaningful prefix or suffix —which in this implementation corresponds to four or more characters— and that a common prefix or suffix might be indicating the same linguistic phenomenon — i.e. the texts' lengths suggest that a

Algorithm D.7: Calculation of the similarity between two texts according to the lengths of their common prefixes and common suffixes.

Input: $text_1$ and $text_2$, the texts to be compared
Output: a score of the similarity between the two texts

```
1: procedure GET-MORPHOLOGICAL-SIMILARITY(text_1, text_2)
2:     if length(text_1) < 4 and length(text_2) < 4 then
3:         return 0.0
4:     end if
5:     if one text is twice as long as the other then
6:         return 0.0
7:     end if
8:     leftsim ← length(common prefix)/length(longest text)
9:     rightsim ← length(common suffix)/length(longest text)
10:    if length(common prefix) < 3 then
11:        leftsim ← 0.0
12:    end if
13:    if length(common suffix) < 3 then
14:        rightsim ← 0.0
15:    end if
16:    if the common prefix and the common suffix overlap then
17:        min(leftsim, rightsim) ← 0.0
18:    end if
19:    return leftsim + rightsim
20: end procedure
```

common prefix or suffix is indeed a common prefix or suffix rather than a coincidence, which is determined by checking that one text is not twice as long as the other one.

If this conditions are complied, the algorithm determines whether the texts have a common prefix or/and a common suffix. These common bits are ignored if they do not have a length higher than two characters. In addition to this, if the texts have both a common prefix and a common suffix that overlaps, algorithm D.7 considers only one of them. These heuristics and fixed values —as well as the ones explained below— have been obtained following the procedure applied earlier, thus analysing a reduced number of examples until the desired output —i.e. obtaining similarity values mostly when the tokens are morphologically related— was observed.

The morphological similarity corresponds to the sum of the lengths of the common prefix and common suffix found, divided by the length of the longest text in analysis.

When the morphological analysis yields null similarity, algorithm D.6 makes a last attempt at finding a relation: it checks whether the texts in analysis are a combination of stop and semi-stop symbols. According to this, the procedure returns a fixed similarity value for matches between two stop symbols ($LPBothStopSymbolsSim = 6.0$), two semi-stop symbols ($LPBothSemistopSymbolsSim = 6.0$) and a stop and a semi-stop symbol ($LPStopAndSemistopSymbolsSim = 0.0$).

D.4 Reducing the size of the problem

Because of the computational costs of retrieving similar cases, an attempt to reduce the number of queries that need to be processed was introduced in biLexMENE. The initial idea was to train a binary classifier to decide whether a query is *relevant* or *irrelevant*, which should try to maximise recall but keeping, at the same time, a low number of spuriously checked queries — i.e. irrelevant queries classified as relevant. However, the parametrisation of such classifier became difficult and soon was clear that a heuristic to filter the classifier's decisions was needed. Algorithm D.8 presents this heuristic procedure.

Basically, a query should be processed when it contains a token which might be part of a named entity. The first approach is to look at the frequency of the token and its binary lexical feature with respect to the named entity classes with which they occur. When this pair is not seen or seldom seen, the decision is left to biLexMENE. If the token has a moderate to high frequency, but every single time it occurs outside a named entity, the token is *a priori* ruled out. Otherwise, that is the pair has been seen within named entities, and its frequency is only moderate, it is consider safer to leave the classification to biLexMENE. When a pair is frequent and has been seen as part of named entities in at least 30% of the time, the token —and consequently, the query— is passed on to biLexMENE. Finally, if the frequent pair is mostly seen outside named entities, a classifier is applied and the token is discarded only if this classifier determines that its probability of being part of a named entity is less than 5%.

This heuristic introduces several parameters which were adjusted following a similar approach to estimating the other parameters of biLexMENE: some values were fixed to apply the heuristic on a small set of examples, then the results were analysed to identify mistakes and new values were obtained in an attempt to reduce these errors. After some iterations, a final set of values was obtained which showed to produce the effect sought.

For this implementation, too little frequency was set to less than three times and high frequency to more than 15 events (thus, pairs occurring three to 15 times with a class are considered to have moderate frequency). The above parameters were also set in this way and the values 30% and 5% used above resulted from the best trade off obtained between a high recall and a low number of spurious checks.

The classifier used in algorithm D.8 is not restricted to any particular approach. The only condition that this classifier must meet is that it should estimate the probability that a token is not part of a named entity. Several algorithms were tested for the current implementation of biLexMENE and the one selected corresponds to a PART decision list learner, freely available in the Weikato Environment for Knowledge Analysis (WEKA) (Witten and Frank 2000), which builds partial decision trees in a series of iterations, transforming into a rule the best leaf of each partial tree (Frank and Witten 1998).

Algorithm D.8: The heuristic used to reduce the number of queries to be processed by biLexMENE.

