Optimisation of Correlation Matrix Memory Prognostic and Diagnostic Systems

Alexander Grindell Fargus
EngD
University of York
Computer Science

February 2015

Abstract

Condition monitoring systems for prognostics and diagnostics can enable large and complex systems to be operated more safely, at a lower cost and have a longer lifetime than is possible without them. AURA Alert is a condition monitoring system that uses a fast approximate k–Nearest Neighbour (kNN) search of a timeseries database containing known system states to identify anomalous system behaviour. This search algorithm, AURA kNN, uses a type of binary associative neural network called a Correlation Matrix Memory (CMM) to facilitate the search of the historical database. AURA kNN is evaluated with respect to the state of the art Locality Sensitive Hashing (LSH) approximate kNN algorithm and shown to be orders of magnitude slower to search large historical databases. As a result, it is determined that the standard AURA kNN scales poorly for large historical databases.

A novel method for generating CMM input tokens called Weighted Overlap Code Construction is presented and combined with Baum Coded output tokens to reduce the query time of the CMM. These modifications are shown to improve the ability of AURA kNN to scale with large databases, but this comes at the cost of accuracy. In the best case an AURA kNN search is 3.1 times faster than LSH with an accuracy penalty of 4% on databases with 1000 features and fewer than 100,000 samples. However the modified AURA kNN is still slower than LSH with databases with fewer features or more samples.

These results suggest that it may be possible for AURA kNN to be improved so that it is competitive with the state of the art LSH algorithm.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisors Jim Austin and John McAvoy for their assistance and guidance throughout my research. In addition I thank Cybula Ltd. for sponsoring me to perform the research.

Finally, I thank my wife, Sandy, for her encouragement and patience over the last 5 years.

Declarations

I declare that the research presented in this thesis is original work undertaken at the Universtiy of York 2010 – 2015 and sponsored by Cybula Ltd. Unless otherwise stated, the contents of this thesis are an original contribution of the author.

Chapter 1

Introduction

1.1 Problem Summary

A system can be considered as a set of "things" that are working together in order to achieve some objective (Oxford Dictionaries 2010). These "things" could form an organic system such as a plant, an inorganic system such as an engine, or a hybrid system consisting of both. In most cases, the individual components can be considered as systems in their own right. For example, a car can be considered as a system that consists of separate sub–systems for propulsion, steering and breaking.

Unfortunately systems can fail. Indeed the consequences of such a failure can be catastrophic (Commission 1986; Travis 1994). It is therefore desirable to be able to predict when such a failure will occur so that the failure can be avoided or the effects mitigated. Prognostics is the process by which the future operation of a system can be predicted based on the current operating characteristics and with reference to some historical notion of normal operating behaviour (Vichare and Pecht 2006). In contrast, diagnostics is the process by which the cause of a failure is identified after it has occurred (Jardine, Lin, and Banjevic 2006).

This thesis presents an evaluation of, and improvements to, the approximate k–Nearest Neighbour algorithm that underlies AURA Alert. AURA Alert is a commercial software system that is used for both prognosis and diagnosis of complex, large–scale industrial systems (Austin et al. 2010). Typically these are systems with over 1000 monitored channels, with sample rates that can range from milliseconds to hours depending on the channel and consisting of at least 3 years of data.

1.2 Motivation

Condition monitoring systems that are used to provide prognostics and diagnostics have a cost associated with their installation and operation. The motivation for deploying such condition monitoring systems therefore stems from a desire to either increase the safety (Commission 1986) or reduce the cost of operating the target system (Pecht 2008).

Costs can be reduced in two ways, by either using prognostics to avoid a catastrophic failure in real time, for example, hard drives routinely have shock protection built in. This functions by detecting a drop as it occurs and seeks to ensure that the disk heads are

parked before impact. As a result, the disk heads do not cause the disks to shatter during impact (Edgerton and Kochis 1998).

Alternatively, the condition monitoring system can be used to justify extending the operating lifetime of a target system (Chinnam and Baruah 2004) because failures can be anticipated and compensated for. This allows the cost of replacing a legacy system to be avoided.

There are three high level strategies for implementing a condition monitoring system. These are: canary systems that are designed to fail before the target system thus providing a warning of iminent failure; physics–based models that use computational modelling of the actual operation of the system to predict problems; and data driven models that learn to model the outputs of a system by analysing recorded data. These strategies are considered further in Chapter 2. However the need to monitor increasingly large and complex systems combined with recent improvements in data collection and storage technologies (Jardine, Lin, and Banjevic 2006) have led to a massive increase in the amount of data that is potentially available about a system. The data driven approaches need to be able to scale with these increasing data volumes to remain relevant.

AURA Alert is one such data driven condition monitoring system. It is able to register an alert notification when the current system characteristics deviate sufficiently from a historical record of safe system states. AURA Alert is discussed in greater detail in Chapter 4. However the core component of the AURA Alert system is an approximate k–Nearest Neighbours (kNN) (Fukunaga 1990) search algorithm that uses a form of binary associative neural network called a Correlation Matrix Memory (CMM) (Austin 1996), specifically using the Advanced Uncertain Reasoning Architecture (AURA) (Austin, Kennedy, and Lees 1995) CMM framework, to facilitate the fast searching of historical system states.

Despite the importance of the AURA kNN algorithm to the operation of AURA Alert, there has been relatively little study of AURA kNN with respect to alternative kNN algorithms. To date, only Hodge and Austin (2005) provide any comparative evaluation of AURA kNN. This comparison is limited to demonstrating that AURA kNN is faster than the naïve Linear scan (Zezula et al. 2006) algorithm with a small loss in accuracy.

Additionally, the AURA kNN algorithm appears to scale linearly with the number of samples that are being searched. This will make AURA kNN increasingly unsuitable as the amount of data needed to be processed increases. However Hobson (2011) provides suggestions for increasing the information density of binary CMMs through the selection of the inputs and outputs that are associated within the CMM. If applied to AURA kNN, this could potentially improve the ability of AURA kNN to scale with larger datasets.

1.3 Research Questions

My thesis is that AURA Alert can be improved through the use of new methods to optimise the AURA kNN component. This leads to the following research questions:

1. How does AURA kNN compare with the existing state of the art kNN algorithms?
2. How can AURA kNN be modified such that it is competitive with those state of the art algorithms?

The research presented in this thesis seeks to provide answers to these two questions by evaluating both the standard AURA kNN and various modifications to AURA kNN with respect to state of the art kNN algorithms in terms of the query time, training time and accuracy of the algorithms.

1.4 Summary of Contributions

As a result of the investigation into the above questions, the following novel contributions are presented in this thesis:

1. An optimisation of the Overlapped Binary Code Construction (OBCC) (Hobson 2011) method for generating binary CMM input tokens (Section 5.2).

2. A method for generating weighted CMM input tokens, Weighted Overlap Code Construction (WOCC), that allows a reduced number of bits to be stored in a binary CMM in comparison to OBCC tokens while recalling identical results (Section 5.3).

3. A modified AURA kNN algorithm that uses WOCC input tokens and 2–Bit Baum Coded (Baum, Moody, and Wilczek 1988) output tokens (Chapter 9.3).

In addition, the results of the comparison experiments in Chapter 7 demonstrate that the standard AURA kNN algorithm is generally slower that the exact KD–Tree (Bentley 1975) and Dual KD–Tree (Gray and Moore 2000) spatial partitioning based kNN algorithms.

1.5 Structure of this Thesis

Chapter 2 contains a high level overview of the approaches to prognostics and diagnostics of systems. Canary systems, mathematical models and data driven models are introduced with data driven models identified as the most promising approach for large and complex systems.

Chapter 3 reviews the steps involved with creating a data driven model for system monitoring before discussing methods for building such models. Search based classification is one such method that uses similarity between the current system state and a historical record of previous system states to determine the health of a system. A review of k–Nearest Neighbour (kNN) algorithms for performing this is then presented.

Chapter 4 introduces AURA Alert, a system for performing search based classification using a type of binary associative neural network called a Correlation Matrix Memory (CMM). This search is called AURA kNN.

Chapter 5 presents three novel modifications to AURA kNN that have the potential to improve the speed and accuracy of the search. Overlapped Binary Code Construction (OBCC) (Hobson 2011) is a method for generating binary inputs for a CMM. Optimisations to this method are introduced to make it feasible for use as part of AURA kNN. Next, a new method of generating inputs for AURA CMMs, Weighted Overlap Code Construction (WOCC), is introduced based on OBCC. Finally the use of Baum Codes (Baum, Moody, and Wilczek 1988) to reduce the query time of the CMM is examined.

Chapter 6 introduces the kNN experiments that forms the basis of the evaluation performed in later chapters. In addition, baseline results are presented for the Linear Scan, KD–Tree, Dual KD–Tree and Locality Sensitive Hashing (LSH) kNN algorithms.

Chapter 7 details the investigation into how OBCC and WOCC input tokens affect the accuracy and query time of AURA kNN when using standard unary output tokens. It is shown that the standard Parabolic Kernel tokens perform best in this situation.

Chapter 8 investigates how the results from Chapter 7 change when 2–bit Baum Coded output tokens are used in place of the standard unary output tokens. It is shown that overall WOCC input tokens perform best when paired with Baum Coded output tokens.

Chapter 9 compares AURA kNN with WOCC input tokens and 2–bit Baum Coded output tokens to both the standard AURA kNN and LSH as the state of the art approximate kNN algorithm. The modified AURA kNN is shown be 67.5 times faster than standard AURA kNN for the largest datasets examined, however with a 4.6% loss in accuracy for these datasets. Compared to LSH, the modified AURA kNN is 3.1 times faster with a 4.0% loss in accuracy on these same datasets. However the accuracy of the modified AURA kNN is shown to be very poor for datasets with relatively few features.

Finally Chapter 10 provides a review of the findings from this research and presents the conclusions along with potential further work.

Chapter 2

Prognostic and Diagnostic Systems

2.1 Introduction

This Chapter introduces three high level strategies that can be considered to approach the task of creating both prognostic and diagnostic systems for any target system. These strategies are: canary systems, physics–based models and data–driven models. Section 2.2 provides a brief description of each approach and discusses some of their advantages and disadvantages.

Most real world applications of both prognostic or diagnostic systems will consist of a combination of these three approaches. The reasons for this are discussed in Section 2.3.

Finally, Section 2.4 provides a brief outline of some issues that effect practically all attempts to build prognostic or diagnostic systems for large or complex systems.

2.2 High Level Strategies

2.2.1 Canary Systems

Canary systems are simple systems that are designed to operate within the same environment as the target system. They are subjected to the same conditions as the target system with the intention that a canary system will fail first. The failure of a canary system indicates that the target system may also be about to fail (Pecht 2008).

The term originates from the use of canaries in mines during the 19th Century. Miners took canaries down the shafts to detect the presence of carbon monoxide (Schalie et al. 1999). If the canary fell ill then the miners had time to evacuate the area before they started to suffer from carbon monoxide poisoning themselves.

A fuse is a typical example of a canary system. When the fuse carries more electrical current than the system can handle, the fuse wire melts. This prevents the excessive current from damaging the rest of the system (Edison 1890).

There are a number of problems with canary systems. In order to provide multiple warnings, or to provide a warning about a range of potential failures, it is necessary to use multiple differently configured canaries (Pecht 2008). These are typically installed within the target system because of the need for the canary to operate within the same environment.

A canary system may therefore be difficult to develop for certain types of system,

especially those where space, weight or access to components is limited. In addition, depending on its application, the deployment of a canary system may require that the target legacy systems be recertified (Pecht 2008).

Increasingly, large and complex systems are created by joining together pre-existing legacy systems (Hopkins and Jenkins 2008). This can often make the development of a suitable canary system for the merged target system very expensive.

Finally after the failure of a canary system, there remains the issue of whether the protected system is affected by the introduction of a replacement canary.

2.2.2 Physics–Based and Mathematical Models

Physics–based models are software systems that are specified by experts in the target system so that they can model as closely as possible the physical effects of a particular input to the system. This approach is only appropriate for systems such as an engine or a bridge that have interactions with the physical world.

Physical models, combined with a record of the loads that a system is subjected to can be used to determine the remaining life of the target system (Pecht 2008). The load to which the target system is subjected can be monitored by both a record of the inputs to the system and sensors placed on or within the system.

When the model is an accurate representation of the system, this approach has been shown to be effective. For example, a mathematical model based on the physics of a gearbox has been shown to model the effects of a tooth crack in a way that corresponds to experimental observations (Howard, Jia, and Wang 2001).

However, it may be difficult or even impossible to build sufficiently accurate models for very large or complex systems (Jardine, Lin, and Banjevic 2006); either because a large and complex IT system may be needed simply to perform the computation required for the prognostics (Pope et al. 2007); or because there is insufficient understanding about how all components of the system operate and interact to build an accurate enough model (Hopkins and Jenkins 2008).

2.2.3 Data Driven Models

The data–driven approach is to build software models of the system that are based solely on data that is recorded about the target system. There is no attempt to model the actual physical workings of the system. This approach to prognostics and diagnostics requires the use of sensors to collect information about the target system and the storage of that information (Jardine, Lin, and Banjevic 2006).

Examination of the historical information recorded by these sensors gives the ability to learn from past events (Pecht 2008). This can allow the identification of failure precursors so that measures can be taken to avoid the failure when such precursors reoccur. This examination of data can take the form of either monitoring continuously as with an online system or monitoring in batch mode during periodic maintenance windows.

Traditionally this examination was performed via time consuming and expensive manual processes (Wehenkel 1998). However with the proliferation of both sensors to create data and networks to transfer it, the amount of data available means that this approach is no longer sustainable. Therefore ways of automatically learning from this data are sought.

Automatic learning can be achieved either through the use of traditional statistics (Jardine, Lin, and Banjevic 2006; Pecht 2008), with machine learning techniques (Jardine, Lin, and Banjevic 2006; Pecht 2008; Bishop 1995) or a combination of both.

However, all approaches suffer from similar issues such as: noise in the data that impedes learning or analysis of the underlying system; identifying the relevant or important information, often from a vast amount of collected data; and a lack of important information, in particular, concerning the operation of the system in failure modes.

There are two ways in which to apply the data–driven approaches to developing prognostics and diagnostics systems.

The first approach is classification. This is used when there is sufficient historical training data available to cover all the operational states of the system, including the failure states. The classification system then examines the current operational state with respect to the training data in order to determine what label to apply to the current state. This approach is useful for identifying known failure conditions.

The second approach is anomaly detection. This is used to identify when the system enters an operational state that has not been seen in the historical training data. This approach is useful for identifying new and unknown operating conditions that may act as a precursor to a new type of failure.

2.3 Practical Application of Current Approaches

In Section 2.2 three general approaches to developing prognostic and diagnostic systems were presented. However not every approach is suitable for every task. In addition, for large and complex systems it may not be possible to apply a single approach to the entire system. Therefore different subsystems may be monitored using different approaches. In practice, prognostic and diagnostic systems for complex systems usually comprise of a combination of all three approaches (Schwabacher 2005; Atlas et al. 2001).

Provided that the system is well understood, it is typically the case that mathematics and physics–based models are able to make very accurate predictions about the system (Howard, Jia, and Wang 2001). For example, the European Centre for Medium-Range Weather Forecasts (ECMWF) is able to correctly predict extreme weather events 5 days in advance (European Centre for Medium Range Weather Forcasting 2013). However, it can often be very computationally expensive to do so. This results in either extremely high equipment costs or a prohibitively long time to make a prediction. The ECMWF spends around £14 million a year on computer equipment alone (European Centre for Medium Range Weather Forcasting 2013) to enable fast and accurate predictions.

As a result, the choice about whether to use a physical model or a data–driven model can often be driven by the real time requirements for a prediction. If a fault in a system can very quickly lead to a significant failure, then a potentially less accurate and faster approach may be more suitable. Similarly if plenty of time is available to run simulations, then greater accuracy in the predictions will be preferred.

The data–driven approach can be used when the underlying system is not sufficiently understood to make accurate physics–based models, or when the time and expertise needed to develop a physical model is not available. This is because the data–driven algorithms can

learn models that exhibit the same behaviour as the system without needing to replicate the interactions that cause the behaviour in the original system. These model can sometimes be generalised to apply to other instances of the same system. However many complex systems are unique, and as result, generalised data models will have to be adapted to the specifics of each system.

2.4 Limitations of Current Approaches

2.4.1 Method Validation

Even the most sophisticated prognostic systems need to be both validated and verified to ensure that they are functioning correctly and to develop operator trust in the predictions given (Pecht 2008). When considering models of real world systems, validation is the process by which it is determined how accurately the model is able to represent its target system (Balci 1997). Verification is the process by which a system is determined to be operating with respect to its specifications (Balci 1997).

If a prognostic system is unable to predict system failures then there is little reason to incur the cost of operating the prognostic system. Similarly if the prognostic system provides a large number of warnings that do not correspond to actual problems then the operator will lose confidence in the provided warnings and may ignore actual problems when they are predicted.

Therefore it is necessary to both validate and verify the prognostic system in order to build operator trust. This is because the cost of responding to a warning may be very high, this cost is only offset by the risk that failure to respond could be even more costly. So to ensure that operators react correctly, they must trust the system to generate accurate warnings.

However this can be very difficult. Verification of software systems can be achieved through a combination of testing (Myers, Sandler, and Badgett 2011) and formal methods such as model checking (Bérard et al. 2010) to prove that the system is operating as specified. However this is beyond the scope of this thesis and how these techniques can be applied successfully to ever larger and more complex systems is an on going topic of much research.

Validation requires that the output of the prognostic system be compared to the real output of its target system. However, consider a system that does not fail during a particular period of time. Assuming that a prognostics is deployed to monitor it during that period and it does not generate any warnings, there is no way to be certain that the prognostic system was operating correctly during that time since a defective prognostic system could also legitimately generate no warnings.

Validating the prognostic system requires that it be tested on systems that both operate correctly and that fail. However, many very expensive or safety critical systems cannot be allowed to fail due to cost or safety reasons. In these situations it can be very difficult to test that the prognostic system would actually work as expected in the event of a real failure and as a result it could be difficult to build confidence in the prognostic system (Pecht 2008).

2.4.2 Liability

Prognostic systems are typically deployed to systems where the consequence of failure is very high. These target systems are often developed to very high standards in order to prevent their failure (O'Halloran and Pygott 2007).

However, the prognostic systems themselves may not be developed to such high standards. In the event of an uncaught failure, or false prediction, who is responsible for the failure? It could be the operator of the prognostic system, the company that developed the system, an individual programmer or many other entities (Pecht 2008). The method by which this issue is resolved is likely to have a significant effect on the development and deployment of prognostic systems (Pecht 2008).

2.5 Summary

In this Chapter three approaches for creating prognostic and diagnostic systems have been described.

Canary systems operate within the same environment as the target system. They are designed to fail quicker than the target system in order to provide a warning about immanent failure or protect the target system from damage during that failure. The application of canary systems to large and complex systems can be problematic, especially where the target system may need to be recertified after the installation of a replacement canary.

Physical or Mathematical models provide a software representation of the target system based on an understanding of how the system works in the real world. When the target system is well enough understood and there is sufficient time, computation and input data, available this approach can provide very accurate predictions about the future of the target system. However in the case of large and complex systems it is rarely possible to satisfy these three requirements for the entire system.

Data driven models produce predictions based on the application of statistical analysis or machine learning techniques to data that is recorded about the target system. This approach can be applied even when the target system is not fully understood because the approach involves trying to automatically learn about the operation of the system. As a result of this however, the way by which some data–driven models work to produce their predictions can be difficult to understand. This therefore makes it harder to build the necessary trust in a data–driven model.

The limitations of both canary systems and physics–based mathematical models mean that these approaches do not scale well when trying to cover the entirety of a large or complex system. Despite this they can be useful for monitoring smaller subsystems within a system of systems. However the rest of this thesis shall focus on the development of data–driven models. This is because the data driven approaches can require fewer resources than the physics–based approaches and they are not required to interact with the target systems as canaries are. Therefore the data–driven approach appears to be best suited to the task of trying to monitor ever larger and more complex systems.

Chapter 3

Data–Driven Systems Monitoring

3.1 Introduction

There are several common steps that are required by practically all applications of data–driven model development. An overview of these common steps is given below:

In order to build a data–driven model, the first step is to acquire the data that drives the model. This data is referred to as the training data and it is often a non-trivial task to obtain sufficient high quality training data to construct useful models. Section 3.2 will cover some of the issues that relate to collection of this training data.

With sufficient training data in hand, the next step is to clean and filter the data. This usually consists of stripping out any invalid or misleading data. Alternatively, it may be necessary to construct additional data to fill gaps that were infeasible or perhaps impossible to gather during the data acquisition step. These issues will be discussed further in Section 3.3.

Once the training data is in a suitable state, it is necessary to extract the features from the data that are to be used in the models. Many models require that the data be transformed in order to better interpret important information from within the data. Several of the most commonly applied techniques are discussed in Section 3.4.

Typically only a small proportion of the extracted features have a disproportionate effect on the measured output of a system. Since many models are only able to handle a limited number of features, it is important to select only the most discriminating features to be used in the models. This is discussed in further detail in Section 3.5.