Input: a query
Output: **true** if it is considered likely that the query contains tokens which are part of a named entity; **false** otherwise

```
1: procedure SHOULD-BE-CHECKED(query)
2:     for each token ∈ query do
3:         if SHOULD-BE-CHECKED(token) then
4:             return true
5:         end if
6:     end for
7:     return false
8: end procedure
```

Input: a token
Output: **true** if it is considered likely that the token is part of a named entity; **false** otherwise

```
9: procedure SHOULD-BE-CHECKED(token)
10:    answer ← TRY-DECISION-BASED-ON-FREQUENCIES(token)
11:    if answer ≠ Maybe then
12:        return answer
13:    end if
14:    features ← Create features for the token with context window of [−2, 2]
15:    Apply a classifier on features
16:    if the classifier determines P(not in a named entity | features) > 0.95 then
17:        return No
18:    end if
19:    return Yes
20: end procedure
```

Input: a token
Output: *Yes*, *No* or *Maybe* to whether it is likely that the token is part of a named entity, according to the frequency of the events in which the token takes part

```
21: procedure TRY-DECISION-BASED-ON-FREQUENCIES(token)
22:    blf ← the main binary lexical feature of token
23:    counts ← the number of times that the pair (token, blf) has been seen
24:    if counts < 3 then
25:        return Yes                ▷ The pair has been seen too few times to ignore it
26:    end if
27:    if all counts are registered outside named entities then
28:        return No                 ▷ The pair will probably not be part of named entities
29:    end if
                                     ▷ Therefore the pair has been seen as part of named entities
30:    if counts ≤ 15 then
31:        return Yes                ▷ Too few times to safely discard the token
32:    end if
33:    if less than 70% of the counts are registered outside named entities then
34:        return Yes                ▷ The pair has often been seen as part of named entities
35:    end if
                                     ▷ Most of the time the pair has been seen outside named entities
36:    return Maybe
37: end procedure
```

The features provided to this classifier for a given token are five: lexical, binary lexical at word and token level, composition type and not-a-name features.

The lexical feature corresponds, as in earlier chapters, to the string of the token in lowercase.

The binary lexical features are the values defined in table 4.1 to indicates the writing style used for the lexical feature. Two binary lexical features are used: one indicates the style of the token itself and the other indicates the style used for the word to which the token belongs. For example, the word *Atlanta-based* is composed of three tokens. At token-level, these lexemes fire the binary lexical features *icp* (*capitalised* token), *ncp* (*symbol* token) and *ucp* (*uncapitalised* token), but all of them fire the feature *icp* (i.e. a capitalised word) at word-level.

The next feature is also related to this phenomenon, as *the composition type* feature indicates the place that a token occupies in a composed word. This feature can take four different values: *FT* when the token is starting a composed word, *MT* when the token is in the middle of a composed word, *LT* when the token is ending a composed word or *UT* when the token is the only lexeme in a one-token word.

Finally, the feature *not-a-name* is added to a token when the pair (lexical feature, token-level binary lexical feature) has been seen outside named entities in three or more occasions.

All these features are obtained from a context window which includes the focus token, the two tokens on the left and the two tokens on the right if they exists. It should be clarified that the classifier is trained with context windows obtained for all tokens in the training documents, but applied only on context windows for tokens in the decoding texts that cannot be decided with the most direct approach based on frequencies, as stipulated by algorithm D.8.

D.5 Obtaining final similarities

As explained in section 5.8.3.1, the activation values of constituent patterns are re-calculated to also consider the head word of the constituents. The comparison of head words, described in algorithm D.9, is similar to the calculation of the similarity between lexical features. The main difference is that algorithm D.9 returns activation values in the range $[0, 1]$. Each of these activation values is multiplied by the similarity score calculated for each tag according to their positions in the pattern. In this way, the final activation values of a constituent pattern range from (approximately) 1.3 to 12.

Activation values for lexical patterns also need to be re-calculated. The problem arises when the memorised cases are selected. If this selection is guided only by the *global*

Algorithm D.9: Calculation of the similarity between two head lexical features. By default, *ChunkLabelWeight* $= \frac{1}{4}$, *BinaryLexicalWeight* $= \frac{1}{4}$ and LexicalWeight $= \frac{1}{2}$.

Input: h_1 and h_2, the two constituent heads to be compared
Output: a score of the similarity of the two heads between 0.0 and 1.0

```
1:  procedure GET-SIMILARITY-LEXICAL-FEATURES(h1, h2)
2:      if both h1 and h2 are null then
3:          return 1.0
4:      end if
5:      if either h1 or h2 is null then
6:          if the not-null head is a stop symbol then
7:              return 1.0
8:          end if
9:          return 0.0
10:     end if
11:     lf ← GET-SIMILARITY-LEXICAL-FEATURES(h1.LF, h2.LF)
12:     normalise lf in the range [0, 1]
13:     blf ← GET-SIMILARITY-TWO-BINARY-LEXICAL-FEATURES(h1.BLF, h2.BLF)
14:     normalise blf in the range [0, 1]
15:     cp ← 1.0
                  ChunkLabelWeight · cp+
16:     score ←   BinaryLexicalWeight · blf+
                  LexicalWeight · lf
17:     return score
18: end procedure
```

activation of a case, which combines the individual activation of all its information entities, there is a high risk of obtaining a set of cases that would not cover every lexical feature in the query. In other words, there would not be examples for some tokens in the query to be used in the adaptation phase.

This problem was expected and the original idea to manage it was to use elements available in the CRN models to control the importance of an information entity, namely the relevance weights, in the global activation of a case. Thus, the relevance of the lexical information entity was increased to three times the relevance of the constituent patterns. Nonetheless, and although the problem was slightly reduced, this approach was unsuccessful and it was clear that a more radical solution was needed.

To solved this problem, biLexMENE manages a different ranking of similar cases for each lexical feature in a query. In each ranking, the final activation value of a case is re-estimated by applying the lexical window on the corresponding focus token.

In this way, if a focus token in a case is aligned with a focus token in the query, the strength of both the focus match and the matches of immediately surrounding tokens will be considered in the activation. On the other hand, if a focus token in a case does not match any of the focus token in the query, the activation value of the case will only consider contextual matches for that token.

This approach proved to be quite successful in retrieving training examples for most of the tokens in a query. The activation weights for the context window were homo-

geneously fixed to 1.0. Thus, the relevance of lexical patterns was kept at three times the relevance of constituent patterns, mainly because lexical features have proved —in the previous chapters as well as in the literature— to be strong predictors of named entities. Therefore, a perfect lexical match will have a final activation value of 36.

In order to reduce the size of the sets of cases maintained for each lexical feature of a query, a heuristic restriction is applied to discard cases whose activation values are under 1.5. This value showed to filter out cases that contribute little or nothing towards the classification of individual tokens.

The approach described above also requires individual activation values for binary lexical features. Therefore, an analogous calculation process is performed by applying the binary window to each focus token in a case.

The weights for the binary window were fixed to values 1/3 for features on the left context, 1/2 to features on the right context and 5/6 for the focus feature. This values were determined so that the maximum activation value for a focus feature is 30, which seems to be an appropriate relevance weight for this information entity.

Once more, a heuristic restriction was imposed to ignore cases that fail in obtaining an activation value of at least 1.5, because they were found to contribute too little information for the classification of a specific token.

D.6 Messy details

The description of biLexMENE given in chapter 5 depicts the conceptual modelling of the system. However, the system was quite challenging to build in reality.

The main problem was that the sheer number of information entities, microfeatures, cases, similarity and relevance arcs, etc. made impossible to construct and maintain a CRN structure in the memory of a normal desk computer.

Therefore, the CRN was stored in fragments and processed by a sequence of programs. For instance, there is one program that builds and stores the left constituent pattern of each case; another one that activates the left constituent patterns for each query to be checked; another one that combines these activations with the ones coming from the right pattern, and so on.

However, even after this separation into individual steps, it was not possible to keep in primary memory a sufficiently large portion of the CRN and the set of activated elements —i.e. microfeatures, information entities and cases— for all the queries in a document. Secondary memory —i.e. databases— could not solve the problem either because it turned the system too slow.

Fortunately, the University of York is part of The White Rose Grid (WRG), which is a multi-site computing system that aims to provide high performance computing to its users. In this system, there are available machines with several gigabytes of primary memory — see (WRG 2004) for more details. BiLexMENE, operating in sequential steps, manages to run on the WRG machines using up to 8GB of primary memory.

As expected, running time was a problem too. BiLexMENE needs from six to ten days of continuing processing, depending of the size of the training and the parameters used. Surprisingly, most of this time is consumed in retrieving the similar cases, and the adaptation phase —i.e. the construction of a maximum entropy model for each query— rarely needs more than 20% of the total running time.

Appendix E

Smoothing function

The smoothing function used by BiLexMENE employs three parameters for re-allocating the probabilities to a more uniform distribution, so that every possible tag has a non-zero probability associated. These parameters are:

α	which establishes the importance of each named entity class
β	which indicates the importance of the not-a-name tag
γ	which assign the importance of each tag withing a named entity class

The smoothing parameters controls the uniformity of the resulting distribution: the greater and the more similar the values, the more uniform the resulting distribution will be. The process is straightforward and it is probably easier to be understood with an example.

Suppose there are three named entity classes: location names, organisation names and person names; and the not-a-name class for words which are not part of these named entities.

Now suppose a maximum entropy model gives the following probability distribution in BIO notation: *{I-organisation=0.2, I-person=0.3, not-a-name=0.5}*, which needs to be smoothed with parameters $\alpha = 0.25$, $\beta = 0.25$ and $\gamma = 0.5$.

The first step is to smooth the distribution among tags. The original distributions are: *{location: B=0, I=0; organisation: B=0, I=0.2; person: B=0, I=0.3}*. Now the function adds the γ parameter to each probability and then normalises them by named entity class. The resulting distributions are: *{location: B=0.5, I=0.5; organisation: B=0.41667, I=0.58333; person: B=0.38461, I=0.61539}*.

The second step is to smooth the distribution among named entity classes. The original distribution is: *{location=0.0, organisation=0.2, person=0.3, not-a-name=0.5}*. The smoothing function adds the parameter β to the probability of the not-a-name class and the parameter α to each named entity class probability. Then it normalises them. The resulting distribution is: *{location=0.125, organisation=0.225, person=0.275, not-a-name=0.375}*.

The final step is the combination of the two distributions by re-allocating the smoothed probabilities of each named-entity class to the smoothed tag distribution. The resulting distribution is: *{B-location=0.0625, I-location=0.0625, B-organisation=0.09375, I-organisation=0.13125, B-person=0.10577, I-person=0.16923, not-a-name=0.375}*.

Bibliography

Aberdeen, J., Burger, J., Day, D., Hirschman, L., Robinson, P. and Vilain, M.: 1995, MITRE: Description of the Alembic system as used for MUC-6, *in* MUC (1995), pp. 141–155.

Abney, S.: 2002, Bootstrapping, *in* ACL (2002), pp. 360–367.

Abney, S.: 2004, Understanding the Yarowsky algorithm, *Computational Linguistics* **30**(3).

Abney, S., Schapire, R. E. and Singer, Y.: 1999, Boosting applied to tagging and PP attachment, *in* EMNLP-VLC (1999), pp. 38–45.

ACL: 2002, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL2003)*, Philadelphia, PA, USA.

ANLP: 1997, *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, Washington, DC, USA.

Auer, P., Holte, R. C. and Maass, W.: 1995, Theory and applications of agnostic PAC-learning with small decision trees, *in* Prieditis and Russell (1995).

Baldrige, J. M. and Bierner, G.: 2001, OpenNLP MAXENT.
URL: *http://maxent.sourceforge.net*

Bender, O., Och, F. J. and Ney, H.: 2003, Maximum entropy models for named entity recogintion, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Bennett, S. W., Aone, C. and Lovell, C.: 1997, Learning to tag multilingual texts through observation, *in* C. Cardie and R. Weischedel (eds), *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP'97)*, Association for Computational Linguistics, Somerset, NJ, USA, pp. 109–116.