A data–driven model can be applied to two purposes; either to assign the currently observed system state into predefined categories, usually derived from the previously observed system states, this is termed classification; or to identify new system states that do not fall into any of the previously observed categories, this is termed Anomaly Detection. The general concepts relating to both classification and anomaly detection are presented in Section 3.6.

In Sections 3.7 – 3.11 a number of techniques for developing both classification and anomaly detection models for systems are discussed.

Finally a short discussion on some of the techniques used to evaluate the models is in Section 3.12.

The issues related to deploying a model will not be covered. This is because such

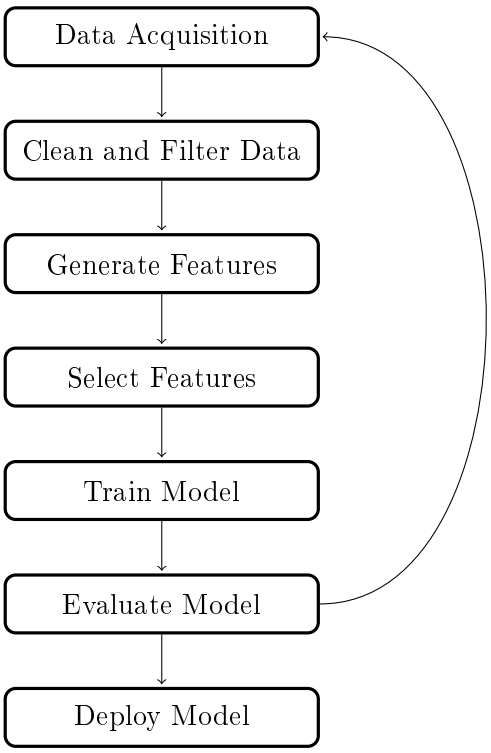


Figure 3.1: The development pipeline for the creation of data–driven prognostics and diagnostic systems

issues are often closely related to social and business problems rather than purely technical problems. A discussion of these topics is beyond the scope of this thesis.

Rather than being a linear progression from one step to the next, building a data–driven model is typically an iterative process where each stage may be returned to many times before the model is deemed suitable for deployment. An outline of these steps can be seen in Figure 3.1.

3.2 Training Data Acquisition

Acquisition of training data is an essential prerequisite for building any data–driven model (Pecht 2008). Jardine, Lin, and Banjevic (2006) define two types of data that can be recorded when monitoring a system - event data and condition data.

Event data is defined as a log of the major events that occur to the system, for example, installation and scheduled or unscheduled maintenance. Usually these events are characterised by their occurrence at irregular intervals and human interaction with the system.

Condition data is defined as data that is collected automatically at regular, fixed intervals and records the operating characteristics of the system. In a physical system, sensors are used to measure physical characteristics such as voltage, temperature, vibrations, etc. In software systems, this could be a record of bandwidth, processor time, etc. It is this data that is primarily used to build data–driven models (Jardine, Lin, and Banjevic 2006).

In recent years, the miniaturisation of electronics combined with advances in wireless networks, has meant that it is easier to deploy sensors for the purpose of monitoring a system. For example, Austin et al. (2013) have developed a device to record EEG

and accelerometer movements at 2kHz for over 24 hours while weighing only 2.3 grams. However this has resulted in a proliferation in the amount of condition data being collected and stored (Pecht 2008).

Yet the condition data itself is usually not sufficient to build a model. The training data will typically also consist of a set of labels that are assigned to the condition data samples. Each label corresponds to the operating characteristics of the system at the time the sample was recorded. Some systems can learn from training data in an unsupervised manner and thereby avoiding the need for labelled training data (Wilson, Keil, and Fahle 1999). However, at a minimum it is necessary to know whether or not the condition data corresponds to a system that is operating normally.

Generally the accuracy with which a data–driven model can represent a target system will largely depend on the volume and quality of the information contained within the training data (Caballero et al. 2006)

The volume of training data required to accurately build a data–driven model is largely dependent on the size and complexity of the target system. In the case of a simple binary classification of a non–parametric system, the size of a training dataset will need to more that double in order to double the accuracy of the classifications (Raudys and Jain 1991). Therefore a large amount of training data is needed to model a large and complex system with a high level of accuracy.

In many cases the labelling of training data has to be performed by a human expert. This introduces a significant cost to gathering training data. In addition it introduces the possibility of samples being mislabelled. Disagreement amongst experts about how to label the samples is also likely to arise in cases where the volume of data necessitates multiple human experts (Stephenson et al. 2009).

Despite the increase in monitoring of systems it can still be difficult to get sufficient training data to cover all the operating conditions of the system. When considering high integrity systems, for example, there may not be any condition data that relates to fault conditions of a system. This is because the occurrence of such conditions are, by design, exceedingly rare (Markou and Singh 2003a). Few system operators of high integrity systems would be willing to damage their systems simply to record the effects.

Finally there is the issue that many companies consider the condition data recorded from their system to be business sensitive information. For that reason they are often unwilling to share the information needed to develop models of their systems.

3.3 Cleaning and Filtering

The purpose of cleaning and filtering the data is to reduce the amount of irrelevant information in the training data. If such information is retained in the training data it can prevent the algorithms used to build data–driven models from converging on an acceptable solution. In some cases, this could be overcome with additional training data (Fukunaga 1990). However, as discussed in Section 3.2, it is not always feasible to obtain more data.

3.3.1 Cleaning

Missing or invalid samples are encountered regularly in time series data (Little 1992). There are a large number of causes for this, but often it is due to a failure in the monitoring system. The process of cleaning the data is to identify and remove these invalid portions of the signal. In general, cleaning data is not easily achieved (Maimon and Rokach 2010). This is because there is often no way of knowing whether a period of unusual or missing data is indicating something important about the health of the system or whether it is being caused by transient environmental issues.

One solution to this problem is to perform manual identification of useful data with the help of graphical tools (Maimon and Rokach 2010). This solution however is both time consuming, and therefore expensive, and also potentially error prone. Additionally, when large quantities of data are being considered, time constraints mean that it may not be feasible to manually review all the data.

Another solution is to assume that the system functioned correctly for the majority of the operating period during which the training data was gathered. Various methods can then be used to identify outliers or anomalous data within this dataset.

Statistical measures such as standard deviation and the mean, or even a simple threshold, could be used to identify normality and any periods that fall outside this would be considered abnormal (Maletic and Marcus 2010). However, this requires some understanding of the data and the underlying system. It may, for example, be expected that the system will occasionally produce extreme values. If this is the case then such values will need to be retained as useful data.

Clustering of the data can also be used to identify outliers (Yu, Sheikholeslami, and Zhang 2002). Any data that does not fall within an identified cluster can be considered to be anomalous, in addition clusters that appear to contain less data than expected can be used to identify where data is missing (Maletic and Marcus 2010).

Other approaches can make use of model checking (Mezzanzanica et al. 2013) to ensure that the collected data conforms to a set of known states and transitions within the data. For example if a system is recorded being in two different states, but with no record of the transition between them, then it can be identified that the data representing the transition is missing.

Once useful data has been identified, any data that is not complete or that contains artefacts could simply be discarded (Little 1992). The advantage of this approach is that it ensures that the training data only consists of high quality data. The disadvantage however is that there may not be sufficient remaining data to be able to build an accurate model of the system (Fukunaga 1990), especially if training data was limited to begin with.

Rather than discard the corrupted training data, it may be possible to partially restore it. For example, an extreme sample that is correctly identified as an artefact could be replaced with the mean of its adjacent samples (Little 1992). Whilst this can retain valuable information, it is possible that the corrections could bias the model causing a reduction in performance. In some cases it may be necessary to accept this performance penalty. For example, if there is insufficient uncorrupted training data available and there is no realistic possibility of obtaining more data from the system.

Another alternative is that once the anomalous data has been identified can be inter-

actively cleaned by combining expert opinions with the statistical models of the data to generate realistic alternative data (Masegosa and Moral 2012)

3.3.2 Filtering

Time series data (a signal) often contains a lot of noise. This is true of both condition data and event data. The term noise describes unwanted modifications to a signal that is being measured (Tuzlukov 2002). Noise can be introduced either due to a fault or deficiency in the monitoring system, or due to the monitoring system being sensitive enough to detect disturbances present due to the environment in which the system is operating. An example of this type of noise is background radiation (Penzias 1979). In many cases it can be difficult to distinguish between the noise and the useful information in a signal.

One method to reduce the amount of noise in the signal is to apply a digital filter. A digital filter is able to modify the signal to either enhance or reduce particular aspects of the signal by performing mathematical operations on the data. A digital filter differs from a standard analogue filter in that it operates on discrete samples of the continuous signal rather than on the continuous signal itself.

A measured signal is typically constructed from a large range of signals operating at different frequencies. Simple filters are used strip out the parts of a signal at unwanted frequencies (the noise) leaving only the parts of a signal that correspond to the system being measured.

A more detailed discussion of filtering is beyond the scope of this review.

3.4 Feature Generation

In principle it would be possible to use the raw data in building a model. However in practice most applications require some transformation of the raw data in order to extract sufficient information from the data (Bishop 1995). In general, using the raw data results in a poorly performing model either due to a poor signal to noise ratio within the raw data, or because many data–driven models are unable to scale up to the vast quantities of raw data that are available.

In essence, feature generation is used to present important aspects of the data that can be used to improve the signal to noise ratio.

There are a large variety of techniques for generating features from time series data. Most of the techniques that are applicable for an online prognostic system divide the signal into consecutive subsequences. Features are then generated that correspond to the period of the subsequence.

It should be noted that in general there is no optimal technique for generating features from a signal (Keogh and Kasetty 2003). However, each technique has different advantages that may suit itself to a particular dataset (Keogh and Kasetty 2003). As a result, selecting the best feature generation technique can be a matter of trial and error.

3.4.1 Statistical Features

Standard statistical features that describe the form of the signal can be used as features. These include the mean, peak, peak to peak interval and standard deviation in addition to

other higher order statistics such as root mean squared, skew and kurtosis (Jardine, Lin, and Banjevic 2006).

Using these statistical features requires the assumption that the properties of the system that are being measured are independent of time (Bishop 1995). For example, there is little point in comparing the mean values across two different periods of a signal that is constantly increasing as time progresses.

3.4.2 Discrete Fourier Transform

Any signal can be approximated through the superposition of a finite number of sine waves each possessing different amplitude, phase and frequency (Maimon and Rokach 2010). The Discrete Fourier Transform (DFT) is a linear transform that allows the computation of the energy from each of the sine waves. DFT is usually implemented by a Fast Fourier Transform (FFT) algorithm, thus the two terms are often used interchangeably. A FFT algorithm is able to compute the Fourier coefficients in $O(nlog(n))$ rather than $O(n^2)$ for the naive DFT implementation.

For a time series $\vec{x} = x_0, x_1, \ldots, x_{N-1}$ its DFT is a sequence of complex numbers $\vec{X} = X_0, X_1, \ldots, X_{N-1}$ such that:

$$DFT(X_k) = \sum_{t=0}^{N-1} x_t \exp(-i2\pi k \frac{t}{N})$$

where $i = \sqrt{-1}$.

For many interesting time series the energy of a signal is dominated by a small number of high energy components (Agrawal, Faloutsos, and Swami 1993). This means that low energy components (perhaps, including those that describe any noise) can be discarded for only a small loss of information (Maimon and Rokach 2010). The advantage of discarding information will be discussed in Section 3.5. It also allows the major trends in the signal, as determined by the high energy components, to be identified.

3.4.3 Discrete Wavelet Transform

Discrete Wavelet Transform (DWT) is similar to DFT in that it provides a representation of the signal by using a basis function. Yet where DFT requires that the signal be constructed using the sine (and cosine) function, DWT makes use of functions called wavelets in its approximation of the signal (Graps 1995).

A wavelet is an oscillatory function that has an average amplitude of zero.

The most basic wavelet function is the Haar wavelet. The Haar wavelet is defined as (Chan and Fu 1999):

$$haar_i^j(x_i) = haar(2^j x_i - i) \qquad i = 0, \ldots, 2^j - 1$$

where:

$$haar(t) = \begin{cases} 1 & : \quad 0.0 \leq t < 0.5 \\ -1 & : \quad 0.5 \leq t < 1.0 \\ 0 & : \quad otherwise \end{cases}$$

this is combined with the scaling function (Chan and Fu 1999):

$$scale(t) = \begin{cases} 1 & : \quad 0 < t < 1 \\ 0 & : \quad otherwise \end{cases}$$

during the DWT.

The Haar wavelet is limited in its potential applications because the function is discontinuous and therefore not continuously differentiable (Graps 1995). However this non continuity also means that it is well suited to modelling discontinuous signals such as those obtained from fault monitoring of systems (Lee 1999). Additionally, it is very fast to compute because the DWT using the Haar wavelet can be computed by a process of averaging adjacent samples with the scaling coefficient obtained by recording the difference between them (Chan and Fu 1999).

The advantage of DWT over DFT is that the DWT is able to describe the shape of the signal at multiple resolutions. The early wavelet coefficients describe the general shape of the entire signal whilst the later coefficients are able to describe local trends within portions of signal (Maimon and Rokach 2010). This also enables noise in the signal to be identified and then reduced when the signal is reconstructed (Maimon and Rokach 2010).

Finally, the Euclidean distance between two signals approximated using DTW with the Haar wavelet provides a lower bound on the distance of the original signals. (Chan and Fu 1999)

3.4.4 Piecewise Linear Approximation

Piecewise Linear Approximation (PLA) forms an approximation by considering the signal as consecutive subsequences of varying length. Each subsequence is represented by a linear approximation that is chosen to minimise the error:

$$error(\vec{x}) = \sum_{t=0}^{N-1} |\varepsilon_t|^2$$

where ε which is typically chosen to be the Euclidean distance between the line and the point (Pavlidis and Horowitz 1974).

It is important to note that this representation of the signal is not necessarily continuous, although the minimisation of $error(\vec{x})$ could be constrained to ensure that the PLA is continuous.

The algorithms for computing the PLA of a signal can be divided into three groups (Keogh et al. 2001). First is the sliding window approach (Keogh et al. 2001). Here a subsequence $\vec{s}$ is repeatedly increased in size until $error(\vec{s})$ exceeds a predefined threshold. At this point the next subsequence is computed.

A bottom-up approach can be used by starting with a linear approximation between all the samples and iteratively merging the subsequences which are sufficiently similar until some stopping condition is met (Keogh et al. 2001).

Alternatively, it can be computed in a top-down approach where the initial condition is a linear approximation of the entire signal. A sample is chosen as the point about which to divide the signal into subsequences. This is repeated recursively on the subsequences

until a stopping condition is reached (Keogh et al. 2001).

The primary advantage of PLA is that it can drastically reduce the amount of data needed to store an approximation of the time series. This is especially the case if the time series consists of long periods where the gradient of the linear approximation does not need to change.

Using the PLA representation of the signal as a feature vector can support for faster detection of both exact matches (Keogh et al. 2001) and similar matches using dynamic time warping (Park, Lee, and Chu 1999) compared to matching the full signal.

3.4.5 Piecewise Aggregate Approximation

The Piecewise Aggregate Approximation (PAA) algorithm divides the time series in two consecutive subsequences of fixed length. Each subsequence is then represented by the mean value of its samples (Keogh and Pazzani 2000).

The primary advantage of this approach is that it is extremely simple and fast to calculate. In addition it retains the property that the Euclidean distance between the PAA of two subsequences provides a lower bound on the distance between the raw subsequences (Yi and Faloutsos 2000).

3.4.6 Binning

Binning is the process by which a continuous signal is converted into a sequence of discrete values. The range of the signal is divided into non–overlapping regions with a bin assigned to each region. There are many strategies for determining where the region boundaries should lie in the data range. In general the best strategy is data dependent, however this problem is discussed further in Section 5.2.1. The binned representation is generated by taking each value of the signal and assigning it the bin corresponding to the region within which it falls.

This approach requires a lookup table to define a distance between bins in order to compute the similarity between two time series since there is no mathematical relationship between arbitrary bins. However it is possible to define the lookup tables so that the distance between two binned representations of a signal can provide a lower bound on their Euclidean distance (Lin et al. 2003).

The primary advantage of this approach is its ability to use a small amount of memory to represent a time series whilst still retaining the ability to perform an efficient comparison (Camerra et al. 2010). However the discrete nature of the features generated in this way makes it very useful for allowing algorithms that work on discrete data to be applied to time series data (Lin et al. 2003).

3.5 Feature Selection

Once some features have been generated via one or more of the methods described in Section 3.4 it is then necessary to select a subset of features that will be learned by the model. Feature selection is the process of identifying the subset of features that contribute the most to the output of the system. This influential subset is then used to train the data models rather than the entire set of features generated.

Using a subset of features results in a reduction in the available information and as a result, if performed incorrectly, this can reduce the capability of the model to distinguish between samples (Fukunaga 1990).

However for the majority of algorithms, beyond a certain number of features, the performance of the models produced starts to degrade (Bishop 1995). This is known as the "curse of dimensionality" (Bellman 1961). Another reason for the desire to reduce the number of features is concerned with the physical constraints of the computer (Camerra et al. 2010). Reducing the number of features considered ensures that a larger number of samples can be stored within the computer's memory.

Time series data in particular usually has a large number of highly correlated features (Chakrabarti et al. 2002). As a result the selection of the most influential features is particularly important. The aim, therefore, of feature selection is to reduce the number of features used whilst retaining as much information about the data as possible (Bishop 1995).

There is no general solution for selecting the optimal subset of features for a given solution. Each system requires specific knowledge of, and often investigation into the system in order to identify the subset of features that result in acceptable performance (Maimon and Rokach 2010).

3.5.1 Expert Knowledge

In many cases, expert knowledge can be used to guide the selection of important features. The experts are people who have prior knowledge about the system being monitored (Bishop 1995). Often these are the engineers who are responsible for the operation and maintenance of the system, or the architects who originally designed it.

The experts often know which features can be ignored and which ones are important to consider. However the expert knowledge available may not be sufficient to identify all the influential features. Also different experts may not agree on which features are the most important or even on how to measure the features they consider to be important (Robert, Guilpin, and Limoge 1999).

3.5.2 Principal Component Analysis

Principal Component Analysis (PCA) is the process of identifying the principal components of a data set (Jolliffe 2002). The first Principal Component (P_0) of a data set is the vector that describes the axis along which the largest variance can be seen. The second principle component (P_1) is a vector, perpendicular to P_0, that illustrates the next largest variance across the data set.

For a dataset X consisting of s samples and m feature then there are a corresponding s perpendicular principle components. In order to reduce the number of features in X to n features where $n < m$ it is necessary to form a $s \times n$ transformation matrix W. The transformation matrix is constructed by taking the n principle components that represent the highest variance.

$$W = [P_0, P_1, \ldots, P_{n-1}] \tag{3.1}$$

The reduced data Y set that retains the maximum variance within the X set with only

n features can then be computed as follows (Jolliffe 2002):

$$Y = W^T \times X \tag{3.2}$$

3.6 Classification and Anomaly Detection

Classification is the process by which new samples are assigned to one of a predefined number of discrete classes (Bishop 1995). The act of assigning a class to a sample is performed by a function known as a classifier. A classifier is defined as:

$$classifier(s) = \begin{cases} C_1 & : \quad s \in C_1 \\ & \vdots \\ C_N & : \quad s \in C_N \end{cases}$$

where s is the new sample and $\{C_1, \ldots, C_N\}$ is the set of N classes.

Many classification algorithms only function as binary classifiers, that is a classifier where $N = 2$. However multiple binary classifiers can be combined to produce a general purpose classifier (Fukunaga 1990; Burges 1998) for an arbitrary number of classes.

Anomaly detection is the process by which new samples are identified that do not belong to one of the predefined classes. Typically the presence of anomalies in a system indicates that either the model of the system incomplete or that there is a fault in either the monitoring system or the system being monitored.

A good anomaly detection algorithm will maximise the identification of anomalies and minimise the number of samples falsely identified as being anomalous (Markou and Singh 2003a)

When building a data–driven anomaly detector, there is an assumption that the normal operation of a system as modelled in the data is stable over a reasonable period of time (Markou and Singh 2003a). If the normal operation of a system is not stable or if the system is not observed in all normal states of operation then the algorithm will not accurately reflect the current state of the system. As a result, it can generate a large number of false positives or need to be regularly retrained for the current circumstances (Markou and Singh 2003b).

The remaining sections of this Chapter will present various algorithms that can be used for either classification or anomaly detection. The first step when building either a classifier or an anomaly detector is usually to extract the same features from the new sample as were used in the training set (Lin et al. 2003). Therefore all the algorithms discussed in the following Sections are assumed to be using the feature vector of a sample (as discussed in Section 3.4) rather than the raw sample data.

3.7 Decision Trees

A decision tree classifier is a function that recursively divides the feature space in order to apply a classification to a sample (Maimon and Rokach 2010). A decision tree forms a directed tree with a single "root" node that has no incoming edges. Each leaf of the tree

is assigned to a single class, should a leaf node be reached whilst traversing the tree then the sample is assigned the classification corresponding to the leaf node.