Berger, A. L., Della Pietra, S. A. and Della Pietra, V. J.: 1996, A maximum entropy approach to natural language processing, *Computational Linguistics* **22**(1).

Bikel, D., Miller, S., Schwartz, R. and Weischedel, R.: 1997, Nymble: a high-performance learning name-finder, *in* ANLP (1997), pp. 194–201.

Bikel, D., Schwartz, R. and Weischedel, R. M.: 1999, An algorithm that learns what's in a name, *Machine Learning* **34**(1–3).

Blum, A. L. and Langley, P.: 1997, Selection of relevant features and examples in machine learning, *Artificial Intelligence* **97**(1–2), 245–271.

Blum, A. L. and Mitchell, T.: 1997, Combining labeled and unlabeled data with Co-training, *in* ANLP (1997), pp. 92–100.

Borthwick, A.: 1999, *A Maximum Entropy Approach to Named Entity Recognition*, PhD thesis, Computer Science Department, New York University.

Borthwick, A., Sterling, J., Agichtein, E. and Grishman, R.: 1998, NYU: Description of the MENE named entity system as used in MUC-7, *in* MUC (1998).
URL: *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Brants, T.: 2000, TnT — a statistical part-of-speech tagger, *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, Seattle, WA, USA, pp. 224–231.

Brill, E.: 1995, Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging, *Computational Linguistics* **21**(4), 543–565.

Brushe, G. D., Mahony, R. E. and Moore, J. B.: 1996, A forward backward algorithm for ML state and sequence estimation, *Proceedings of the Fourth International Symposiumon Signal Processing and Its Applications (ISSPA '96)*, Gold Coast, Australia.

Brushe, G. D., Mahony, R. E. and Moore, J. B.: 1998, A soft output hybrid algorithm for ML/MAP sequence estimation, *IEEE Transactions on Information Theory* **44**(7), 3129–3134.

Budanitsky, A. and Hirst, G.: 2001, Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures, *in* NAACL (2001).

Burkhard, H. D.: 1998, Extending some concepts of CBR — foundations of case retrieval nets, *in* M. Lenz, B. Bartsch-Spörl, H. D. Burkhard and S. Wess (eds), *Case-Based Reasoning Technology: From Foundations to Applications*, Vol. 1400 of *Lecture Notes in Artificial Intelligence. Subseries of Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 17–50.

Cardie, C.: 1993, Using decision trees to improve case-based learning, *Proceedings of the Tenth International Conference on Machine Learning (ICML-1993)*, Morgan Kaufmann, Amherst, MA, USA.

Cardie, C.: 1997, Empirical methods in information extraction, *AI Magazine* **18**(4), 65–80. Special Issue on Empirical Natural Language Processing.

Cardie, C., Daelemans, W., Nédellec, C. and Tjong Kim Sang, E. (eds): 2000, *Proceedings of Fourth Conference on Computational Natural Language Learning (CoNLL-2000) and of the Second Learning Language in Logic Workshop (LLL-2000)*, Lisbon, Portugal.
URL: *http://cnts.uia.ac.be/conll2000*

Carreras, X., Màrquez, L. and Padró, L.: 2002, Named entity extraction using adaboost, *in* Roth and van den Bosch (2002).
URL: *http://cnts.uia.ac.be/conll2002*

Chieu, H. L. and Ng, H. T.: 2003, Named entity recogintion with a maximum entropy approach, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Chinchor, N.: 1998a, MUC-7 information extraction task definition, *in* MUC (1998). Version 3.5.
URL: *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Chinchor, N.: 1998b, MUC-7 test scores, *in* MUC (1998).
URL: *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Clark, S., Curran, J. R. and Osborne, M.: 2003, Bootstrapping POS taggers using unlabelled data, *in* Daelemans and Osborne (2003), pp. 49–55.
URL: *http://cnts.uia.ac.be/conll2003*

Coates-Stephens, S.: 1992, *The Analysis and Acquisition of Proper Names for Robust Text Understanding*, PhD thesis, Department of Computer Science, City University, London.

Cohen, W. W.: 1995, Fast effective rule induction, *in* Prieditis and Russell (1995).

Cohen, W. W.: 1996, Learning trees and rules with set-valued features, *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-1996)*, AAAI Press, Portland, OR, USA.

Collins, M. and Singer, Y.: 1999, Unsupervised models for named entity classification, *in* EMNLP-VLC (1999), pp. 189–196.

Cover, T. M. and Thomas, J. A.: 1991, *Elements of Information Theory*, Wiley, New York.

Cowie, J.: 1995, Description of the CRL/NMSU system used for MUC-6, *in* MUC (1995).

Cowie, J. and Lehnert, W.: 1996, Information extraction, *Special NLP Issue of the Communications of the ACM* **39**(1), 80–91.

CSL: 2004, Wordnet: a lexical database for the English language. Web page developed by the Cognitive Science Laboratory at Princeton University. Accessed last in February 2004.
URL: *http://www.cogsci.princeton.edu/~wn/*

Cucchiarelli, A., Luzi, D. and Velardi, P.: 1998, Automatic semantic tagging of unknown proper names, *Proceedings of the Seventeenth International Conference on Computational linguistics, COLING-98*, Association for Computational Linguistics, Morristown, NJ, USA, pp. 286–292.

Cucerzan, S. and Yarowsky, D.: 2002, Language independent NER using a unified model of internal and contextual evidence, *in* Roth and van den Bosch (2002).
URL: *http://cnts.uia.ac.be/conll2002*

Cui, J. and Guthrie, D.: 2004, Maximum entropy modeling in sparse semantic tagging, *in* S. Dumais, D. Marcu and S. Roukos (eds), *Proceedings of the HLT-NAACL 2004: Student Research Workshop*, Association for Computational Linguistics, Boston, MA, USA, pp. 13–18.

Culotta, A. and McCallum, A.: 2004, Confidence estimation for information extraction, *in* S. Dumais, D. Marcu and S. Roukos (eds), *Proceedings of the HLT-NAACL 2004: Short Papers*, Association for Computational Linguistics, Boston, MA, USA, pp. 109–112. (Short paper).

Cunningham, H.: 2000, *Software Architecture for Language Engineering*, PhD thesis, Computer Science Department, University of Sheffield.

Curran, J. R. and Clark, S.: 2003a, Investigating GIS and smoothing for maximum entropy taggers, *in* EACL (2003), pp. 91–98.

Curran, J. R. and Clark, S.: 2003b, Language independent NER using a maximum entropy tagger, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Daelemans, W. and Osborne, M. (eds): 2003, *Proceedings of Seventh Conference on Natural Language Learning CoNLL-2003*, Edmonton, Alberta, Canada.
URL: *http://cnts.uia.ac.be/conll2003*

Daelemans, W., van de Bosch, A. and Savrel, J.: 1999, Forgetting exceptions is harmful in language learning, *Machine Learning* **34**, 11–43.

Daelemans, W., Veenstra, J. and Buchholz, S.: 1999, Memory-based shallow parsing, *in* M. Osborne and E. Tjong Kim Sang (eds), *Proceedings of the Workshop on Computational Language Learning (CoNLL-99)*, Bergen, Norway.
URL: *http://ilk.uvt.nl/cgi-bin/tstchunk/demo.pl*

Daelemans, W. and Zajac, R. (eds): 2001, *Proceedings of the Fifth Workshop on Computational Language Learning (CoNLL-2001)*, Toulouse, France.
URL: *http://cnts.uia.ac.be/conll2001*

Daelemans, W., Zavrel, Z., van der Sloot, K. and van den Bosch, A.: 2003, Timbl: Tilburg memory based learner, version 5.0, reference guide, *Technical Report ILK-0310*, ILK Research Group, Tilburg University, Tilburg, The Netherlands.

Darroch, J. N. and Ratcliff, D.: 1972, Generalized iterative scaling for log-linear models, *The Annals of Mathematical Statistics* **43**(5), 1470–1480.

Dasgupta, S., Littman, M. L. and McAllester, D. A.: 2002, PAC generalization bounds for Co-training, *in* T. G. Dietterich, S. Becker and Z. Ghahramani (eds), *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, MIT Press, Cambridge, MA, USA, pp. 375–382.

de Boni, M.: 2004, *Relevance In Open Domain Question Answering: Theoretical Framework and Application*, PhD thesis, Department of Computer Science, University of York.

De Meulder, F. and Daelemans, W.: 2003, Memory-based named entity recognition using unannotated data, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Deerwester, S. C., Dumais, S. T., Landauer, Thomas K. andFurnas, G. W. and Harshman, R. A.: 1990, Indexing by latent semantic analysis, *Journal of the American Society of Information Science* **41**(6).

Della Pietra, S., Della Pietra, V. and Lafferty, J.: 1995, Inducing features of random fields, *Technical report*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Della Pietra, S., Della Pietra, V. and Lafferty, J.: 1997, Inducing features of random fields, *IEEE Transactions Pattern Analysis and Machine Intelligence* **19**(4).

Dobkin, D., Fulton, T., Gunopulos, D., Kasif, S. and Salzberg, S.: 2000, Induction of shallow decision trees. Submitted to the IEEE Transactions on Pattern Analysis and Machine Intelligence.

Douthat, A.: 1998, The message understanding conference scoring software user's manual, *in* MUC (1998).
URL: *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Durbin, R., Eddy, S., Krogh, A. and Mitchison, G.: 1998, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press.

EACL: 2003, *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, Budapest, Hungary.

EMNLP-VLC: 1999, *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP-VLC)*, College Park, MD, USA.

Escudero, G., Màrquez, L. and Rigau, G.: 2000, Boosting applied to word sense disambiguation, *Proceedings of the Twelfth European Conference on Machine Learning (ECML-2000)*, Barcelona, Spain, pp. 129–141.

Fellbaum, C. (ed.): 1998, *WordNet: An Electronic Lexical Database*, MIT Press.

Fisher, D., Soderland, S., McCarthy, J., Feng, F. and Lehnert, W.: 1995, Description of the UMass system as used for MUC-6, *in* MUC (1995).

Florian, R., Ittycheriah, A., Jing, H. and Zhang, T.: 2003, Named entity recogintion through classifier combination, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Forney, G. D.: 1972, The Viterbi algorithm, *Proceedings of the IEEE* **61**, 268–278.

Frank, E. and Witten, I. H.: 1998, Generating accurate rule sets without global optimization, *in* J. Shavlik (ed.), *Proceedings of the Fifteenth International Conference on Machine Learning (ICML'98)*, Morgan Kaufmann, pp. 144–151.

Freund, Y. and Schapire, R. E.: 1996, Experiments with a new boosting algorithm, *in* L. Saitta (ed.), *Proceedings of the Thirteenth International Conference on Machine Learning (ICML '96)*, Morgan Kaufmann, Bari, Italy, pp. 148–156.

Freund, Y. and Schapire, R. E.: 1999, A short introduction to boosting, *Journal of Japanese Society for Artifical Intelligence* **15**(5), 771–780. In Japanese, translation by Noaki Abe.