The primary advantage of this approach over the others discussed below is that the process by which a certain sample is assigned a classification is readily comprehensible to humans (Geurts 2001; Wehenkel 1998). In addition a classification is relatively cheap to compute using the decision tree.

A threshold function could be considered as a special case of a decision tree with depth 1. Thresholds are widely used in prognostic systems, primarily because they are very simple to both implement and to understand. However it should be noted that, despite their widespread application, thresholds are very limited in their classification capabilities (Bishop 1995).

Training a decision tree to classify a dataset is a difficult problem. Indeed to generate a minimal decision tree that correctly classifies all samples of the training set has been shown to be NP-hard (Hancock et al. 1996). Therefore it is not feasible to generate an optimal decision tree for the type of large and complex systems that are being considered (Maimon and Rokach 2010).

A heuristic approach can therefore be adopted to generate decision trees with acceptable performance. The most commonly used method is a top-down approach (Maimon and Rokach 2010). Here the goal is that the nodes near the root differentiate between the largest differences in the dataset, whilst the nodes nearer the leaves are concerned with fine grained differences. Usually the criteria for determining which features to use for a decision at each node are based on information theory, i.e. maximising the amount of information gained at each decision (Maimon and Rokach 2010). However there are several other approaches for the selection criteria (Geurts 2001; Ferri, Flach, and Hernandez-Orallo 2002) and no criteria has been found to be optimal for all datasets (Maimon and Rokach 2010).

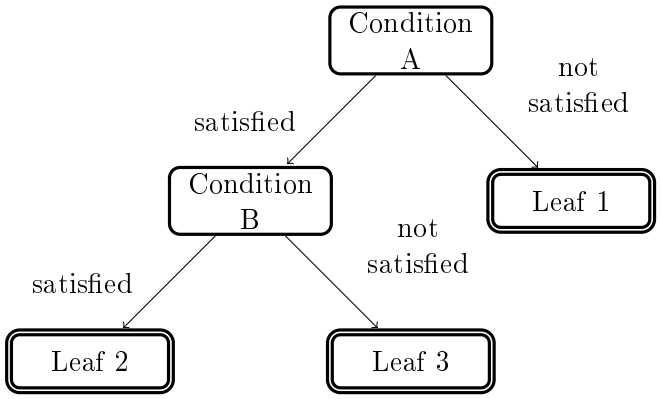


Figure 3.2: Example: Decision Tree

3.8 Statistical Approaches

The statistical approach to classification is to compute the probability P that the sample belongs to each class C_i, i.e. :

$$P(s \in C_i), i = 1 \ldots N$$

where N is the number of predefined classes. The sample is assigned to the class with the highest probability of containing the sample.

In the case of anomaly detection, if the largest value of P falls below some predefined threshold, then the sample is considered to be anomalous (Markou and Singh 2003a).

If the probability density function for data produced by the system is known then this approach allows the creation of an optimal classifier for the system using Bayes' theorem (Fukunaga 1990). Bayes' theorem provides a method by which observations, such as the time series data, can be interpreted in the context of previous knowledge, expressed as the probability density function, to predict the expected outcome (Stone 2013). However the true probability density function is usually unknown and therefore has to be estimated from the training samples provided.

The statistical methods for training a classifier can be grouped in to two distinct categories, the parametric and the non-parametric methods.

The parametric methods require that the system produces data samples according to a known distribution type, and that the distribution parameters are either known or are assumed to be correct (Fukunaga 1990). However in the case of real systems that are being monitored, the distribution and parameters of the data are typically unknown. Indeed it may even vary with respect to time or other environmental factors. As a result the parametric techniques have little practical application to large and complex systems (Markou and Singh 2003a).

The non-parametric methods, in contrast, do not make any assumption about the data distribution (Markou and Singh 2003a) and rely solely on the provided training sample. This means that computing accurate estimated density functions is very difficult (Fukunaga 1990). Therefore the estimates will be less reliable and subject to the bias in the training samples (Fukunaga 1990).

The most common method for estimating the probability density function is the Kernel Density Estimation method (Silverman 1986). The density estimation function of a training set $S = s_0, \ldots, s_{N-1}$ is given by:

$$density\_estimate(s) = \frac{1}{Nh} \sum_{i=0}^{N-1} K\left(\frac{s - s_i}{h}\right)$$

where K is a kernel function that integrates to 1 and h is the width of the kernel window. Commonly the kernel function K will be the standard normal density function ϕ where:

$$\phi(x) = \frac{e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}} \tag{3.3}$$

The disadvantage of this approach is that in order for the density estimation to be accurate, a large number of training samples are needed. However a large number of samples makes the computation of the density estimate very slow (Bishop 1995). Although modern computation capabilities may alleviate this somewhat.

Another issue with these approaches is that they are often not practical for use with non–stationary systems where the system will regularly change over time (Maimon and Rokach 2010). This is because the probability density function will need to be updated in order to reflect the newly changed system which will likely necessitate a large number of

new training samples that reflect the changed system.

3.9 Artificial Neural Networks

Artificial Neural Networks (ANN) are a technique for supporting general parameters on either a linear or a non-linear mapping from the training samples (as inputs) to their classifications (as outputs) (Bishop 1995). Artificial neural networks were inspired by the biological neural networks of synapses and neurons found in an animal's nervous system. As a result, an ANN can model complex behaviour through the interactions of simple processing units.

The capabilities of an ANN depend on the number of layers in the network. A single layer neural network consists of a set of weights on the inputs and a non-linear function that determines the output of a node. Successive layers take the output of nodes in previous layer and apply their own weights to a, possibly non–linear, function in order to determine the output of the layer. Figure 3.3 provides an example of a three-layer neural network.

A single layer network can achieve optimal classification of new input samples, provided that the samples are linearly separable (Bishop 1995). For data sets where the samples cannot be linearly separated, additional layers are needed. Theoretically a neural network with an input layer, output layer and a single hidden layer as depicted in Figure 3.3 should be sufficient to approximate any continuous function (Hornik 1991). However, in practice it is often easier to allow additional hidden layers in order to aid the training of the network.

Determining the number of nodes in the network and the weights on the inputs to those nodes is a difficult problem. This is typically achieved by using an error function that describes the performance of the network on the training set. This error function is then minimised with respect to the weights and the nodes that describe the network. The are a large number of optimisation algorithms that can be used to minimise the error function (Bishop 1995) however a discussion of these is beyond the scope of this review.

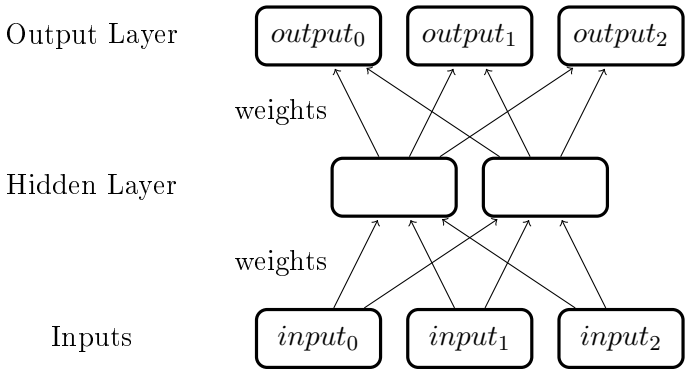


Figure 3.3: Neural Network Representation

Neural network based detectors are often difficult to train (Patcha and Park 2007). This can make them unsuitable for some types of system, particularly those that will require periodic retraining to adjust to changes in the operating conditions (Markou and Singh 2003b).

However because neural networks do not require much prior knowledge about the data to build a detector, neural network techniques are commonly applied to anomaly detection

problems (Markou and Singh 2003b; Patcha and Park 2007).

The Multi-Layer Perceptron (MLP) technique has been applied to detecting anomalous user actions on computers (Ryan, Lin, and Miikkulainen 1998). Here the network provides a confidence about the sample classifications that it produces. This is achieved by evaluating the differences between the weights that are applied to the output neurones (Vasconcelos, Fairhurst, and Bisset 1995). Similar to the statistical approaches described in Section 3.8, if the confidence falls below a particular threshold then the sample is considered to be anomalous.

The performance of the MLP network can be improved by assuming that samples which are deemed anomalous by a very large margin have been classified correctly. These samples can then be used as negative examples to retrain the network (Vasconcelos, Fairhurst, and Bisset 1995).

3.9.1 Support Vector Machines

Support Vector Machines (SVM) are another form of neural network. It has been shown that an SVM using a linear basis function is equivalent to the original Perceptron neural network (Collobert and Bengio 2004) and that an SVM using the Sigmoid basis function is equivalent to a two-layer Multi-Layered Perceptron (Burges 1998).

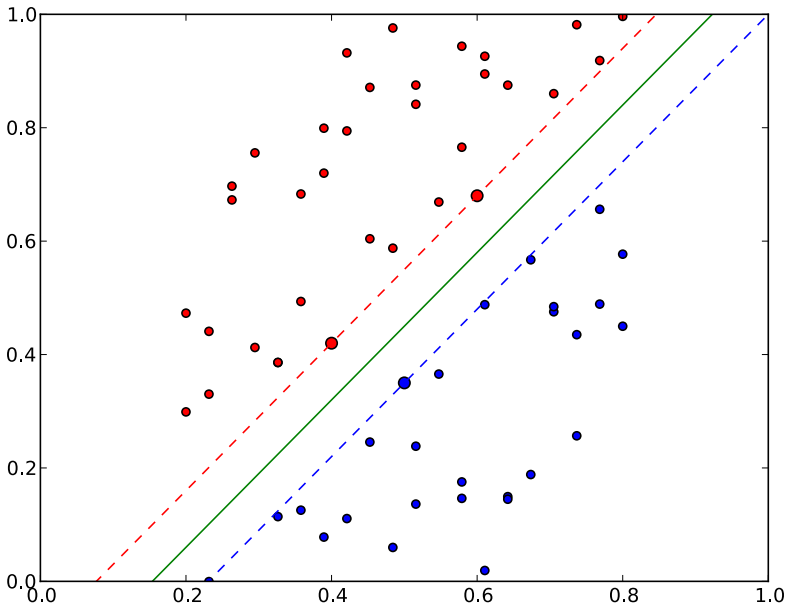


Figure 3.4: Example: Support Vector Machine with Linear Kernel

SVMs have been used for a wide range of systems monitoring and classification applications. For example, diagnosing faults in motors (Poyhonen, Jover, and Hyotyniemi 2004), chemical processes (Guo et al. 2003) and sleep state prediction (Fargus 2012).

An SVM is a binary classifier that seeks to find the optimal hyperplanes that separate the samples belonging to two distinct classes (Markou and Singh 2003b). The optimal hyperplane that forms the decision boundary between the two classes is chosen by maximising the margin between parallel support vectors. This is illustrated in Figure 3.4. The support

vectors of a training set D are identified by minimising $\frac{\|w\|^2}{2} + C\sum_{i=0}^{N-1}\varepsilon_i$ such that:

$$y_i(w \cdot x_i - b) > 1 - \varepsilon, \varepsilon > 0$$

where $x_i \in D$, $y_i \in \{-1, 1\}$ are the class labels of x_i, $\frac{|b|}{\|w\|}$ is the perpendicular distance of the decision boundary from the origin, C is the error penalty and ε is the error introduced when the datasets are not linearly separable (Burges 1998).

In order to support a non-linear decision boundary, SVMs employ the Kernel trick that enables the samples to be mapped into a higher (possibly infinite) dimensional space (Burges 1998). There are many suitable kernel functions. Some examples include the polynomial, sigmoidal and gaussian radial basis function kernels (Burges 1998).

SVMs have often been applied to anomaly detection (Tax and Duin 1999; Gardner et al. 2006; Ma and Perkins 2003). However there are typically only training samples available for the normal operation of the system for anomaly detection. As a result the decision boundary is constructed as a hypersphere with minimal radius that encloses the training samples (Tax and Duin 1999) rather than the optimal boundary between the two classes.

This approach requires that the training data accurately represents the boundaries of normal system operation. If the training data does not contain samples near to the boundaries then the radius of the hypersphere will be too small. A small radius would increase probability of the occurrence of false positives while a large radius increases the occurrence of false negatives.

3.10 Artificial Immune Systems

Artificial Immune Systems (AIS) are another form of biologically inspired algorithm (Forrest et al. 1994). AIS algorithms attempt to learn the normal operation of a system by generating a large number of detectors, each detector is a binary classifier intended to learn a particular form of anomalous sample.

Since there is no training data for anomalous samples, a negative selection algorithm is used to identify detectors that might detect anomalous samples (Dasgupta and Forrest 1996). A negative selection algorithm is as follows. First a detector is generated. The training samples are then fed into the new detector. If it does not detect any of the training samples, then it is retained, otherwise it is discarded. This process is repeated many times until a large collection of detectors have been generated that do not detect any of the training samples (Forrest et al. 1994). In this way the AIS system is trained to recognise only samples that do not occur in normal operation.

When the AIS system is running, each new sample is given to the collection of detectors. If any of the detectors recognise a sample then that sample is assumed to be anomalous because the system is trained to ignore normal samples.

Clearly the method for generating the candidate detectors is important. Originally the detectors were generated randomly (Forrest et al. 1994). However a greedy algorithm that tries to distribute the generated detectors as much as possible (D'haeseleer, Forrest, and Helman 1996) and a clonal selection algorithm that uses mutation operations (Cutello et al. 2007) have also been used.

The problem with an AIS approach is that a huge number of detectors may be needed

to cover all possible anomalies. Indeed it may not be possible to cover all the possible anomalies, leaving gaps where an anomalous sample would go uncaught.

3.11 Search Based Approaches

The k–Nearest Neighbour (k–NN) algorithm is commonly used for time series classification tasks (Yu et al. 2011; Jiang et al. 2007). The reason for its popularity is due to the ease of implementation and relatively good performance across a large number of situations (Geurts 2001; Jiang et al. 2007). Indeed with an infinite of number training samples, the k–NN algorithm functions as an ideal classifier (Fukunaga 1990).

The k–NN algorithm is very simple. To classify a new sample q, the first step is to identify the k training samples that are the most similar to q. These k samples are called the neighbours of q. The classification applied to q is then determined based on the classes of q's neighbours. A common strategy is to simply assign q the most common classification of it's neighbours. This majority vote amongst the k neighbours provides a layer of protection against invalid states being learned by the model.

k–NN is also commonly used in the context of anomaly detection. In order to identify whether a sample is anomalous, the distance between q and its neighbours is compared to a threshold distance. If the distance to its neighbours exceeds the threshold then q is considered to be anomalous.

Identification of the k neighbours most similar to q is performed using a distance measure, the most commonly used distance measure being the Euclidean distance. The distance between q and every training sample is computed and the k training samples with the shortest distance to q are selected as its nearest neighbours.

There are a large number of distance measures that can be used in this algorithm and the choice of distance measure has a significant effect on the performance of the classifier (Fukunaga 1990; Jiang et al. 2007). Several common distance metrics that have been used for time series classification are discussed in Section 3.11.1

The primary disadvantage of k–NN is that the standard implementations do not scale well with large training sets which makes it unsuitable for many real applications (Li et al. 2002). Several algorithms that implement the k–NN algorithm, but do not suffer from this problem, albeit with certain trade–offs, are discussed later in this Section.

In Section 2.4 two major issues were identified that affect all monitoring systems. They are how to validate the monitoring system and how to determine who is liable for any errors that occur with the monitoring system. The k–NN algorithm has distinct advantages with respect to these issues.

Firstly, it is easy to understand how the k–NN algorithm works, as a result it is easier for human experts to verify any predictions about the system state that are made by k–NN based models. Secondly, k–NN models only returns samples that have previously been observed as an answer when queried. Assuming that only a small proportion of the states that are trained into the model are invalid, this means that the model will only provide valid system states as its predictions, although this does not necessarily aid in determining the implications of being in a particular system state.

These two properties make it easier to build confidence that the system is operating

correctly and free of errors.

3.11.1 Distance Measures

In order to compute the k–NN it is necessary to use some form of distance measure. A distance measure is a function that maps two feature vectors to a single value that is somehow representative of the similarity of the two vectors.

A distance measure D is also a distance metric if it satisfies the following constraints (Weinberger and Saul 2009):

$$D(x, y) + D(y, z) \geq D(x, z) \tag{3.4}$$

$$D(x, y) \geq 0 \tag{3.5}$$

$$D(x, y) = D(y, x) \tag{3.6}$$

$$D(x, y) = 0 \Leftrightarrow x = y \tag{3.7}$$

There are a huge number of distance measures that can be used. I shall briefly describe several measures that are commonly used with time series data.

L_p Norms When $p = 2$, this is the traditional Euclidean distance defined as:

$$euclidean(x, y) = \sqrt{\sum_{i=0}^{N-1} (x_i - y_i)^2} \tag{3.8}$$

This is the default distance metric used when there is no prior knowledge available about the data set (Jiang et al. 2007; Weinberger and Saul 2009).

Often the square root will be omitted as in Equation 3.9 because this distance function provides the same ordering of samples as the Euclidean distance and therefore will yield the same nearest neighbours and it omits the expensive computation of a square root required by the Euclidean distance.

$$squared\_euclidean(x, y) = euclidean(x, y)^2 = \sum_{i=0}^{N-1} (x_i - y_i)^2 \tag{3.9}$$

Unfortunately the Squared Euclidean distance is not a true metric as it does not satisfy the triangle inequality (Equation 3.4), this does not effect the result of a Linear Scan (Section 3.11.2), however it does mean that this optimisation cannot necessarily be applied to all other k–NN algorithms.

The problem with using Euclidean distance in time series classification is that it is unable to detect similarity between time series with different amplitudes. In some cases, such as where matches are desired to be of similar amplitude, this issue could be considered to be an advantage of this approach. However in other cases, it can be overcome by normalising the time series before computing the distance between samples (Goldin and Kanellakis 1995). However a stretched or compressed version of a similar time series cannot be identified (Maimon and Rokach 2010).

Dynamic Time Warping This distance measure allows two time series to be compared for similarities despite the features being out of phase (Berndt and Clifford 1994). The DTW distance for two sequences $X = x_0, \ldots, x_i, \ldots, x_{|X|-1}$ and $Y = y_0, \ldots, y_i, \ldots, y_{|Y|-1}$ is computed by first constructing a $|X|$ by $|Y|$ matrix where $element_{i,j} = squared\_euclidean(x_i, y_j)$. A warping path through the matrix $W = w_0, \ldots, w_{|W|}$ can be computed where $max(|X|, |Y|) \leq |W| \leq |X|+|Y|-1$ and $w_k = (i, j)_k$. The DTW distance is the path through the matrix that minimises the cumulative distance. DTW is computed by (Yu et al. 2011):

$$DTW(x_i, y_j) = squared\_euclidean(x_i, y_j) + min \begin{cases} DTW(x_{i-1}, y_j) \\ DTW(x_{i-1}, y_{j-1}) \\ DTW(x_i, y_{j-1}) \end{cases} \tag{3.10}$$

Unfortunately, this approach does not scale very well given that it is very computationally expensive (Vlachos et al. 2006). Additionally it does not work well in the presence of noise, since it will try to match the outliers in the time series (Vlachos et al. 2006).

Longest Common Subsequence This distance measure has similar advantages to DTW but also performs better in the presence of noise since it is able to ignore outliers during the matching process (Vlachos et al. 2006). The LCSS distance is computed by:

$$LCSS(x_i, y_j) = \begin{cases} 0 & : \quad i, j = 0 \\ LCSS(x_{i-1}, y_{j-1}) + 1 & : \quad |x_i - y_j| < \varepsilon \\ max(LCSS(x_i, y_{j-1}), LCSS(x_{i-1}, y_j)) & : \quad otherwise \end{cases} \tag{3.11}$$

However with a sufficiently large training dataset, it has been shown empirically that the performance of LCSS distance is no better than DTW (Ding et al. 2008). Therefore the advantage of this distance measure is its improved performance where only small datasets are available.

3.11.2 Linear Scan

The standard method for computing the nearest neighbours of a particular query is to perform a linear scan, comparing the query sample q with each of the samples $s \in S$. After each comparison, the Euclidean distance between q and s is then checked with the current set of neighbours, if the distance is less than the distance to the neighbour n_k, the current k-th nearest neighbour to q, then s is added to the set of neighbours and n_k is discarded. Once all samples have been compared against q then the surviving set of neighbours are known to be the k–nearest neighbours.

This approach has a number of advantages over other methods of computing the nearest neighbours. The first advantage is that the linear scan is an exact method, i.e. each query is guaranteed to provide the correct set of samples that are the closest to q. In addition, a linear scan does not require any preprocessing or training stages in order to operate. As a result it is generally very easy to implement and thus can be optimised to have a relatively low overhead for executing queries.

However there is a significant downside to the linear scan. The execution time of this algorithm is clearly $\mathcal{O}(N)$ where N is the number of samples in D. Therefore when N is large a query can take a long time to execute.