Fürnkranz, J. and Widmer, G.: 1994, Incremental reduced error pruning, *in* W. W. Cohen and H. Hirsh (eds), *Proceedings the Eleventh International Conference on Machine Learning (ICML-1994)*, Morgan Kaufmann, New Brunswick, NJ, USA, pp. 70–77.

Gaizauskas, R. and Humphreys, K.: 1997, Using a semantic network for information extraction, *Technical report*, Department of Computer Science, University of Sheffield, Sheffield, UK.

Gaizauskas, R., Wakao, T., Humphreys, K., Cunningham, H. and Wilks, Y.: 1995, University of Sheffield: Description of the LaSIE system as used for MUC-6, *in* MUC (1995).

Gale, W. A., Church, K. W. and Yarowsky, D.: 1992, A method for disambiguating word senses in a large corpus, *Computers and the Humanities* **26**, 415–439.

Godfrey, J., Holliman, E. and McDaniel, J.: 1999, SWITCHBOARD: Telephone speech corpus for research and development, *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '99)*, San Francisco, CA, USA, pp. 517–520.

Goldman, S. A. and Zhou, Y.: 2000, Enhancing supervised learning with unlabeled data, *in* P. Langley (ed.), *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, Morgan Kaufmann, Standord, CA, USA, pp. 327–334.

Grishman, R.: 1997, Information extraction: techniques and challenges, *in* Pazienza (1997), pp. 10–27.

Guarino, N.: 1997, Semantic matching: Formal ontological distinctions for information organization, extraction, and integration, *in* Pazienza (1997), pp. 139–170.

Hagenauer, J. and Hoeher, P.: 1989, A Viterbi algorithm with soft-decision outputs and its applications, *Proceedings of the IEEE GLOBECOM '89*, Dallas, TX, USA, pp. 1680–1686.

Hendrickx, I. and van den Bosch, A.: 2003, Memory-based one-step named-entity recognition: Effects of seed list features, classifier stacking, and unannotated data, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Hobbs, J. R.: 1993, The generic information extraction system, *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, Morgan Kaufmann, Baltimore, MD, USA, pp. 87–91.

Humphreys, K., Gaizauskas, R., Cunningham, H. and Azzam, S.: 1998, VIE technical specifications, *Technical report*, Department of Computer Science and Institute for Language, Speech and Hearing (ILASH), University of Sheffield, Sheffield, UK.

Jaynes, E. T.: 1991, Notes on present status and future prospects, *in* W. T. Grandy, Jr. and L. H. Shick (eds), *Maximum Entropy and Bayesian Methods*, Kluwer Academic, Dordrecht, pp. 1–13.

Junkawitsch, J., Neubauer, L., Höge, H. and Ruske, G.: 1996, A new keyword spotting algorithm with pre-calculated optimal thresholds, *Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP '96)*, Philadelphia, PA, USA, pp. 2067–2070.

Kilgarriff, A. and Rosenzweig, J.: 2000, English SENSEVAL: Report and results, *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece.

Klein, D., Smarr, J., Nguyen, H. and Manning, C. D.: 2003, Named entity recognition with character-level models, *in* Daelemans and Osborne (2003), pp. 180–183.
URL: *http://cnts.uia.ac.be/conll2003*

Krupka, G. R. and Hausman, K.: 1998, Description of the NetOwl^{TM} extractor system as used for MUC-7, *in* MUC (1998).
URL: *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Kubat, M.: 1998, Decision trees can initialize radial-basis-function networks, *IEEE Transactions on Neural Networks* **9**, 813–821.

Kubat, M., Flotzinger, D. and Pfurtscheller, G.: 1993, Discovering patterns in EEG-signals: Comparative study of a few methods, *in* P. Brazdil (ed.), *Proceedings of the Sixth European Conference on Machine Learning*, Vol. 667 of *Lecture Notes in Artificial Intelligence. Subseries of Lecture Notes in Computer Science*, Springer-Verlag, London, UK, pp. 366–371.

Kullback, S. and Leibler, R. A.: 1951, On information and sufficiency, *Annals of Mathematical Statistics* **22**, 76–86.

Lau, R.: 1994, *Adaptive statistical language modelling*, Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

LDC: 2005, ACE. Linguistic Data Consortium, University of Pennsylvania. Accessed last in December 2005.
URL: *http://projects.ldc.upenn.edu/ace/intro.html*

Lehnert, W., Cardie, C., Fisher, D., McCarthy, J., Riloff, E. and Soderland, S.: 1994, Evaluating an information extraction system, *Journal of Integrated Computer-Aided Engineering* **1**(6).

Lenz, M.: 1999, *Case Retrieval Nets as Model for Building Flexible Information Systems*, PhD thesis, Department of Computer Science, Humboldt University.

Lenz, M. and Burkhard, H. D.: 1996, Case retrieval nets: Foundations, properties, implementation, and results, *Technical report*, Department of Computer Science, Humboldt University, Lindenstr. 54a, 10117 Berlin, Germany.

Li, X., Malkin, J. and Bilmes, J.: 2004, Codebook design for ASR systems using custom arithmetic units, *Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004)*, Vol. 1, Montreal, Quebec, Canada, pp. 845–848.

Malouf, R.: 2002, Markov models for language-independent named entity recognition, *in* Roth and van den Bosch (2002).
URL: *http://cnts.uia.ac.be/conll2002*

Marcus, M. P., Santorini, B. and Marcinkiewicz, M. A.: 1993, Building a large annotated corpus of English: The Penn treebank, *Computational Linguistics* **19**, 313–330.

McCarthy, J. F.: 1996, *A Trainable Approach To Coreference Resolution For Information Extraction*, PhD thesis, Department of Computer Science, University of Massachusetts Amherst.

McDonald, D.: 1996, Internal and external evidence in the identification of proper names, The MIT Press, Cambridge, MA, USA, chapter 2, pp. 21–39.

Mikheev, A.: 1998, Feature lattices and maximum entropy models. Submitted to the Journal of Natural Language Engineering.
URL: *http://www.ltg.ed.ac.uk/~mikheev*

Mikheev, A., Grover, C. and Moens, M.: 1998, Description of the LTG system used for MUC-7, *in* MUC (1998).
URL: *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Mikheev, A., Grover, C. and Moens, M.: 1999, XML tools and architecture for named entity recognition, *Markup Languages* **1**(3), 89–113.

Mikheev, A., Moens, M. and Grover, C.: 1999, Named entity recognition without gazetteers, *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, Bergen, Norway, pp. 1–8.

Miller, G. A.: 1995, WordNet: A lexical database, *Communications of the ACM* **38**(11), 39–41.

Mitchell, T. M. (ed.): 1997, *Machine Learning*, McGraw-Hill.

Morguet, P. and Lang, M.: 1998, An integral stochastic approach to image sequence segmentation and classification, *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, Vol. 5, Seattle, WA, USA, pp. 2705–2708.

MUC: 1995, *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Morgan Kaufmann, San Francisco, CA, USA.

MUC: 1998, *Proceedings of the Seventh Message Understanding Conference (MUC-7).* **URL:** *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Müller, C., Rapp, S. and Strube, M.: 2002, Applying Co-training to reference resolution, *in* ACL (2002), pp. 352–359.

Muslea, I., Minton, S. and Knoblock, C.: 2002, Active + semi-supervised learning = robust multi-view learning, *in* Sammut and Hoffmann (2002), pp. 435–442.

NAACL: 2001, *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2001)*, Pittsburgh, PA, USA.

Needleman, S. B. and Wunsch, C. D.: 1970, A general method applicable to the search for similarities in amino acid sequences of two proteins, *Journal of Molecular Biology* **48**(3), 443–453.

Ng, V. and Cardie, C.: 2003, Bootstrapping coreference classifiers with multiple machine learning algorithms, *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP 2003)*, Association for Computational Linguistics, Sapporo, Japan, pp. 113–120.

Paliouras, G., Karkaletsis, V., Petasis, G. and Spyropoulos, C. D.: 2000, Learning decision trees for named-entity recognition and classification, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI 2000)*, Berlin, Germany.

Palmer, D. D. and Day, D. S.: 1997, A statistical profile of the named entity task, *in* ANLP (1997), pp. 190–193.

Park, S.-B. and Zhang, B.-T.: 2002, A boosted maximum entropy model for learning text chuncking, *in* Sammut and Hoffmann (2002).

Pazienza, M. T. (ed.): 1997, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, Vol. 1299 of *Lecture Notes in Artificial Intelligence. Subseries of Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany.

Penrose, O.: 1979, Foundations of statistical mechanics, *Reports on Progress in Physics* **42**, 1937–2006.

Pierce, D. and Cardie, C.: 2001, Limitations of Co-training for natural language learning from large datasets, *in* L. Lee and D. Harman (eds), *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP 2001)*, Pittsburgh, PA, USA, pp. 1–9.

Prieditis, A. and Russell, S. J. (eds): 1995, *Proceedings of the Twelfth International Conference on Machine Learning (ICML-1995)*, Morgan Kaufmann, Tahoe City, CA, USA.

Quinlan, J. R.: 1983, Learning efficient classification procedures and their application to chess end games, *in* R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds), *Machine Learning. An Artificial Intelligence Approach*, Tioga Press, Palo Alto, CA, USA, pp. 463–482.

Quinlan, J. R.: 1986, Induction of decision trees, *Machine Learning* **1**(1), 81–106.

Quinlan, J. R.: 1993, *C4.5: Programs For Machine Learning*, Morgan Kaufmann.

Rabiner, L. R.: 1989, A tutorial on hidden markov models and selected applications in speech recognition, *Proceedings of the IEEE* **77**(2), 257–286.

Ratnaparkhi, A.: 1996, A maximum entropy part-of-speech tagger, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'96)*, University of Pennsylvania.

Ratnaparkhi, A.: 1998, *Maximum Entropy Models for Natural Language Ambiguity Resolution*, PhD thesis, University of Pennsylvania.

Riloff, E. and Jones, R.: 1999, Learning dictionaries for information extraction by multi-level bootstrapping, *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-1999)*, AAAI Press, Orlando, FL, USA, pp. 474–479.

Ristad, E. S.: 1997, Maximum entropy modeling for discrete domains, Tutorial given at the Workshop on Mathematical Techniques to Mine Massive Data Set.

Ristad, E. S.: 1998, Maximum entropy modeling toolkit. Version 1.6 Beta. Now a product of Mnemonic Technology Inc.

Rosenfeld, R.: 1996, A maximum entropy approach to adaptive statistical language modeling, *Computer Speech and Language* **10**(3), 197–228.

Rotaru, M. and Litman, D. J.: 2003, Exceptionality and natural language learning, *in* Daelemans and Osborne (2003), pp. 63–70.
URL: *http://cnts.uia.ac.be/conll2003*

Roth, D. and van den Bosch, A. (eds): 2002, *Proceedings of Sixth Workshop on Computational Language Learning CoNLL-2002*, Taipei, Taiwan.
URL: *http://cnts.uia.ac.be/conll2002*

Salzberg, S. L.: 1990, *Learning with Nested Generalized Exemplars*, Kluwer Academic, Norwell, MA, USA.