Typically, the majority of the execution time is spent computing the distance between q and $s \in S$. This is particularly the case when the number of features f in each sample is large. However there are several optimisations that can be used to reduce this computation time.

One way to speed the computation in a Linear Scan is to use a function that provides a fast lower bound on the distance measure in order to avoid the need for a full distance computation (Zezula et al. 2006). Using a lower bound it is possible to determine whether the sample can possibly be one of the nearest neighbours. If the lower bound is greater than the distance to n_k then it is unnecessary to compute the expensive distance measure since the sample can never be a nearest neighbour.

When computing a lower bound it is necessary to make a trade off between the tightness of the lower bound and the time required to compute it. A tighter bound enables more pruning of distance computations, however if the tight bound is expensive to compute then it may not result in a reduction of execution time. Depending on the distance measure used, there are a wide range of lower bounding functions that can be applied (Zezula et al. 2006; Rakthanmanon et al. 2012)

Additionally, it is often the case that the calculation can be abandoned before completing the computation of the distance measure (Rakthanmanon et al. 2012). For example, the typical method for computing the Euclidean distance involves iteratively adding the squared distance between each of the features. If the partial distance computation exceeds $squared\_euclidean(q, n_k)$ then there is no reason to continue the computation of the distance as the sample cannot be one of the nearest neighbours.

3.11.3 Spatial Partitioning Trees

In general the problem with the Linear Scan algorithm occurs when a large number of samples need to be compared with the query sample q. Therefore in order to speed up computation of the nearest neighbours it is necessary to reduce the number of distance computations required. Spatial partitioning is a common approach to achieving this goal. In essence spatial partitioning consists of repeatedly dividing the search space that encompasses D so that distance computations are only needed for the samples that lie within a subset of the partitions. These approaches make use of the triangle inequality constraint on distance metrics to partition the multidimensional space. This allows a large portion of the samples to be efficiently pruned from the search and therefore can significantly speed the computation of the nearest neighbours (Marteau 2008).

The spatial partitions are typically represented using a tree structure, with each branch of the tree representing a new partition. Clearly, a preprocessing or training stage is required by these methods to build the tree, as a result this approach is better suited to situations where multiple queries are going to be asked of D in order to offset the overhead introduced by the tree building stage.

There are a large number of spatial partitioning algorithms that have been applied to the kNN problem, such as: kD Trees (Bentley 1975), Ball Trees (Omohundro 1989), Cover

Trees (Beygelzimer, Kakade, and Langford 2006) and R–Trees (Agrawal et al. 1995).

In Section 3.11.3 the k–Dimensional (kD) Tree is examined as one of the oldest and most commonly used general spatial partitioning algorithms for the calculating the k-NN.

k–Dimensional Tree

The k–Dimensional Tree (Bentley 1975) is a multidimensional binary search tree, where k refers to the number of features in each sample.

At each level of the tree, the data space is divided into two partitions by a hyperplane that runs perpendicular to an axis that represents a specific feature within the data space. The location of the boundary hyperplane along the feature axis is determined by a pivot sample $p \in S$. It is important that p is chosen such that the tree will be balanced. Figure 3.5 provides an illustration of a kD Tree with 2 features.

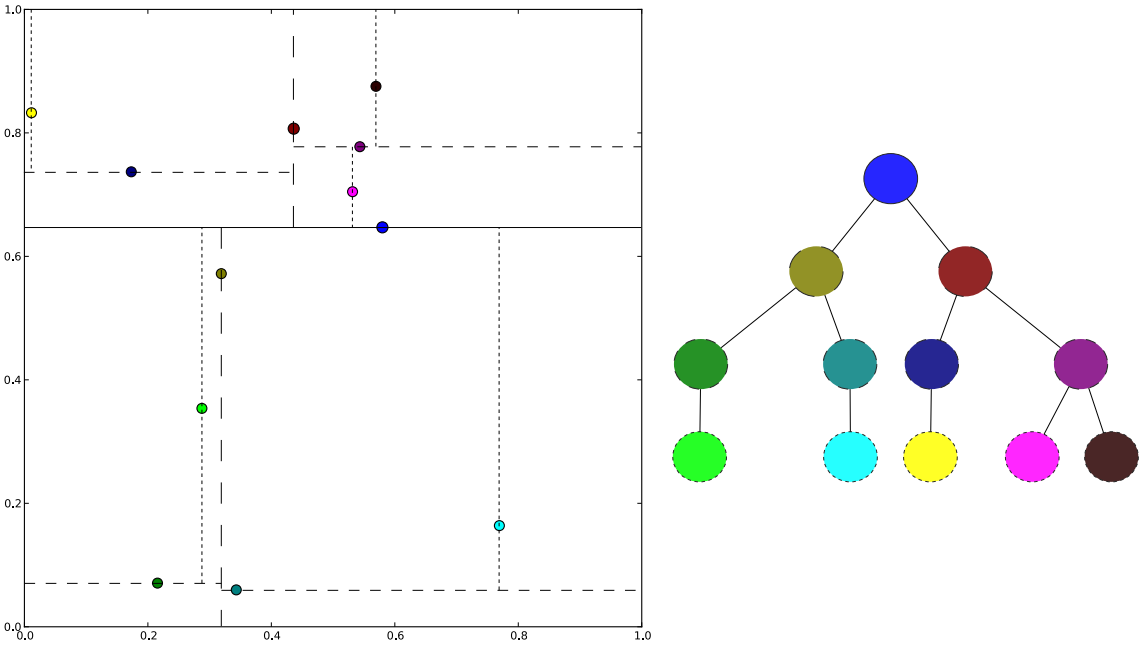


Figure 3.5: An Illustration of the kD Tree

This Figure shows how the kD Tree (shown right) is built from the samples in the data space (left). The lines through the data space represent the hyperplanes that form the partition boundaries. The colours of the nodes in the tree correspond to the same coloured sample in the data space.

During construction of the kD Tree it is necessary to select both the target feature and the pivot sample at each level of the tree. The target feature is typically selected by choosing the feature containing the largest spread in values before the partition occurs (Friedman, Bentley, and Finkel 1977). The pivot sample is chosen by selecting a sample with the median value in the feature dimension, this ensures that the tree remains balanced and thereby reduces the number of comparisons required during a query (Friedman, Bentley, and Finkel 1977). Once the feature and pivot have been chosen, the data is partitioned and the process is repeated at the next level for both sides of the partition.

Querying of the kD Tree is most easily understood as a backtracking recursive algorithm (Friedman, Bentley, and Finkel 1977). At each node of the tree, q is compared to the pivot value p. If the distance between p and q is less than the distance to the neighbour n_k,

the current k-th nearest neighbour to q, then q is added to the set of neighbours and n_k is discarded. If the node is a leaf, then the algorithm returns up the tree. Otherwise q is compared to p in the single pivot dimension of the node in order to determine which child node to be search next, this child is then searched. Once control returns from searching the first child node, a second test is performed to determine whether the other child needs to be searched, or whether that sub–tree can be pruned. This test checks whether there exists a point on the other side of the partition that is also contained by a ball of radius equal to the distance between q and n_k. If so then the child is searched, otherwise that branch does not need to be checked for neighbours.

This search algorithm is known to produce an exact result for the k–nearest neighbours in $\mathcal{O}(G(f) \times \log n)$ where $G(f)$ is an exponential function on the number of features f being searched (Shakhnarovich, Indyk, and Darrell 2006). As a result, when data spaces containing relatively few samples need to be searched, the kD Tree can be significantly faster than the linear scan, however it quickly becomes less efficient as the number of features increases. This is demonstrated in Chapter 6 where the kD Tree algorithm has a mean query time that is 24 times faster than Linear Scan with 10 features but degrades to be essentially equivalent, being just 6% faster for datasets with 100 features.

Dual Trees

If it is necessary to compute the nearest neighbours for a set of query samples $Q = q_0, q_1, \ldots, q_n$, significant reduction in computation time can be obtained by using a dual tree formulation to compute the neighbours for each $q \in Q$ simultaneously (Gray and Moore 2000).

The dual tree formulation requires that a tree be built at query time to contain the query samples $q \in Q$, this is in addition to the tree containing the samples $s \in S$ that is built during the training stage (March 2013).

The query is then performed by traversing both the query tree and the search tree simultaneously. At each pair of nodes, a lower bound distance is computed between the sample q_i as the query node and the sample s_i as the search node. If the lower bound exceeds the distance between q_i and its current k–th nearest neighbour the all search samples that are children of s_i can be pruned from the search, otherwise the child nodes are traversed as normal. The result of this pruning is that the k–nearest neighbours can be computed for all $q \in Q$ in the worst case of $\mathcal{O}(N)$ (Ram et al. 2009). This contrasts with the $\mathcal{O}(N \times log(N))$ required if the neighbours for each q_i are computed by querying the sample tree independently (Ram et al. 2009).

In order to obtain any advantage from this algorithm over standard tree based algorithms, it is necessary to batch many query samples together. As a result this does limit its potential use to non–real time applications.

Number of Features

The primary problem with all the spatial partitioning techniques is that they do not scale well with an increasing number of features. This is known as the curse of dimensionality. With the spatial partitioning techniques, increasing the number of features causes the

algorithm to degrade to essentially a linear scan (Weber, Schek, and Blott 1998). As a result the usefulness of these techniques is limited to datasets with relatively few features.

3.11.4 Approximate Nearest Neighbours

For very large datasets, such as those consisting of many features or a very large number of samples, the exact k–Nearest Neighbour algorithms discussed above may not be able to locate the nearest neighbours sufficiently quickly for many applications.

Fortunately, for many of these use cases, it is not always necessary to have the exact nearest neighbours. Often a set of neighbours that are close to q but not the true nearest neighbours would be sufficient. In these cases it is possible to trade accuracy for speed in finding the k–Nearest Neighbours. Algorithms that perform this trade off are referred to as Approximate Nearest Neighbour algorithms.

Clearly, for some datasets, the distribution of classes throughout the data space will mean that the use of an Approximate Nearest Neighbour algorithm will result in an incorrect classification. However, in practice, this effect can often be mitigated by selecting different features to represent each sample.

Locality Sensitive Hashing

The Locality Sensitive Hashing (LSH) approach forms the basis for some of the state of the art approximate k-NN algorithms. As with the tree based algorithms described in Section 3.11.3, LSH algorithms seek to reduce the number of comparisons that need to be computed in order to calculate the k–Nearest Neighbours to q. However rather than using a hierarchical space partitioning approach, LSH algorithms have a single layer of space partitions that are referred to as buckets (Datar et al. 2004).

A hashing function is used to map a sample into its relevant bucket. However unlike a typical hash function, locality sensitive hash functions are designed so that there is a high probability of collision between samples that are similar and a low probability of collision between samples that are dissimilar. LSH algorithms are able to provide a significant reduction in the query time in comparison to the spatial partitioning algorithms using the hash to directly access a single bucket to search for nearest neighbours rather than having to traverse a tree (Datar et al. 2004), albeit at the cost of some accuracy.

LSH algorithms require a training stage in order to assign each of the samples $s \in S$ to their specific buckets. Typically, many buckets will be empty and therefore only the buckets that contain any samples need to be retained.

In order to query the dataset, the hash function is applied to q and the relevant bucket is determined. The contents of this bucket are added to a set of candidate neighbours. Finally the candidate neighbours are then searched using a linear scan in order to find an approximation of the nearest neighbours to q. Since there are relatively few samples contained within each bucket, this linear scan is fast to execute.

Clearly for this approach to work, it is necessary to have an effective locality sensitive hashing function. In order to compute the nearest neighbours in Euclidean space, an effective locality sensitive hash function will need to place samples that have a small Euclidean distance between them into the same bucket with a high probability and place samples

with a large Euclidean distance between them into different buckets (Indyk and Motwani 1998).

Originally this was achieved by a process of embedding the Euclidean space into Hamming space in order to implement the hash (Indyk and Motwani 1998). However this embedding could add a significant overhead and source of errors to the algorithm (Datar et al. 2004). As a result, a family of hash functions that operate directly within Euclidean space were developed. The process for creating such a hash function is described below.

The first step is to generate a random sample a consisting of a feature drawn from a Gaussian distribution for each dimension in the data to be searched. In order to generate the hash of sample s, the dot product $a.s$ is computed and a random variable b is added, this forms a projection onto a line that represents the data space. The data space is partitioned along this line using equi–width bins of width w. In essence w specifies the size of each bucket in the data space and it is chosen via an optimisation algorithm to balance the speed of a query and the accuracy of the result (Datar et al. 2004). It is worth noting that many implementations choose w so that a sample can be assigned to only 2 bins, this makes the binning function fast and trivial to implement.

Formally, a hash function $h(s)$ is defined as:

$$h(s) = bin_w(a.s + b) \tag{3.12}$$

Each bin is assigned to a specific bucket therefore $h(s)$ provided an index to the correct data space partition for s (Datar et al. 2004). This hash can be considered to be locality sensitive for Euclidean space because there is a high probability that when the distance between two samples $euclidean(s_1, s_2)$ is small, the distance $\|a.s_1 - a.s_2\|$ will also be small.

However the space partitions from a single hash $h(s)$ can be relatively large, especially if w is also large. As a result, it is typically the case that a hash function $H(s)$ is constructed by concatenating the results of l hash functions $h_0(s), h_1(s), \ldots, h_{l-1}(s)$ as defined in Equation 3.13.

$$H(s) = [h_0(s), h_1(s), \ldots, h_{l-1}(s)] \tag{3.13}$$

In this way the function $H(s)$ is used to define the bucket that sample s belongs to. Figure 3.7 illustrates how the bucket is formed around a query sample for a hash function with $l = 2$. Clearly, a larger l will reduce the size of the bucket and further prune the number of samples that have to be searched. However this will also increase the time required to compute $H(s)$.

An alternative approach is to vary the value of l for each bucket (Bawa, Condie, and Ganesan 2005). In this situation, the value of l for each bucket is set to be just large enough to uniquely identify the bucket. It is possible that some buckets will be very similar and therefore more hashes will be required to distinguish between those buckets than between very dissimilar buckets. A prefix tree can then be constructed from the set of variable length hashes that represent each of the buckets. An example of this is given in Figure 3.6. During a query, this tree can then be traversed to locate the relevant bucket.

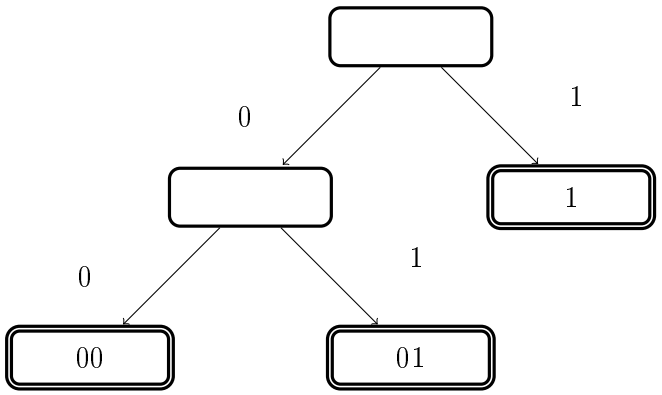


Figure 3.6: An example of a prefix tree containing the variable length binary hashes $\{00, 01, 1\}$

In order to improve the likelihood that the true nearest neighbours are contained within the set of candidate neighbours identified from the buckets, an ensemble of hash functions $H_0(s), H_1(s) \ldots, H_n(s)$ can be used. The set of candidate neighbours are then composed of the content of all buckets identified by one of the random hashes. It is from this larger set of candidate neighbours that the approximate k–nearest neighbours are selected.

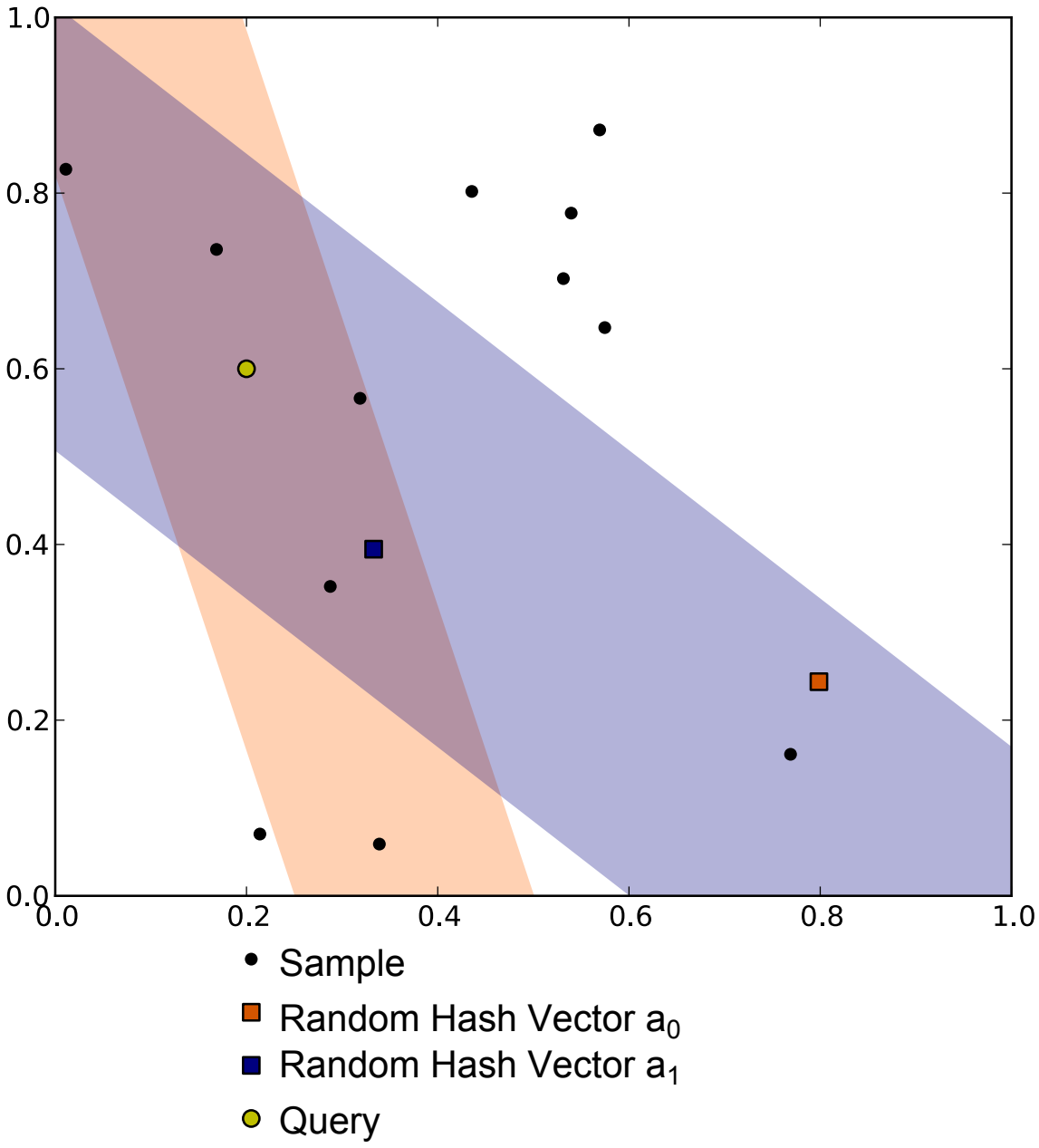


Figure 3.7: An Illustration of Locality Sensitive Hashing

This Figure shows how Locality Sensitive Hashing is used to prune the number of samples that need to be evaluated to compute the k–Nearest Neighbours. In this example there are two hash functions, the red hash derived from the random sample a_0 and the green hash derived from a_1. The area of the data space that corresponds to the same hashed bucket as the query sample has been shaded with the relevant colour. Each partition has a width of 0.2 and, for simplicity, the random offset b has been set to 0. When computing the approximate nearest neighbours, the set of candidate neighbours are those that lie within the same bucket as the query sample. The query bucket can be seen as the intersection of the two shaded areas.

3.12 Model Evaluation

The evaluation of a data–driven model is an essential part of the model development process. The decision of whether to deploy the model or to continue development is based on the outcome of the model evaluation.

One of the key characteristics of the model that need to be evaluated is its accuracy. Specifically it is important that the number of false positives and false negatives are minimised. A false positive occurs when an error is predicted but does not occur while a false negative occurs then there is an error that was not predicted by the model.

A low false negative rate is very important for all systems. However the level of false positives that can be accepted will depend on the application and the cost involved with verifying the prediction. A low false negative rate can usually be achieved at the cost of many false positives. In most applications it is necessary to tune the models to achieve the required balance because too many false positives will cause the users to lose trust in the system.

Another key characteristic is the speed of prediction. In a real time monitoring system it is essential that any failures that can be predicted by a model are done so before the failure actually occurs. It is also important that the monitoring system can make predictions at a fast enough rate to handle new data that is recorded from the target system.

3.13 Summary

In this Chapter, the common steps needed to develop a data–driven model for use in system prognostics and diagnostics have been introduced. First data about the target system must be obtained; this data then has to be cleaned and filtered to remove noise and artefacts of the acquisition process.

The next step is to generate features based on the obtained data. Several methods for generating features from time series data have been discussed, including; basic statistics, frequency analysis of the signals and strategies for compressing the representation of the actual signal. This was followed by a brief discussion of how to identify which are the best features to use.

Once the important features have been identified, the next stage is to train a model to learn the features of the data. Two different approaches to building a model were presented. Classification of a system involves selecting the predefined operating state that most closely resembles the current operation of the system whereas anomaly detection involves identifying when the current operation of the system has moved outside what is considered to be its normal operating envelope.

Several common methods for accomplishing both classification and anomaly detection on time series data were discussed along with discussions for when it is best to choose each method. Using a search based approach to implement both classification and anomaly detection was also discussed and followed by a presentation of several algorithms that can be used to implement such a search based approach.

Finally the Chapter concluded with a brief discussion about how to evaluate the performance of a data–driven model.

In the next Chapter, a specific data–driven model called AURA Alert will be discussed in detail. However the development pipeline illustrated in Figure 3.1 and discussed throughout this Chapter applies equally to the development of an AURA Alert data–driven model.

Chapter 4

Advanced Uncertain Reasoning Architecture

4.1 Introduction

The Advanced Uncertain Reasoning Architecture (AURA) is a framework that provides a set of methods that make use of binary associative neural networks called Correlation Matrix Memories (CMM) to perform pattern matching (Austin 1996). In this Chapter the process by which an AURA CMM can be used to build a k-NN based anomaly detection model will be presented. Section 4.2 will provide an overview of CMMs and how input and output tokens are used to store and recall data from them. Section 4.6 will describe the AURA Alert model for k-NN based anomaly detection that is built on top of the AURA framework. Sections 4.3 and 4.4 describe several methods for generating the necessary input and output tokens respectively. Finally Section 4.5 describes several algorithms for applying a threshold to the output from a CMM in order to decode the results of a query.

4.2 Correlation Matrix Memories

A Correlation Matrix Memory (CMM) is a single layer neural network in which the input and output neurones are fully connected. The CMM is able to learn associations between the input neurones and output neurones. A CMM with n input neurones and m output neurones is represented by an $n \times m$ matrix. The elements of an AURA based CMM contain binary values that represent the weights of the network as shown in Figure 4.1.

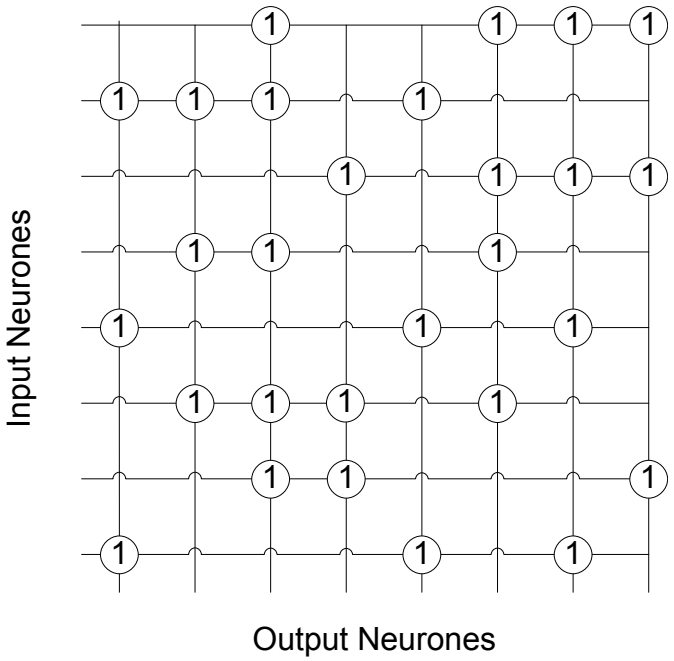


Figure 4.1: Example Correlation Matrix Memory

The inputs and outputs to the CMM are specified as a vector of binary bits called tokens. The number of bits required to represent a token is the token length. In general shorter tokens are preferred since the size and memory usage of an AURA CMM is dependent on the length of the stored tokens.

The number of bits in the token that are set to 1 is the token weight. When storing associations between input and output tokens within a CMM it is important that all input tokens have the same weight and all output tokens have the same weight. This is known as a fixed weight coding. In Casasent and Telfer (1992) it has been shown that a CMM can store the associations between a large number of pairs of input and output tokens when the tokens have a fixed weight.

The number of sample pairs that can be stored within a CMM before there is guaranteed to be at least a single bit error during recall is calculated as shown in Equation 4.1 (Hobson 2011).

$$n\_pairs = \frac{\ln\left(1 - \frac{1}{output\_length^{\frac{1}{input\_weight}}}\right)}{1 - \left(\frac{output\_weight \times input\_weight}{output\_length \times input\_length}\right)} \tag{4.1}$$

As a result, in order to maximise the number of associations that can be stored within a CMM of a given size, it is also important that the tokens are very sparse. A sparse token is one where the proportion of bits set in the token is relatively low in comparison to its length. This is important for performance reasons because the time required to store and retrieve samples from a CMM is highly dependent on the size of the CMM and the number of bits set within it. In order to achieve the required sparsity, the optimal token weight has been shown to be $log_2(token\_length)$ (Hobson 2011; Willshaw, Buneman, and Longuet-Higgins 1969).

Finally a CMM also provides fault tolerance as well as the ability to generalise between associations that have been learned (Hobson 2011).

The inputs to, and outputs from, a binary CMM are required to be discrete. However, often samples of continuous data have to be learned by the CMM. As a result the first step when either training the CMM to learn a sample or querying the CMM to retrieve a sample is to discretise the sample via binning (Section 3.4.6). Once the sample has been binned, the necessary tokens for training and querying can be generated.

4.2.1 Training

The training of a CMM is performed in a deterministic manner. The CMM is represented by a binary matrix M. An untrained CMM (M_0) is initialised with all weights set to 0. To train the network to recognise a training sample x_i, a pair of input and output tokens, I_i and O_i respectively, are generated that correspond to x_i. There are various alternatives for generating I_i and O_i from x_i. The generation of input tokens are discussed in detail in Section 4.3 and the generation of output tokens are discussed in Section 4.4. Once the input and output tokens have been generated, the CMM is then updated via Equation 4.2 in order to store the association between I_i and O_i in the CMM.

$$M_i = M_{i-1} \cup (I_i \times O_i) \tag{4.2}$$

This procedure can be repeated for every sample in the training set (Liang and Austin 2005). As a result the entire dataset can be learned by the network with a single pass of the training set.

4.2.2 Retrieval

In order to query a trained CMM M for the output token that is associated with the sample x_j, the input token I_j must first be generated from x_j. The CMM output scores S_j are then computed by the operation:

$$S_j = I_j \times M \tag{4.3}$$

Finally a threshold can be applied to the output scores S_j in order to retrieve the output token O_j. Depending on the type of threshold applied it is possible to retrieve multiple output tokens that are associated to samples that are similar to x_j(Liang and Austin 2005). There are many possible threshold algorithms that can be used. These algorithms are discussed further in Section 4.5.

4.3 Input Tokens

In this Section, the process by which a feature vector of a sample can be converted into an input token for a CMM is discussed. This process is referred to as encoding the sample. It is essential that any input tokens that are generated will be similar if the original feature vectors were also similar and that they will be dissimilar if the original feature vectors were dissimilar so that the CMM is able to generalise to unseen samples.

The feature vectors of two samples are considered to be similar when they are separated by a small *distance*. When considering tokens, two tokens are considered to be similar when they have a large number of overlapping bits. The number of bits that need to

overlap in order represent a specific level of similarity is dependent on the method used to encode the tokens.

The Parabolic Kernel method, described in Section 4.3.1, is currently the most commonly used method to encode continuous features for AURA CMMs. An alternative method Overlapped Binary Code Construction (OBCC) is a more recent alternative and is described in Section 4.3.2.

4.3.1 Parabolic Kernel Tokens

The Parabolic Kernel method for generating input tokens produces different tokens for both training the CMM and querying the CMM. The query token makes use of a feature of AURA CMMs that allows weights to be applied to the bits of an input token.

The CMM stores associations between binary tokens and as a result the training input token has to be a binary token as normal. In order to encode a token from a single feature of m bins a vector of m bits is required. A feature assigned to bin i simply has the ith bit of its vector set to 1 (Hodge and Austin 2005).

A query to an AURA CMM supports weighted query tokens where a weight is applied to each bit that is set in the token. During a query, if there is a match, the bit weight is added to the output score for the column.

In order to encode an input token for querying a CMM, a parabolic kernel is then superimposed over the bits of each feature in the training token. This is to overcome the boundary effect of the binning process by weighting the CMM output scores in favour of the closest training samples (Weeks et al. 2003).

The weight of bit j in the parabolic kernel for a feature with bit i set is given by:

$$weight(j) = max\left(\frac{m^2}{2} - (i-j)^2, 0\right) \tag{4.4}$$

In order to encode multiple features, the vectors the of bits representing each individual feature are simply concatenated together.

4.3.2 Overlapped Binary Code Constructed Tokens

Overlapped Binary Code Construction (OBCC)(Hobson 2011) defines a method for creating sparse, fixed weight, binary input tokens that ensure a predefined number of overlapping bits between two tokens created to encode a pair of samples. This method is based on the earlier Sparse Similarity Preserving Codes (SSPC) method of (Palm, Schwenker, and Sommer 1994). The OBCC method can be described in three stages. The first stage is to create an overlap matrix. The overlap matrix is used to specify the number of overlapping bits between a pair of tokens. This is based on a pairwise *distance* between the entire range of values to be encoded and described in Section 4.3.2.1.

Note that unlike the Parabolic Kernel approach described above, by using this method it is possible to directly encode multiple features into a token without the need for concatenation 5.2.2. The overlap matrix need simply specify the number of overlapping bits between multi–feature values.

The next stage is to generate the sparse tokens from the overlap matrix via a process called clique decomposition. This is described in Section 4.3.2.2. It is through the selection

of cliques for this stage that OBCC differs from SSPC.

Finally the tokens are padded to ensure that they have a fixed weight. This is discussed in Section 4.3.2.3

4.3.2.1 Overlap Matrix

The first step in creating an overlap matrix is to compute the pairwise distance matrix between all the values that need to be encoded. As discussed in Section 3.11.1, a *distance* function is considered to be a metric function if it is non–negative, symmetrical, satisfies the triangle inequality and provides a result of 0 between identical values.

In order to encode tokens in which the overlap between values can approximate the distance function it is necessary that the *distance* function is non–negative, symmetrical and provides a results of 0 between identical values. However it is not required to satisfy the triangle inequality.

Binary input tokens can only support a finite level of precision in representing the distance between tokens. In order to increase this level of precision it is necessary to produce increasingly large tokens.

In order to counter this the concept of a maximum pairwise distance was introduced (Hobson 2011). This maximum distance specifies the largest pairwise distance allowed between two values for those two values to be considered similar to each other. Any values with a greater pairwise distance are considered to have no similarity, this translates into having no overlapping bits in their respective tokens. As a result the distance matrix is computed as:

$$D_{ij} = \begin{cases} distance(i,j) & : \quad distance(i,j) < max\_distance \\ \infty & : \quad otherwise \end{cases} \tag{4.5}$$

Clearly the number of overlapping bits between two binary tokens has to be a whole number. Therefore in order to convert the distance matrix into an overlap matrix the distances have to be converted to integer values.

Simply rounding the float values to the nearest integer is not likely to result in a useful overlap matrix. Consider the situation in which the bins represent normalised values in the range $(-1, 1)$ and a maximum distance of 0.5 representing a quarter of this range. Rounding the Euclidean distance between bins will result in only two potential values, 1 or 0. This is a problem because it only allows the discrimination between two distance levels, one of which already represents no similarity between values.

It is necessary, therefore, to scale the distance matrix so that it can be rounded to represent an appropriate number of discrete distance levels. The number of discrete distances is determined by a maximum overlap parameter. This specifies the largest number of overlapping bits between two tokens that represent different values. The number of overlapping bits between equal values is the weight of the token and is not known at this point in the process.

In order to map the distance matrix into an overlap matrix it is suggested that the continuous distance values can be grouped together and represented by an integer level of similarity (Palm, Schwenker, and Sommer 1994). One strategy for performing this is to divide the distance range $(0, max\_distance)$ into $max\_overlap$ equi–width bins. Where

max_overlap is a parameter that specifies the required level of precision. Each bin corresponds to a specified number of overlapping bits that will represent the distance, with the bins containing smaller distances corresponding to the larger overlaps. This process is performed by the *assign_overlap* function and is illustrated in Figure 4.2.

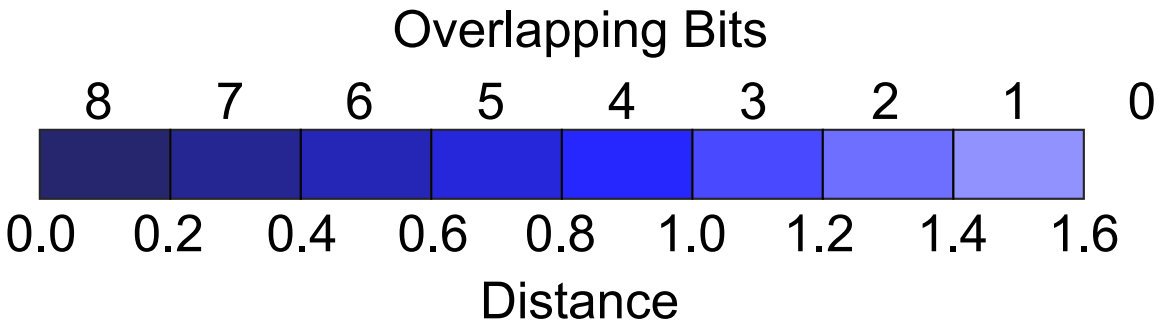


Figure 4.2: Converting continuous distance values to discrete overlaps

.

This Figure shows how a pairwise distance is mapped into the desired number of overlapping bits as part of the *assign_overlap* function. The continuous distance is given by the lower values and defines the bin boundaries. The overlap corresponding to each bin is given by the upper values. Each distance is assigned an overlap determined by the bin within which it falls. Any distance greater than 1.6 is assigned an overlap of 0.

Using the *assign_overlap* function, the overlap matrix O can be created from the distance matrix as shown in Equation 4.6.

$$O_{ij} = \begin{cases} 0 & : \quad D_{ij} = \infty \\ assign\_overlap(D_{ij}, max\_overlap) & : \quad otherwise \end{cases} \tag{4.6}$$

An example of the process of generating an overlap matrix can be seen in Figure 4.3.

4.3.2.2 Code Generation

The next step is to convert an overlap matrix into fixed weight tokens for each of the values. The tokens are generated such that the number of overlapping bits between two tokens is specified by the corresponding cell in the overlap matrix. This is achieved by creating a token matrix. The token matrix is $m \times l$ where m is the number of values to be encoded and l is the length of the tokens. In essence, each row of the token matrix corresponds to a token for a single value.

The overlap matrix is considered as an undirected weighted graph with m nodes corresponding to the m values and the edges corresponding to similarity between the values. The edge weights are initially equal to the desired overlap between the two values as specified by the overlap matrix. An overlap of 0 between two values has no corresponding edge between the relevant nodes.

Clique decomposition is an iterative process whereby a clique is selected and then removed from a graph until the graph is empty. A clique is a subgraph in which every node is connected to every other node.

In order to generate the token matrix, a clique is iteratively removed from its parent graph by subtracting the weight of each edge in the clique from the corresponding edge in the parent graph. If the new weight of the edge is 0 then the edge is deleted. For each

clique identified and removed from the overlap matrix, a single column is added to the token matrix with the rows corresponding to the members of the clique set to 1. This process is illustrated in Figure 4.4.

The most simple strategy for selecting cliques is to simply choose pairs of nodes with an edge between them (Palm, Schwenker, and Sommer 1994). However this results in tokens that are very long and have a large weight (Hobson 2011) and, for performance reasons, it is desirable to have short tokens with a low weight.

Since the length of the tokens is to a large extent determined by the number of cliques that have to be removed from the overlap matrix, it is desirable to remove fewer cliques of greater size. A maximal clique is a clique that cannot be enlarged by the addition of another node whilst maintaining the property of all nodes being connected to each other (Bron and Kerbosch 1973). A maximum clique is a clique that is both maximal and for which there are no other cliques of greater size, clearly there can however be be several maximum cliques of equal size (Bomze et al. 1999).

Selecting maximum cliques for removal at each iteration results in tokens that are considerably shorter and with a lower weight than selecting pairs (Hobson 2011). Table 5.12 shows pairwise tokens that are 5.5 times longer than the maximal clique tokens. However this greedy clique decomposition approach is not guaranteed to produce the optimal length tokens. Finding the optimal clique decomposition is a problem that is known to be NP–Complete (Golumbic 2004).

In general, the task of identifying a maximum clique from a graph is also NP–Complete (Karp 1972). For this reason, and because maximum clique problem occurs in many different domains (Bomze et al. 1999) there has been much research into methods to solve this problem.

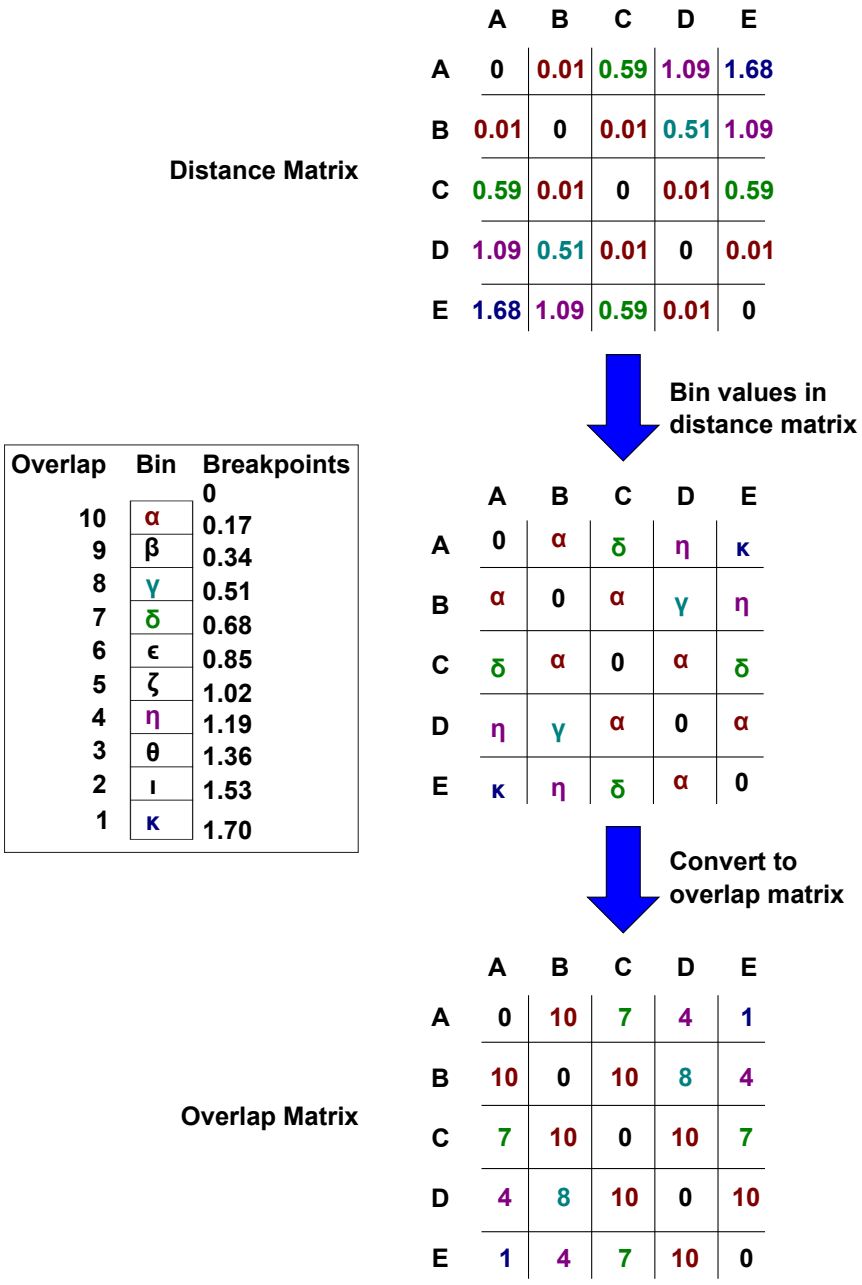


Figure 4.3: Illustration of the steps to create an overlap matrix

In this Figure, the distance matrix between the values $\{A, B, C, D, E\}$ is converted to an overlap matrix with a maximum distance of 1.70 and maximum overlap of 10. The distances are first binned, for example, the distance between A and C falls between the breakpoints 0.51 and 0.68 and is therefore assigned to bin γ. The bins are then assigned the relevant overlap value, for example, the bin γ for A, C has a overlap of 7.