Sammut, C. and Hoffmann, A. G. (eds): 2002, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML-2002)*, Morgan Kaufmann, Sydney, Australia.

Sarkar, A.: 2001, Applying Co-training methods to statistical parsing, *in* NAACL (2001), pp. 95–102.

Schapire, R. E.: 1990, The strength of weak learnability, *Machine Learning* **5**(2), 197–227.

Schlapbach, A. and Bunke, H.: 2004, Using HMM based recognizers for writer identification and verification, *Proceedings of the Nineth International Workshop on Frontiers in Handwritting Recognition (IWFHR-9 2004)*, Tokyo, Japan, pp. 167–172.

Sekine, S.: 1998, NYU: Description of the japanese NE system used for MET-2, *in* MUC (1998).
URL: *http://www.itl.nist.gov/iad/894.02/related_projects/muc*

Sellers, P. H.: 1974, Theory and computation of evolutionary distances, *SIAM Journal of Applied Math* **26**, 787–793.

Sheffield NLP Group: 2005, The ACE system. The Natural Language Processing Research Group, Department of Computer Science, University of Sheffield. Accessed last in December 2005.
URL: *http://www.dcs.shef.ac.uk/nlp/muse/ace.html*

Soderland, S., Fisher, D. and Lehnert, W.: 1997, Automatically learned vs. handcrafted text analysis rules, *Technical Report TE-44*, National Centre for Intelligent Information Retrieval, University of Massachusetts, Amherst MA.

Steedman, M., Sarkar, A., Osborne, M., Hwa, R., Clark, S., Hockenmaier, J., Ruhlen, P., Baker, S. and Crim, J.: 2003, Bootstrapping statistical parsers from small datasets, *in* EACL (2003), pp. 331–338.

Stevenson, M.: 1998, Extracting syntactic relations using heuristics, *Proceedings of the Tenth European Summer School in Logic, Language and Information (ESSLLI '98)*, Saarbrucken, Germany.

Sundheim, B. M.: 1995, MUC6 named entity task definition, version 2.1, *in* MUC (1995).

Tjong Kim Sang, E. F.: 2002a, Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition, *in* Roth and van den Bosch (2002).
URL: *http://cnts.uia.ac.be/conll2002*

Tjong Kim Sang, E. F.: 2002b, Memory-based named entity recognition, *in* Roth and van den Bosch (2002).
URL: *http://cnts.uia.ac.be/conll2002*

Tjong Kim Sang, E. F. and De Meulder, F.: 2003, Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

van Rijsbergen, C. J.: 1979, *Information Retrieval.* Online version available.
URL: *http://www.dcs.gla.ac.uk/Keith/Preface.html*

Viterbi, A. J.: 1967, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Transactions on Information Theory* **IT-13**, 260–269.

Vossen, P. (ed.): 1998, *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*, Kluwer Academic.

Wacholder, N., Ravin, Y. and Choi, M.: 1997, Disambiguation of proper names in text, *in* ANLP (1997), pp. 202–208.

Wakao, T., Gaizauskas, R. and Wilks, Y.: 1996, Evaluation of an algorithm for the recognition and classification of proper names, *Proceedings of the Sixteenth International Conference on Computational Linguistics (COLING-96)*, Morgan Kaufmann, Copenhagen, Denmark, pp. 418–423.

Whitelaw, C. and Patrick, J.: 2003, Evaluating corpora for named entity recognition using character-level features, *Proceedings of Sixteenth Australian Joint Conference on Artificial Intelligence AI'03*, Perth, Western Australia, pp. 910–921.

Wilks, Y.: 1997, Information extraction as a core language technology, *in* Pazienza (1997), pp. 1–9.

Witten, I. H. and Frank, E.: 2000, *Data Mining: Practical Machine Learning Tools with Java Implementations*, Morgan Kaufmann, San Francisco, CA, USA.

Wolpert, D. H.: 1992, On overfitting avoidance as bias, *Technical Report SFI TR 92-03-5001*, The Santa Fe Institute.

WRG: 2004, White Rose Grid At York. Accessed last in September 2004.
URL: *http://www.wrg.york.ac.uk/WhiteRoseGridYork/*

Wu, D., Ngai, G. and Carpuat, M.: 2003, A stacked, voted, stacked model for named entity recognition, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Wu, D., Ngai, G., Carpuat, M., Larsen, J. and Yang, Y.: 2002, Boosting for named entity recognition, *in* Roth and van den Bosch (2002).
URL: *http://cnts.uia.ac.be/conll2002*

Yarowsky, D.: 1995, Unsupervised word sense disambiguation rivaling supervised methods, *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics ACL'95*, Cambridge, MA, USA, pp. 189–196.

Zhang, J.: 1992, Selecting typical instances in instance-based learning, *in* D. H. Sleeman and P. Edwards (eds), *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992)*, Morgan Kaufmann, Aberdeen, Scotland, pp. 470–479.

Zhang, J., Shen, D., Zhou, Guodong andSu, J. and Tan, C.-L.: 2004, Enhancing HMM-based biomdical named entity recognition by studying special phenomena, *Journal of Biomedical Information* **To Appear**. Special Issue on Natural Language Processing in Biomedicine: Aims, Achievements and Challenges.

Zhang, T., Damerau, F. and Johnson, D. E.: 2002, Text chunking based on a generalization of Winnow, *Journal of Machine Learning Research* **2**, 615–637.

Zhang, T. and Johnson, D.: 2003, A robust risk minimization based named entity recogintion system, *in* Daelemans and Osborne (2003).
URL: *http://cnts.uia.ac.be/conll2003*

Zhou, G. and Su, J.: 2002, Named entity recogintion using an HMM-based chunk tagger, *in* ACL (2002).