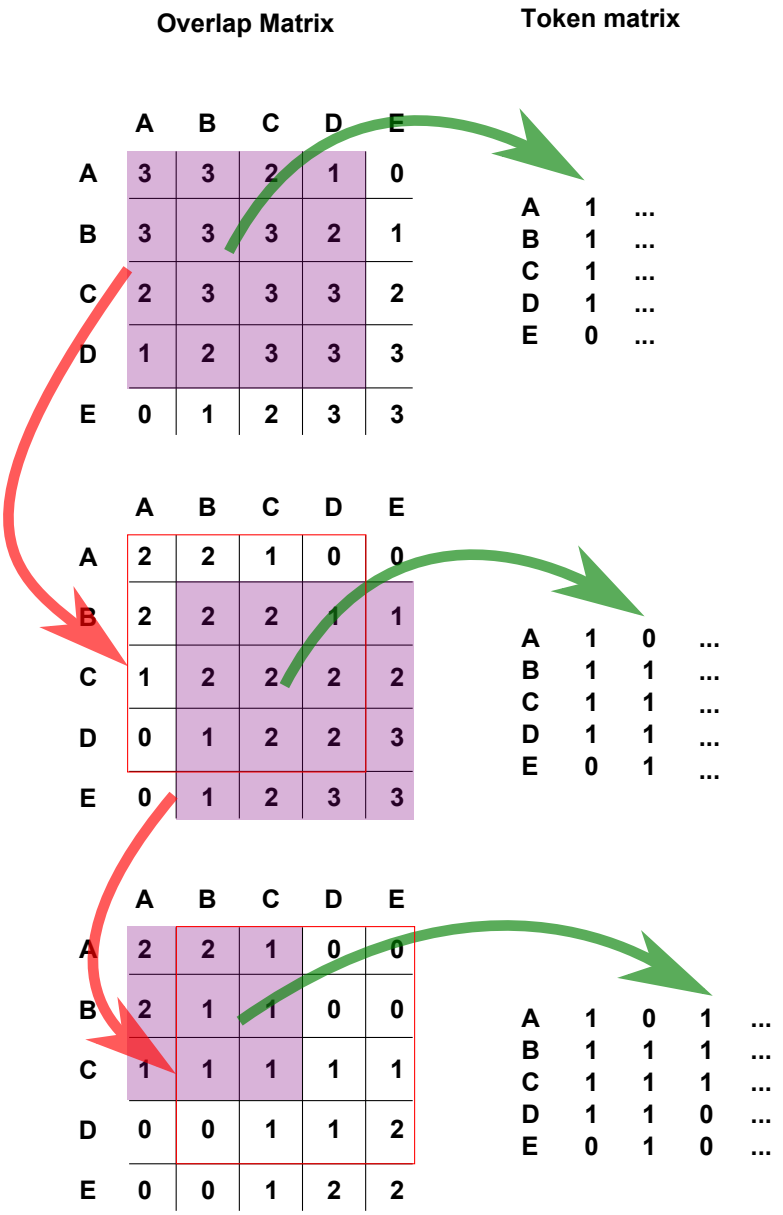


Figure 4.4: Illustration of clique decomposition used to build a Token Matrix

The purple box highlights a clique to be removed at each iteration. The green arrow shows the selected clique being added to the to Token Matrix while the red arrow shows how the clique is removed from the Overlap Matrix.

4.3.2.3 Fixed Weight

Finally it is necessary to ensure that the tokens that have been generated all have the same weight. The motivation for this property was discussed in Section 4.2. It is achieved by adding columns to the token matrix in which only a single bit is set. This increases the weight of one of the tokens without causing any additional overlap between the tokens. This process is called padding and an example is given in Figure 4.5.

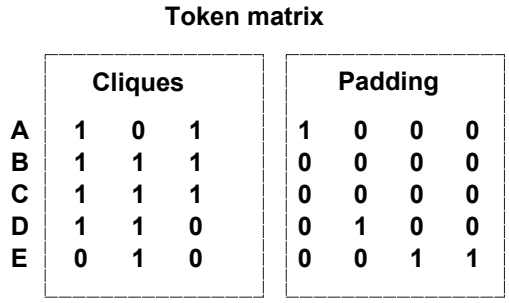


Figure 4.5: The token matrix generated from the clique decomposition has the addition of 4 padding columns, each containing a single row set to 1, in order to ensure fixed weight tokens.

Clearly padding the token can have a significant effect on the length of the tokens generated. It is therefore desirable to choose the clique decomposition strategy that minimises the amount of padding required. This is investigated further in Section 5.2.4.

4.4 Output Tokens

This Section describes the process for generating an output token that can be paired with an input token in a CMM. The output token can generally serve two purposes. The first is to identify specific properties of the sample being represented by its associated input token. For example, if the input token encoded the number plate of a car, then its paired output token could encode the colour of the car and it would then be possible to directly query the CMM for the colour of a car given the number plate.

The alternative approach is to encode a unique identifier as the output token. The CMM query will then return the unique identifier that can be used to query a traditional database for all the information about the car. The CMM query in this case allows a partial match of the number plate to determine the correct identifier for the car. It is this second approach is used for AURA Alert as described in Section 4.6 and shall be the focus of the rest of this section.

A good method for generating output tokens to represent unique identifiers will need to produce tokens that are sparse, short and have a fixed weight. In addition it is important that a large number of unique tokens can be generated for a given token size.

4.4.1 Unary Tokens

The simplest method for producing output tokens is to generate unary tokens. Unary tokens have only a single bit set to 1 in the total length of the token (Baum, Moody, and Wilczek 1988). The primary advantages of this method are that it is very quick and simple to generate the codes and that each input token will be stored in a single column of the

CMM. As a result, provided the correct input token is supplied to query the CMM there will be no error caused by interference of other data stored in the output token retrieved.

There are however significant downsides to this method. Firstly, this method requires that there be a single column in the CMM for every sample that has to be learned. This can lead to very large CMMs when large datasets need to be learned. Secondly, this method reduces the query of a CMM to what is essentially a linear scan in the binary domain, as a result it will scale poorly to larger datasets. Finally, the fault tolerance granted by the CMM approach is lost because the input tokens are not distributed across the CMM (Hobson 2011).

4.4.2 Baum Codes

Baum Codes can be used to produce tokens that have a fixed weight and a relatively small amount of overlap (Baum, Moody, and Wilczek 1988). The token is divided in to l sections where the lengths of all the sections are coprime (do not share any common factors). Each section has a single bit set to 1 and therefore the the token weight is equal to l. In order to generate the Baum Coded token t with sections $s_1, \ldots, s_l$, bit j in the token is set according to Equation 4.7.

$$bit_j^t = \begin{cases} 1 & : \quad j - \sum_{k=1}^{l-1} s_k = c \mod s_l \\ 0 & : \quad otherwise \end{cases} \tag{4.7}$$

As a result a single bit is set to 1 in each section and the position of the set bit increments with each new token t and wraps around when it reaches the end of its section. This is demonstrated in Figure 4.6.

s_1			s_2	
1	0	0	1	0
0	1	0	0	1
0	0	1	1	0
1	0	0	0	1
0	1	0	1	0
0	0	1	0	1

Figure 4.6: Example of Baum Code Generation
This Figure illustrates the process of generating Baum Codes for tokens of length 5 and weight 2. Here the sections s_1 and s_2 have length 3 and length 2 respectively. This allows a total of 6 unique tokens to be constructed.

This strategy for generating codes allows $s_1 \times s_2 \times \ldots \times s_l$ unique tokens to be created that are guaranteed to have only a small number of overlapping bits between them. This will allow more samples to be learned by a CMM of a given size than is the case with unary tokens. However it also introduces the possibility of an error occurring during recall as a result of interference between samples in the CMM. The results of this potential error are examined further in Chapter 8.

4.5 Thresholding

The output from a CMM query is not a binary token but an array of scores that represent the level to which the input token was matched across each row. In order to decode these match scores into the relevant output tokens it is necessary to apply a threshold to the output scores. Any score that is greater or equal to the threshold value is set to 1 and any score below the threshold is set to 0. In this way the binary output tokens that are retrieved can be decoded. An example of applying a threshold can been seen in Figure 4.7.

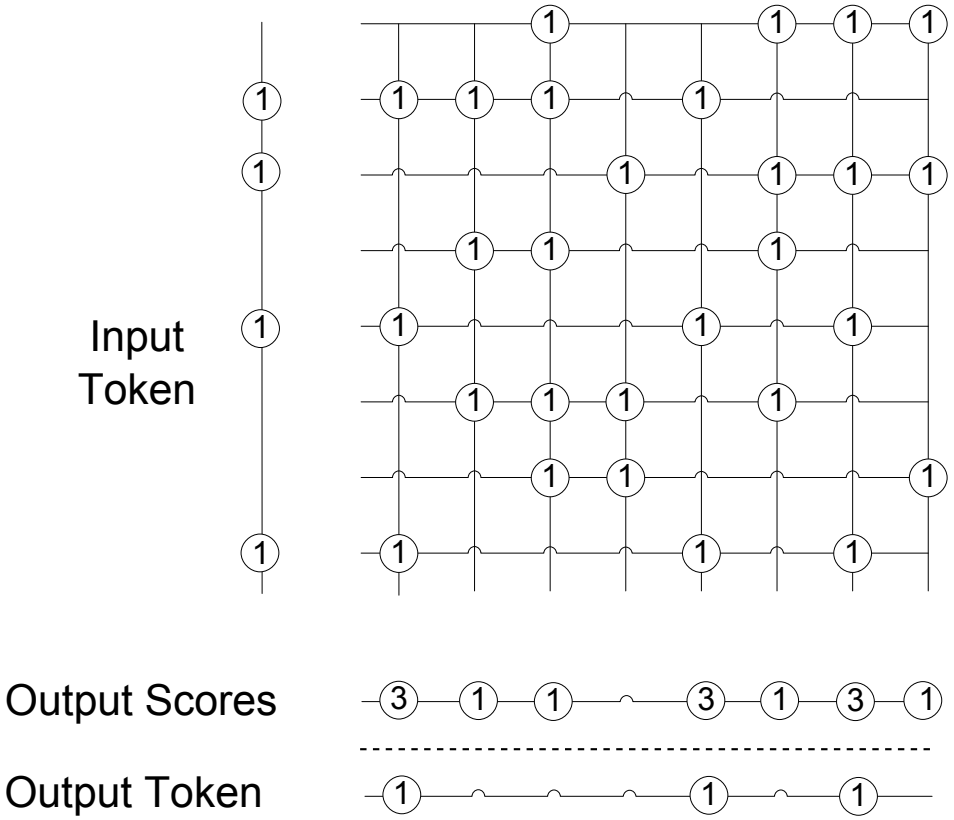


Figure 4.7: An Example CMM Query with threshold value of 3.

One threshold strategy is to set the threshold value equal to the weight of the input token. This is called Willshaw thresholding (Willshaw, Buneman, and Longuet-Higgins 1969). Willshaw thresholding is effective when the input token forms an exact match with a sample that has been learned by the CMM however it does not support generalisation to match similar but different input tokens(Austin and Stonham 1987).

Provided that the input tokens used to train the CMM have a fixed weight, it is possible to use the L–max threshold strategy. The L-max threshold (Austin 1996) is used to identify the L closest matches from the CMM output score vector. It works by iteratively lowering the threshold value applied to the output scores until at least L matches are observed. The problem with this approach is that it is considerably less effective when unary output token are not used (Hobson 2011) and can potentially yield output tokens that have not been learned by the CMM as its result.

The L–WTA strategy was introduced to overcome the problem of retrieving unlearned output tokens when using Baum Coded output tokens (Hobson 2011). With Baum Codes it is known that only 1 bit can set in each section if the code is valid. As a result the

L–WTA strategy is to set the threshold value independently in each section so that only the closest match in each section is set to 1. This ensures that a valid Baum Code is retrieved from the CMM and increases the likelihood that the retrieved token matches one learned by the CMM(Hobson 2011). However the downside of this method is that it can only return the single best match for a token.

In order to retrieve multiple matches, it is possible to use a second CMM to match the output scores so that the L–max threshold can be used on single bit outputs. This is the approach taken in the ADAM system (Austin 1987), a precursor to AURA, but the addition of a second CMM can potentially add a significant overhead to the query.

4.6 AURA Alert

AURA Alert is an anomaly detection model based on the AURA framework. It has been used successfully to monitor complex assets such as industrial gas turbines (Austin et al. 2010) and traffic flow patterns (Krishnan et al. 2010). AURA Alert learns the normal operating behaviour of a system by being trained on a dataset of system states that are recorded while the system is known to be operating normally.

There are three major components to the AURA Alert model. The first part of the model is to define the system states that need to be learned. This is achieved by using several sensors to monitor the target system. At each time point a feature vector that represents the system state can be created by combining the readings from a subset of the sensors. This allows the system state to be represented as a single point in a multi–dimensional space which can be stored within the a CMM.

The second component is an approximate Euclidean k-NN algorithm that is implemented on top of the AURA framework (Weeks et al. 2003). This is achieved with input tokens created by using equi–width binning with the Parabolic Kernel method, unary output tokens and the L–max threshold. During the training phase, the binning process causes multiple samples to be assigned to the same input token. In this case the samples are considered to belong to the same state.

During the monitoring phase, the recorded sample in converted into an input token which is then used to query the CMM. This retrieves the best matching state from the CMM. The final stage is to perform a linear scan of all the samples that are associated with the retrieved state. This is called the filtering state and is used to identify the closest system state sample. The distance between this sample and the query sample is then thresholded to identify whether the query sample represents an anomalous system state. In the case where the sample is actually anomalous it is potentially possible to evaluate the features of the sample to diagnose the cause of the fault (Austin et al. 2010).

4.7 Summary

The AURA Alert data–driven model has been introduced in this Chapter. This model makes use of a Correlation Matrix Memory (CMM), a form of binary associative neural network, to implement a distance based anomaly detection algorithm.

The CMM is trained by learning associations between pairs of binary input and output

tokens. The data storage capacity of a CMM, and therefore the speed and accuracy with which it can be queried, is dependent on the length and weight of both the input and output tokens. As a result, several methods for generating both the input tokens and output tokens have been examined.

The anomaly detection algorithm works by using the partial matching functionality of a CMM query to implement an approximate kNN algorithm. A threshold function is applied to the output scores from the CMM query to decode a set of candidates that can then be filtered by a linear scan to identify a set of neighbours. An anomaly is identified if the distance of a sample to its neighbours is sufficiently large.

Various modifications to this algorithm will be presented in the next Chapter.

Chapter 5

Modifications to AURA k–NN

5.1 Introduction

The components of the AURA k–NN algorithm at the center of AURA Alert were described in Chapter 4. The purpose of this Chapter is to present several novel modifications to the AURA k–NN algorithm. The aim of these modifications is to increase the speed of AURA k–NN while maintaining or improving its accuracy. If successful, this would allow AURA Alert to handle the expected increase in data volumes described in Section 3.2.

The modifications consist of changing the input and output tokens used to train and query the AURA CMM. The rest of the algorithm remains the same. The decision to focus on changing the input and output tokens is due to the results of preliminary experiments that highlighted the CMM query as being the largest bottleneck in the existing algorithm.

In Section 5.2 the use of Overlapped Binary Code Construction (OBCC) tokens is investigated with a view to replacing Parabolic Kernel tokens as the input tokens used in AURA k–NN. The primary motivation for this change is that OBCC tokens can support any distance metric and therefore present the opportunity to better tailor the algorithm to solve specific problems. This investigation led to the development of Weighted Overlap Code Construction (WOCC) tokens which are presented in Section 5.3. WOCC tokens are an optimised version of OBCC tokens that are specifically designed to make use of the facilities of an AURA CMM to support weighted rows in the input tokens. Finally the use of Baum Codes for the output tokens is investigated as a means of reducing the size of the CMM in order to decrease query times. This is investigated in Section 5.4.

5.2 Improved OBCC

The OBCC process for generating input tokens, described in Section 4.3.2, can be summarised in four steps. First a distance matrix is computed of the pairwise distance between the values to be encoded. This is then transformed into an overlap matrix. A token matrix is produced via a clique decomposition of the overlap matrix and finally the token matrix is padded to ensure that all the tokens have a fixed weight.

In this section, several of these steps are examined in detail with the aim of improving the process of generating OBCC tokens. Improvement is measured with respect to the time required to generate the tokens and the length and weight of the tokens in comparison to the standard OBCC process described by Hobson (2011).

5.2.1 Binning

The first step in generating OBCC tokens is to bin the signal in order to convert it into a sequence of discrete values. This is necessary because the input to and output from a binary CMM is required to be discrete and the raw signals or features usually consist of continuous data. The process of binning a signal applies a form of lossy compression whereby each value in the signal is assigned to one of a finite number of bins.

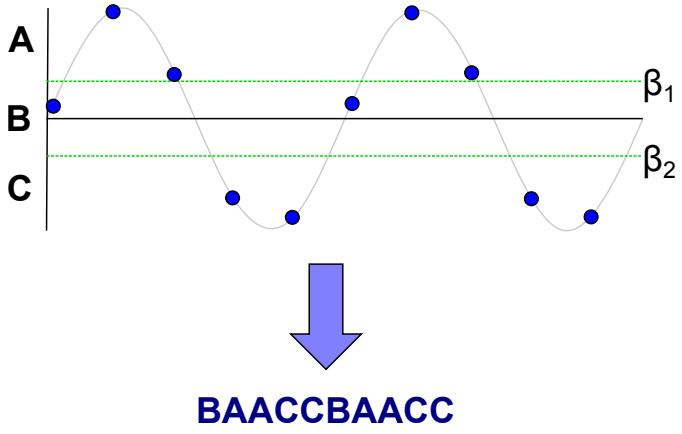


Figure 5.1: Illustration of binning process

The set of n bins $B = b_0, \ldots, b_{n-1}$ is defined via an ordered list of $n+1$ boundary values known as breakpoints $\beta_0, \ldots, \beta_n$ where β_0 and β_n correspond to $-\infty$ and ∞ respectively. The value x is assigned to bin b_i if it lies between the upper and lower breakpoints that define the bin. The binning process, where a continuous signal is mapped to its binned representation is illustrated in Figure 5.1.

The process of defining B depends on both the number of bins to be defined and where the breakpoints are positioned. The strategies for placing the breakpoints are discussed in Section 5.2.1.1 while a discussion of the number of bins to use is given in Section 5.2.1.2

While binning is necessary in order to enable a signal to be stored within a CMM, there is also a distinct advantage provided by the binning process.

Binning drastically reduces the space of values from which the signal can consist. This makes it possible to encode all of these values as tokens for the CMM. As a result, an encoding strategy can be chosen which guarantees that the tokens for all similar values will have some proportion of their bits overlapping and that tokens for dissimilar values will have no overlapping bits.

In addition, by grouping similar signal items together, binning provides a form of generalisation. This enables signals that differ by very small amounts to be easily recognised as similar to each other and reduces the sensitivity of the algorithm to noise within the signal.

Clearly this will introduce boundary effects where two values that are close to, but either side of, a breakpoint will appear to be more dissimilar than two values that fall within the same bin. This is regardless of the actual level of similarity between both pairs of values. Within the AURA Alert system these boundary effects are mitigated via the token encoding method that is selected, both PK tokens and OBCC tokens ensure that adjacent bins have many overlapping bits in common, and the linear scan of the candidate samples that are drawn from partial matches within the CMM.

5.2.1.1 Binning Strategy

Each of the bins in B is defined in terms of the upper and lower boundary values called breakpoints. The positioning of the breakpoints affects the width of each bin and therefore how many values fall within each bin. As a result, the strategy used to select the positions of the breakpoints has a large effect on the tokens that are generated.

Dougherty, Kohavi, and Sahami (1995) categorise binning strategies as being either: global or local, supervised or unsupervised and static or dynamic. Global strategies are those that are applied to all data, while local strategies apply different breakpoints to subsets of the overall data. Supervised strategies make use of labelled data to inform the location of the breakpoints while unsupervised strategies do not. Finally, static strategies determine the breakpoints for features independently via a single pass of the data in comparison to dynamic strategies that optimise the breakpoints to capture interdependencies between different features.

The generation of CMM tokens has to be performed separately to the novelty detection search due to the computational expense of generating the tokens. The reasons for this computational expense are discussed in Section 5.2.3. However as a consequence of this, only strategies that are global, static and unsupervised are appropriate for this purpose. Additionally, in order to make the tokens more generally applicable, it is desirable to require minimal prior knowledge about the data before binning takes place. Several binning strategies such as Equi–width (Dougherty, Kohavi, and Sahami 1995), Optimised Equi–width (Schmidberger and Frank 2005), Equi–frequency (Dougherty, Kohavi, and Sahami 1995), k–Means Clustering (Min 2009) and Expectation Maximisation (Dempster, Laird, and Rubin 1977) based clustering have previously been evaluated for the purpose of encoding CMM tokens (Hodge and Austin 2012). However, of these, only the Equi–width and Equi–frequency strategies conform to the requirements specified above.

These two strategies are now considered in detail in the following section.

Equi–width The breakpoints are chosen so that the width of every bin is equal across the input range. The exception is the outer bins which also include any values outside the expected input range.

The breakpoints for equi–width bins are computed such that the following condition holds:

$$width = \beta_{i+1} - \beta_i, i \notin \{0, n-1\} \tag{5.1}$$

Equi–frequency The breakpoints are chosen so that the number of values from the training data placed within each bin are distributed equally across all the bins. Typically this requires advance knowledge about the values of the signals to be encoded (Hodge and Austin 2012). However in the case of time series signals, if the signals are first normalised, it can be assumed that the values within the signal will occur with an approximately Gaussian distribution (Lin et al. 2003). As a result the breakpoints can be chosen such that the area between consecutive breakpoints under the Gaussian curve, normalised such that $\int_{-\infty}^{\infty} N = 1$, is:

$$area(\beta_i, \beta_{i+1}) = \frac{1}{n}, i \notin \{0, n-1\} \tag{5.2}$$

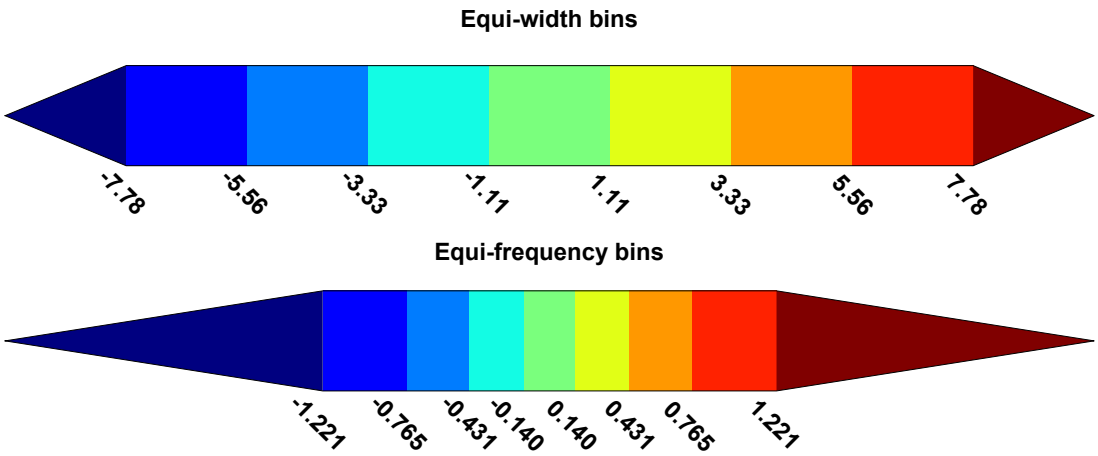


Figure 5.2: A comparison of binning strategies.

This Figure illustrates how the different binning strategies yield different breakpoints when computed to give 9 bins for data expected to be in the range $(-10, 10)$. Each bin is assigned a unique colour. With the Equi–width strategy all the bins are the same size. In contrast, the Equi–frequency strategy produces smaller bins near the middle of the range and larger bins at the extremes. The triangles at the edges represent these outer bins including values outside the expected range ($(-\infty, \beta_1)$ and $(\beta_8, +\infty)$).

Figure 5.2 shows the differences between the bins generated by these two strategies. The Equi–frequency approach yields smaller bins towards the mean of the data range. This is because it is assumed that once the signal has been normalised the values will more frequently fall near the mean than at the edges of the data range (Lin et al. 2003). A side effect of this is that this strategy provides a greater discriminatory capability between bins around the mean than is available using the Equi–width strategy. However it is at the expense of a lower discriminatory capability at the edges of the data range.

In order to determine which strategy is best for generating CMM tokens, I have performed two experiments. The first evaluates how the binning strategy effects the length and the weight of the tokens that are generated and the second investigates which binning strategy best represents the signal.

Token Properties In order to determine the effect of binning strategy on the length and weight of the tokens produced. I have performed an experiment to measure the difference between tokens generated to encode a single feature of 20 bins using both Equi–width and Equi–frequency binning. Table 5.1 provides the results with respect to the token length and Table 5.2 provides the results with respect to the token weight.

Table 5.1: The effect of binning strategy on token length

Max Distance	Max Overlap	Equi Width Length	Equi Frequency Length
0.5	5	98	90
0.5	10	104	140
0.5	15	135	204
1.0	5	82	109
1.0	10	139	104
1.0	15	143	131
1.5	5	138	105
1.5	10	106	128
1.5	15	143	135
2.0	5	116	88
2.0	10	123	132
2.0	15	133	171
2.5	5	153	84
2.5	10	119	135
2.5	15	124	140

This table shows how the length of tokens generated to encode a single feature with 20 bins is affected by the binning strategy chosen.

Table 5.2: The effect of binning strategy on token weight

Max Distance	Max Overlap	Equi Width Weight	Equi Frequency Weight
0.5	5	11	10
0.5	10	16	17
0.5	15	22	25
1.0	5	11	12
1.0	10	19	16
1.0	15	23	22
1.5	5	14	12
1.5	10	17	18
1.5	15	24	23
2.0	5	14	11
2.0	10	18	18
2.0	15	24	25
2.5	5	16	12
2.5	10	18	18
2.5	15	23	24

This table shows how the weight of tokens generated to encode a single feature with 20 bins is affected by the binning strategy chosen.

The relative difference between the length and weight of the tokens generated using both binning strategies is very small. In these experiments, the mean length of Equi–width tokens is 4% shorter while the mean weight of Equi–frequency tokens is 3.4% lower. Overall the Equi–width tokens are shortest for 8 of the overlap matrices while the Equi–frequency tokens are shortest for the remaining 7. With respect to token weight, Equi–frequency tokens have a lower weight for 7 overlap matrices, Equi–width tokens are lower for 6 overlap matrices and there are 2 ties.

In these experiments, the overlap matrices were generated with the distance metric described in Section 5.2.3.1 that is used in all the AURA kNN experiments. For the overlap matrices generated by this distance metric, the difference between tokens caused by the

choice in binning strategy is small enough to have no significant effect on the resulting CMM query times.

Signal Representation In order to determine which binning strategy best represents the signal, I have performed an experiment to measure the error introduced by binning a signal and then reconstructing the original signal from its binned representation. The reconstruction is performed by using the mid–point of each bin as the value of that bin. Once the signal has been reconstructed, the Euclidean distance between the original sample and the reconstructed sample is computed and scaled by the number of features in the signal. A smaller distance represents a more accurate representation by the binned signal.

This experiment was performed using the UCR Time Series Classification Datasets (Keogh and Kasetty 2003) (see Section 6.2.3.2) and the results of this experiment are presented in Table 5.3.

Table 5.3: Comparison of reconstruction error between binning strategies

Dataset	Equi Width	Equi Frequency
50words	0.016	0.029
Adiac	0.019	0.031
Beef	0.011	0.005
CBF	0.023	0.042
ChlorineConcentration	0.021	0.032
CinC ECG torso	0.021	0.024
Coffee	1.502	1.549
Cricket X	0.038	0.071
Cricket Y	0.027	0.037
Cricket Z	0.038	0.072
DiatomSizeReduction	0.014	0.022
ECG200	0.028	0.048
ECGFiveDays	0.064	0.075
FaceAll	0.023	0.040
FaceFour	0.014	0.024
FacesUCR	0.023	0.039
fish	0.012	0.016
Gun Point	0.023	0.016
Haptics	0.012	0.016
InlineSkate	0.006	0.010
ItalyPowerDemand	0.055	0.093
Lighting2	0.013	0.020
Lighting7	0.015	0.024
MALLAT	0.008	0.012
MedicalImages	0.048	0.064
MoteStrain	0.028	0.040
NonInvasiveFatalECG Thorax1	0.014	0.022
NonInvasiveFatalECG Thorax2	0.011	0.018
OliveOil	0.011	0.005
OSULeaf	0.013	0.020
SonyAIBORobot Surface	0.031	0.060
SonyAIBORobot SurfaceII	0.033	0.056
StarLightCurves	0.008	0.013
SwedishLeaf	0.023	0.037
Symbols	0.013	0.019
synthetic control	0.033	0.035
Trace	0.014	0.011
Two Patterns	0.022	0.021
TwoLeadECG	0.028	0.033
uWaveGestureLibrary X	0.014	0.020
uWaveGestureLibrary Y	0.015	0.026
uWaveGestureLibrary Z	0.014	0.019
wafer	0.021	0.034
WordsSynonyms	0.020	0.032
yoga	0.013	0.021

This table shows how accurately a signal can be reconstructed after binning using both Equi–Width and Equi–Frequency binning. The signal is reconstructed as s' using the mid–point of each bin and the mean feature distance, computed as $Euclidean(s, s')/|s|$, is provided for both binning strategies.

It is clear that Equi–width binning results in a smaller reconstruction error for the majority of these datasets. The mean scaled reconstruction error across all datasets is 23% lower when using the Equi–width binning strategy. However Equi–width is not universally more accurate than Equi–frequency. For example the Beef dataset has a reconstruction error that is 2.2 times lower with Equi–frequency binning. This aligns with the results of (Hodge and Austin 2012) that the best binning strategy will be dataset dependant. How-

ever overall it appears that Equi–width binning is the best strategy to use for representing the signals in the UCR Time Series Classification Datasets.

5.2.1.2 Number of Bins

Increasing the number of bins increases the granularity of the encoding. Table 5.4 clearly illustrates that as the number of bins increases, the length and the weight of the generated tokens increases significantly. Using the equi–width binning strategy, the tokens for 100 bins are 83 times longer and have 4.3 times as many bits set than the 10 bin tokens. With the equi–frequency binning strategy this increase is less being only 30 times longer and having 2.4 times more bits set. Hodge and Austin 2012 show that the optimal number of bins to use for best accuracy is dataset dependent. However it must also be considered that increasing the number of bins can have a significant effect on the size of the CMM due to the increased token length and this will result in a reduction in query times.

Table 5.4: The effect of the number of bins on the properties of generated tokens.

Bin Strategy	Bins	Token Length	Token Weight
Equi Width	10	35	13
Equi Width	20	139	19
Equi Width	30	227	21
Equi Width	40	454	26
Equi Width	50	605	29
Equi Width	60	987	35
Equi Width	70	1,184	37
Equi Width	80	1,414	39
Equi Width	90	2,122	47
Equi Width	100	2,929	56
Equi Frequency	10	55	15
Equi Frequency	20	104	16
Equi Frequency	30	214	19
Equi Frequency	40	429	24
Equi Frequency	50	577	25
Equi Frequency	60	596	25
Equi Frequency	70	636	25
Equi Frequency	80	1,268	33
Equi Frequency	90	1,389	34
Equi Frequency	100	1,651	36

This table shows how the length and weight of the generated tokens are affected by increasing the number of bins used to represent each feature.

5.2.1.3 Summary

In this Section the Equi–width and Equi–frequency binning strategies have been introduced and evaluated with the view to determining whether either strategy results in CMM tokens with a consistently lower length or weight.

When encoding a feature of 20 bins there is little difference between token generated with either strategy. However when increasing the number of bins it appears that the Equi–frequency strategy scales better. Despite this, increasing the number of bins has a large effect on the length and weight of the tokens. A 10 fold increase in bins resulted in tokens that were 83 times longer and requires 4.3 times more bits. As a result, the number

of bins must be minimised where possible in order to achieve good performance from the CMM.

For the majority of datasets examined, the Equi–width strategy provided a better representation of the signal. However this advantage is dataset dependent with some datasets showing Equi–frequency to perform significantly better.

The existing AURA kNN implementation using Parabolic Kernel input tokens uses Equi–width binning and the results of these experiments provide no compelling evidence to advocate not also using the Equi–width strategy for OBCC tokens.

5.2.2 Multiple Features

It is often desirable to generate tokens that can represent multiple features. For example, consider the representation of geographic coordinates. Each location requires the specification of a value for both its latitude and longitude. As a result each specific location is represented by 2 features. In this Section two methods are detailed for using OBCC to represent values consisting of multiple features as tokens. The concatenation approach is discussed in Section 5.2.2.1 and the n–tuple approach is discussed in Section 5.2.2.2. A comparison between these two approaches is given in Section 5.2.2.3.

5.2.2.1 Concatenation

The concatenation approach for creating multi–feature tokens is very straightforward. Tokens are first generated individually for each of the features. These tokens are then concatenated in order to produce a longer token that represents all the features. The simplicity of this approach makes it extremely fast to implement. As a result, this approach has previously been used in various applications (Austin et al. 2010; Hodge and Austin 2005; Krishnan et al. 2010).

5.2.2.2 n–Tuple

The alternative approach is to consider the combination of multiple values as representing a state for which a token should be generated. This combination of values is called an n–tuple where n specifies the number of features that are being combined.

The original binary n–tuple method proposed by Bledsoe and Browning (1959) essentially uses a lookup table to map a specific n–tuple to a particular state. Each state is then assigned a unique token that is used to represent the n–tuple value. In order to increase the generalisation ability within the features of the n–tuple this method was extended to support features that had been binned to a predetermined number of discrete values and is referred to as the Grey Scale n–tuple method(Austin 1988). In this approach, each grey scale n–tuple is sorted and each ordering of the elements is assigned to a state and represented by a unique token.

While the grey scale n–tuple method improves the generalisation between features, it does not maintain any generalisation between the states and the tokens that represent them. For example, consider two 4–tuples $t^a = (2, 3, 6, 1)$ and $t^b = (2, 3, 4, 1)$, these tuples could be considered relatively similar with only a difference of 2 in the third element t_2. This similarity is expressed using the grey scale n–tuple method by t^a and t^b being

assigned the same token to represent them. This is because the ordering is the same when the elements of both tuples are sorted (t_0, t_1, t_3, t_2). However the tuple $t^c = (4, 3, 6, 1)$ also only has a difference of 2 from t^a, this time in the first element t_0. However because this difference causes a change in the relative ordering of the elements in the tuple, t^a and t^c are assigned to different states and therefore given different tokens. The tokens assigned to each state do not reflect any similarity that may be present between the different states.

Using OBCC it is possible to generate tokens where the Hamming distance between an arbitrary pair of tokens reflects the similarity between the states that they represent. This is achieved by assigning a unique state to each n–tuple value. An overlap matrix can then be generated as described in Section 4.3.2.1 using a pairwise similarity metric between the unique states as the basis for determining the desired overlap between the states. The remaining steps in the process for generating OBCC tokens are as normal. However as the number of features increases, the number of these unique states that need to be encoded increases exponentially. As a result this approach can quickly become infeasible with more than 4 features. This is examined further in the following Section.

5.2.2.3 Comparison

Consider the distances given by comparing a value $v \in V$ with each of the values $v' \in V$. An ideal mapping from the value v to a token t ensures that the ordering of values v' as sorted by the chosen distance function from v would be preserved if instead sorted by the Hamming distance between t and t'.

In order to demonstrate that the n–tuple encoding of tokens is better at preserving the relationship between the ordering of $sorted(distance(v, v'))$ and $sorted(Hamming(t, t'))$, a two dimensional feature space was generated with a range $(-1, 1)$ along each axis. This feature space was overlaid with a regular grid consisting of 100 points. Tokens were generated for each of these 2–dimensional data points using both: the concatenate method where OBCC was used to encode the values for each dimension individually before joining them together; and the n–tuple method where OBCC is used to encode the pair of values directly. These data points were then sorted according to their distance from the mid–point at $(0, 0)$. Next the corresponding tokens were sorted according to their Hamming distance from the token representing the mid–point. Finally the two token orderings were compared to the correct ordering to determine which was closest.

Determining the closer ordering is performed using two measures of disorder (Estivill-Castro and Wood 1992). The first measure of disorder is a count of the number of exchanges required to correctly order the tokens. An exchange is defined as the positions of two tokens being swapped. This gives a measure of how many of the tokens are incorrectly positioned. The second measure of disorder is the maximum exchange distance. The exchange distance is the number of positions that a token moves during an exchange. This gives a measure of how far out of position a token is.

The results of this experiment can be seen in Table 5.6. It is clear that the n–tuple approach is superior by both measures, with the concatenate method requiring a mean of 2.1 times as many exchanges and a maximum exchange distance that is 3.4 times larger in order to correctly order the tokens.

Table 5.6: Comparing token ordering for concatenation and n–tuple multi–feature tokens.

Method	Max Overlap	Exchange Count	Max Exchange Distance
Concatenate	5	69	58
Tuple	5	29	17
Concatenate	10	79	58
Tuple	10	42	17
Concatenate	15	83	57
Tuple	15	41	17

This Table compares the disorder observed when multi–feature tokens generated using either the concatenation method or the tuple method are sorted by the number of overlapping bits with the token representing the mid–point of the data space. The correct ordering, as determined by the Euclidean distance, would yield an Exchange Count of 0 and a Maximum Exchange Distance of 0. The data space consists of 2 features in the range $(-1, 1)$ and each feature is divided into 21 bins. Tokens were generated for 100 points placed in a regular grid across the data space using a maximum distance of 1.1.

The greater disorder observed with the concatenate method is due to greater weight being placed on an exact match with a single feature than close matches across several features. This bias is illustrated in Figure 5.3 where an increased overlap can be seen along the horizontal and vertical axis that marks an exact match with the target mid–point value while using the concatenate method.

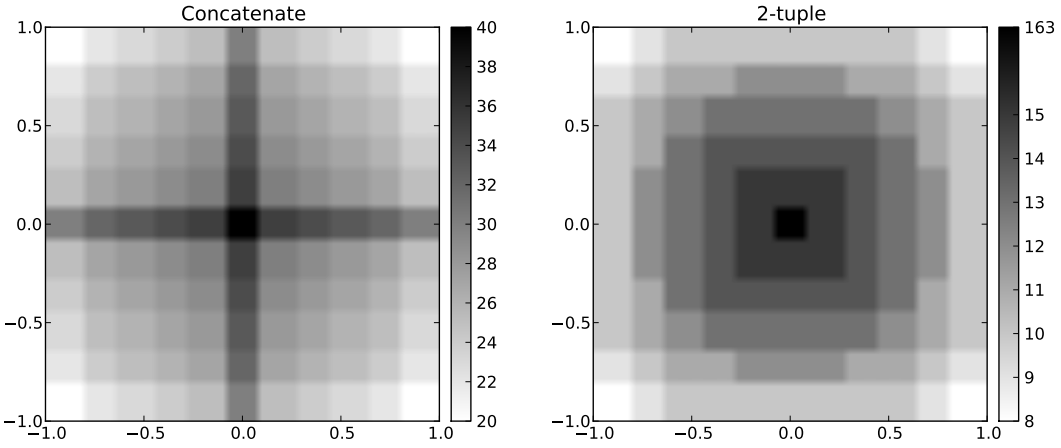


Figure 5.3: Comparison of token overlap between the Concatenate method and the n–Tuple

This Figure shows the amount of overlap between tokens representing the 2–feature values from $(-1, -1)$ to $(1, 1)$ and the token representing the mid point value of $(0, 0)$. The left plot shows tokens created using the concatenate method to combine features and the right plot shows tokens created using the n-tuple method. Both sets of tokens were created to encode Euclidean distance with a maximum overlap of 15, a maximum distance of 2.0 and 11 equi–width bins per feature.

This illustrates the primary advantage of the n–tuple method. It is better able to accurately encode arbitrary distance functions that require a combination of multiple features. In contrast, the concatenation method simply provides a sum of the distances along each

axis. A concrete example is that the n–tuple method can approximate the Euclidean distance for multi–feature values while the concatenation method would more closely resemble the Manhattan distance between the values.

Another feature of the n–tuple method is that it can accurately encode arbitrary distance functions and therefore tokens can be generated to represent classes of values that are not linearly separable. This is simply not possible with the concatenation approach. For example, consider the problem presented in Figure 5.4. This is an example of the XOR problem (Austin 1993). The objective is to generate tokens that represent the similarity between points in the feature space. Points that have low pairwise Euclidean distance are considered similar, with the caveat that any pairs of points in different classes are considered to have no similarity. This is represented in the tokens by having some bits overlapping between tokens for the points in the same classes and no overlapping bits if the points are in different classes.

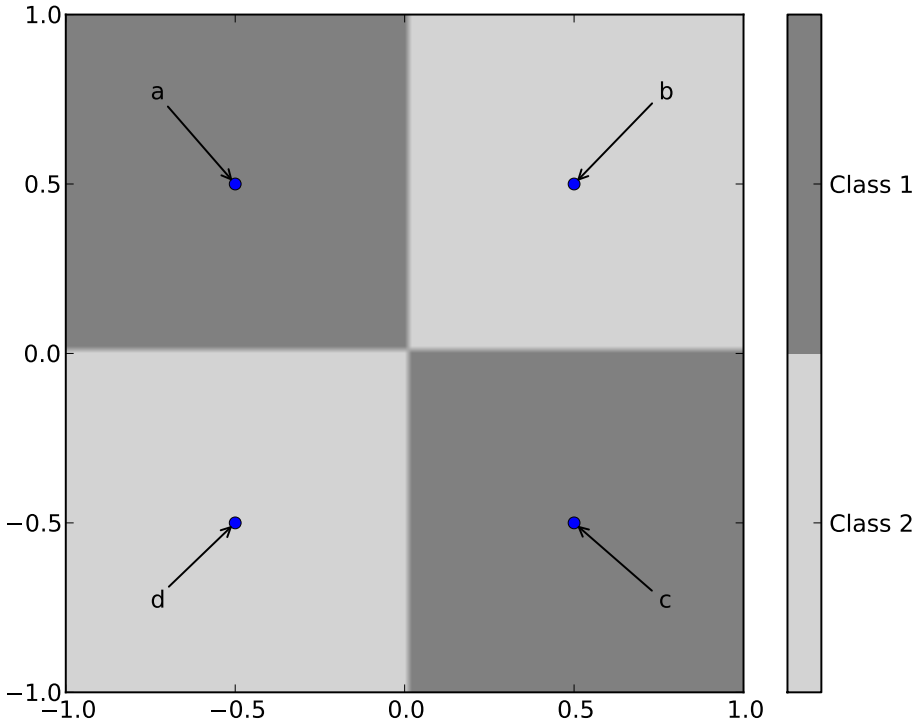


Figure 5.4: Example of a non–linearly separable problem

This Figure shows an example of the XOR problem. The problem is to encode tokens for points in both classes with no overlapping bits between points in different classes. The points A, C are in Class 1 and B, D are in Class 2.

Using the problem presented in Figure 5.4, tokens were generated using OBCC for both the concatenation method and the n–tuple method. For simplicity a total of 4 bins per feature and a maximum overlap of 3 bits were chosen. The tokens generated using the n-tuple method for the points $\{A, B, C, D\}$ are given in Table 5.7 while the tokens generated using the concatenation method can be found in Table 5.8.

Table 5.7: n–tuple tokens for problem given in Figure 5.4.

Point	Class	Token
a	1	0101100000000000010000000000011000000000
b	2	1000010100000001000001000000000000000001
c	1	0101001000000010000100000000000000000100
d	2	1010010010001000001000000000000000000000

This Table shows the tokens generated for points $\{A, B, C, D\}$ in Figure 5.4 using the n–tuple method for encoding multiple features.

Table 5.8: Concatenate tokens for problem given in Figure 5.4.

Point	Class	Token
a	1	111100110110
b	2	110110110110
c	1	110110111100
d	2	111100111100

This Table shows the tokens generated for points $\{A, B, C, D\}$ in Figure 5.4 using the concatenate method for encoding multiple features.

Evaluating the overlapping bits between tokens shows that the concatenation method yields tokens with more overlapping bits between tokens from different classes than between tokens within classes. In contrast, the n–tuple method tokens behave as desired and do not have any overlapping bits between different classes. These results can be seen in Figure 5.9.

Table 5.9: Comparing token overlap between non–linearly separable classes

n–tuple

	a	b	c	d
a	6	0	2	0
b	0	6	0	2
c	2	0	6	0
d	0	2	0	6

Concatenate

	a	b	c	d
a	8	7	6	7
b	7	8	7	6
c	6	7	8	7
d	7	6	7	8

These Tables show the pairwise Hamming distance between the tokens representing the points $\{A, B, C, D\}$ from Figure 5.4 that are generated using both the n–tuple and concatenate methods. The n–tuple tokens are provided in Table 5.7 and the concatenate tokens are provided in Table 5.8. The shaded cells highlight a comparison between points assigned to different classes. It is desired that the shaded cells contain a lower overlap than the non–shaded cells. Clearly this is not observed with tokens generated using the concatenate method.

However despite the theoretical advantages to using the n–tuple method, there are several issues that make it less practical to use than the concatenate method. The first issue is that of scaling. The total number of discrete values that need to be encoded is given by Equation 5.3.

$$n\_values = n\_bins^{tuple\_length} \tag{5.3}$$

Since the number of values to encode tokens for increases exponentially with the number of features to be encoded, it is clear that this approach does not scale to a large number of features. Without strategies to remove a significant number values that have to be encoded, preliminary experiments show that it becomes infeasible to encode more than 4 features using this method. This is due to a combination of: the computation time and memory required to perform the encoding; and the length and weight of the tokens that are produced.

Another issue is with the number of similarity levels represented in the different methods. With the n–tuple method, the total number of discrete distances that can be represented by the token overlap is determined by the maximum overlap parameter. In contrast, with the concatenation method, the maximum overlap parameter only applies to a single feature. As a result after the features are concatenated, combining the overlaps of multiple features allows a greater granularity in representing the similarity between values. This can be seen in Figure 5.3 where the concatenate tokens show more variety and a greater range of overlap values. Clearly this can be accounted for in the n–tuple method by increasing the maximum overlap, however as shown in Section 5.2.3.3 the effect of doing this is to significantly increase the length and weight of the tokens.

The final issue with the n–tuple method is that it produces tokens that are significantly longer and with a higher weight than using the concatenate method. This can be seen in Table 5.9 where the n–tuple method produces tokens that are orders of magnitude larger for both length and weight despite all other parameters being equal.

Table 5.9: Comparing token properties for concatenation and n–tuple multi–feature tokens.

Method	Max Overlap	Weight	Length
Concatenate	5	26	254
Tuple	5	169	39,066
Concatenate	10	38	292
Tuple	10	248	57,353
Concatenate	15	50	344
Tuple	15	346	80,984

This Table compares the length and weight of multi–feature tokens generated using either the concatenation method or the n–tuple method. The tokens encode the Euclidean distance of two features of 11 equi–width bins, a range of $(-1, 1)$ for each feature and a maximum distance of 1.1.

5.2.2.4 Summary

In this Section, two methods have been presented for generating tokens that represent multi–featured values. The concatenation method involves simply appending the tokens representing individual features and the n–tuple method involves directly encoding the distance between the multiple features using OBCC.

The n–tuple method has several theoretical advantages over the concatenation method. Specifically it is able to encode non–linear multi–feature distance functions and will provide a more accurate mapping to the number of overlapping bits between tokens from the true distance between the values.

However from a practical standpoint, the n–tuple method scales very poorly with regards to both the computational resources required to generate the tokens and the length and weight of the tokens that are generated. This limits the applicability of the n–tuple method to real world problems.

5.2.3 Overlap Matrix Generation

The process of generating overlap matrices requires the selection of three variables: the distance metric, the maximum distance and the maximum overlap. In this Section, the choices associated with these variables are examined with respect to their effect on the process of generating OBCC tokens.

5.2.3.1 Distance Metric

The most common general purpose *distance* function for comparing the similarity between two values is the Euclidean distance between them (Lin et al. 2003).

As mentioned in Section 3.11.1, the Euclidean distance between a pair of tuples, q and s, is given by:

$$euclidean(q,s) = \sqrt{\sum_{i=1}^{n}(q_i - s_i)^2} \tag{5.4}$$

This is a metric distance function since it satisfies all four conditions stated in Section 3.11.1. Since the tuples consist of binned values, the Euclidean distance between binned values can be computed as follows:

$$binned\_euclidean(q_{binned}, s_{binned}) = \sqrt{\sum_{i=1}^{n} bin\_distance(q_i, s_i)^2} \tag{5.5}$$

where:

$$bin\_distance(i,j) = \begin{cases} 0 & : \quad i = j \\ abs(bin_i^{upper\_breakpoint} - bin_j^{lower\_breakpoint}) & : \quad i < j \\ abs(bin_i^{lower\_breakpoint} - bin_j^{upper\_breakpoint}) & : \quad i > j \end{cases}$$

Figure 5.5 provides an illustration of how the $bin\_distance$ is calculated. Using this method it is guaranteed that $euclidan(q,s) \geq binned\_euclidean(q_{binned}, s_{binned})$ (Lin et al. 2003).

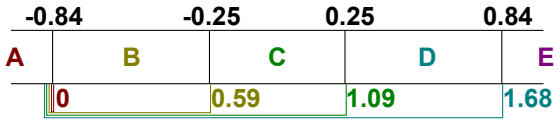


Figure 5.5: Illustration distance calculations between individual bins

This Figure demonstrates how the *binned_distance* between a value in bin A and values in bins B, C, D and E are calculated.

The *binned_euclidean* distance of two binned tuples provides a lower bound on the Euclidean distance between the original continuous tuples. Using a lower bound on the Euclidean distance ensures that any tokens retrieved from a similarity search will not be underestimating the similarity between the target and retrieved tokens and therefore samples will not be overlooked as a result of inaccuracies in the approximation of the Euclidean distance between binned samples. In addition, this property enables the computation of the distance matrix to be pruned for a significant speed up. The method for pruning the computation of distance matrix is described in the next Section.

5.2.3.2 Maximum Distance

The maximum distance parameter specifies the largest pairwise distance allowable between two samples for those samples to be considered in some way similar. Recall that the distance matrix is computed as:

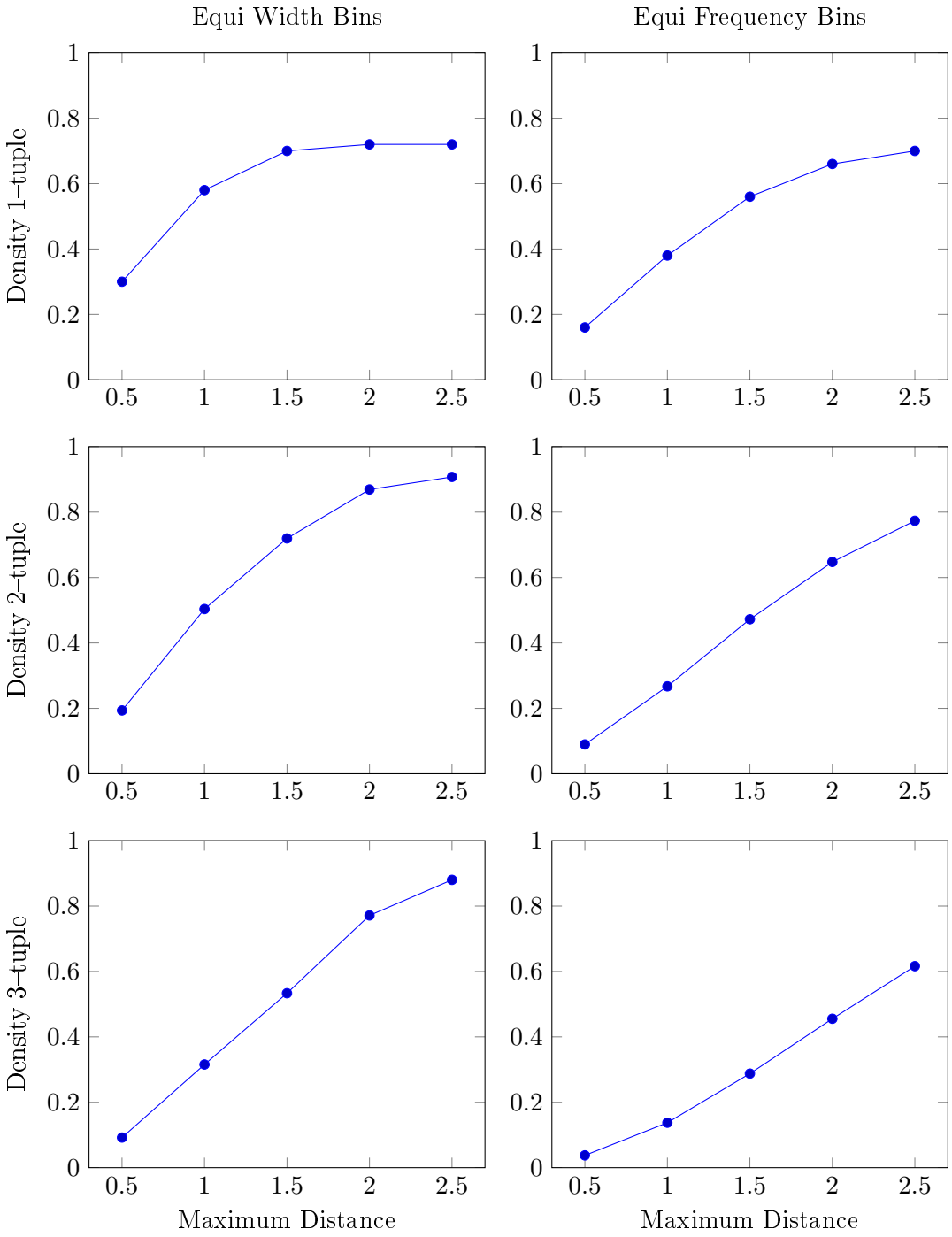
$$D_{ij} = \begin{cases} distance(i,j) & : \quad distance(i,j) < max\_distance \\ 0 & : \quad otherwise \end{cases} \tag{5.6}$$

As a result, depending on the value of the maximum distance, the distance matrix can be very sparse. This means that the amount of memory required to encode a large number of different values can be managed via the maximum distance parameter. This can be seen in Figure 5.6 where a smaller maximum distance reduces the density of the distance matrix which in turn reflects a reduction in the number of distances that have to be stored.

The remaining issue is that it is still very computationally expensive to compute the distance matrix for a large number of values. However provided that the values can be ordered and the distance function satisfies the triangle inequality (See Equation 3.4), it is possible to enumerate only the similar values and prevent the computation of distances that are known to exceed the max distance. This pruning of distant values will not effect the computation time of the distance matrix in the worst case, however on average it can result in a significant speed up. This can be seen in Table 5.10 where the use of pruning results in an 82% reduction in the execution time.

The value chosen as the maximum distance is essentially constrained by the computation resources available to compute the overlap matrix and convert the overlap matrix into binary tokens. However this choice is also problem dependent because the maximum distance is ideally chosen to be as small as possible while still representing the required level of similarity between values.

Figure 5.6: Density of Overlap Matrices as the Maximum Distance Varies



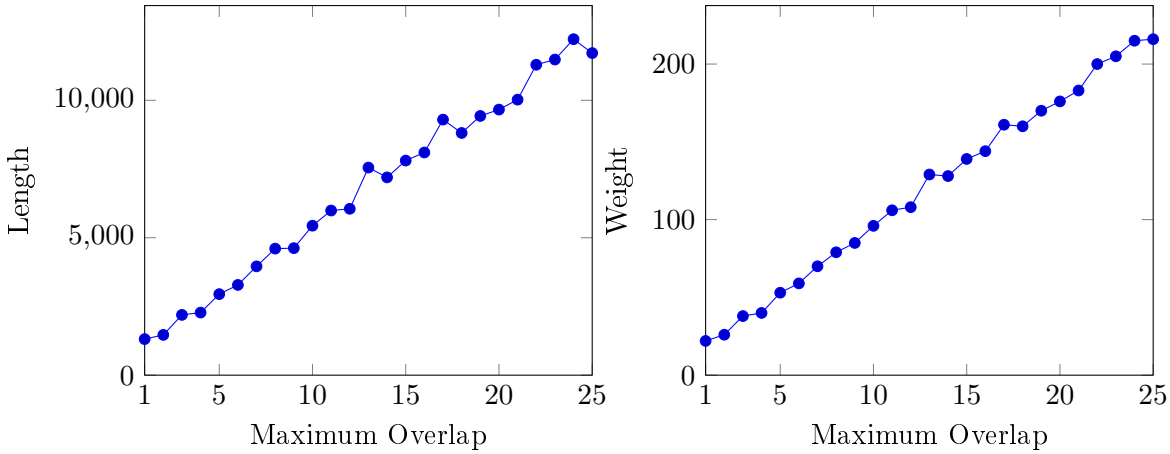
This Figure shows how the density of the overlap matrices increases as the Maximum Distance parameter is decreased. The x–axis is the Maximum Distance and the y–axis is the density of the overlap matrix where a density of 1 represents a fully dense matrix in which all nodes are connected to each other. It is desirable to have a sparse matrix which is indicated by a low density value. The left column is for overlap matrices created using equi–width binning and the right column is for overlap matrices using equi–frequency binning. The top row is for 1–tuple overlap matrices, the middle row for 2–tuple overlap matrices and the bottom row for 3–tuple overlap matrices. In all cases the same pattern is observed.

Table 5.10: Runtime comparison of computing a Distance Matrix with and without pruning.

Distance Matrix	Execution Time (s)
Without Pruning	27.90
With Pruning	4.89

This table compares the execution time required to compute identical distance matrices with and without pruning the pairwise Euclidean distances to be calculated. The distance matrices consist of a total 1000 3–tuple, 10 bin values. The range of the bins is (-1, 1) with a maximum distance of 0.5.

Figure 5.7: The effect of Maximum Overlap on the Length and Weight of tokens from a 2–tuple Overlap Matrix

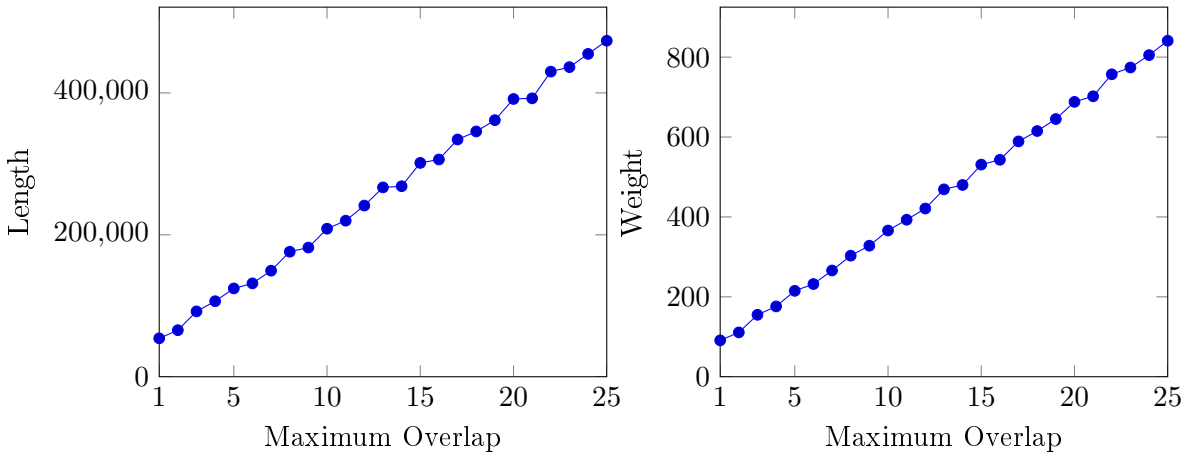


This Figure shows how varying the maximum overlap of an overlap matrix is reflected in the length and weight of the tokens that are generated from it. The tokens were generated to encode 2–tuple values. Each feature has a range of $(-1, 1)$ and 10 equi–frequency bins with a maximum distance of 1.0. The Min algorithm (Section 5.2.4.1) was used in converting the overlap matrix.

5.2.3.3 Maximum Overlap

The maximum overlap parameter defines the number of discrete levels of similarity in the range of 0 to maximum distance that can be encoded by the resulting OBCC tokens. As a result, increasing the maximum overlap allows greater discrimination between the pairwise distances that are represented in the tokens. However this also has an effect on the length and the weight of those tokens. As can be seen from Figures 5.7 and 5.8, the length and weight of tokens appear to increase linearly as the maximum overlap increases. It can also be seen that despite a linear relationship with the maximum overlap, the length and weight of the tokens soon become very large. As a result, the maximum overlap should be chosen to be as small as possible while still allowing the necessary discrimination between pairwise distances. As with the maximum distance parameter, the trade off between token size and the level of discrimination required is problem dependent.

Figure 5.8: The effect of Maximum Overlap on the Length and Weight of tokens from a 3–tuple Overlap Matrix



This Figure shows how varying the maximum overlap of an overlap matrix is reflected in the length and weight of the tokens that are generated from it. The tokens were generated to encode 3–tuple values. Each feature has a range of $(-1, 1)$ and 10 equi–frequency bins with a maximum distance of 1.0. The Min algorithm (Section 5.2.4.1) was used in converting the overlap matrix.

5.2.3.4 Summary

In this Section the parameters that affect the process for creating an overlap matrix have been discussed. The *binned_distance* metric has been chosen because it provides a lower bounded approximation of the Euclidean distance between the binned samples.

The maximum distance parameter serves a dual purpose, primarily it is used to specify the maximum allowable distance between two samples for them to be considered similar. However it also serves as a method of controlling the amount of memory required to generate the OBCC tokens by increasing the sparsity of the distance matrix and when combined with the *binnned_distance* function can be used to prune the computation of large portions of the distance matrix.

Finally the maximum overlap parameter specifies the number of specific similarity levels that are to be encoded by the OBCC tokens. Increasing the maximum overlap increases the amount of discrimination between values that can be encoded, however this comes at the cost of increasing the length and weight of the resulting tokens. Both the length and the weight of the tokens appear to increase linearly with an increase in the maximum overlap.

5.2.4 Token Generation

Once the overlap matrix has been generated, the next step is to perform a clique decomposition of the graph representing the overlap matrix. This clique decomposition is used to transform the overlap matrix into a token matrix. In this Section, several methods for selecting cliques during the decomposition are examined with respect to both the runtime of the decomposition and its effect on the length and weight of the tokens that are produced.

5.2.4.1 Clique Selection Strategies

The standard algorithm for identifying the maximum clique in a graph is the Bron–Kerbosch algorithm (Bron and Kerbosch 1973). This is a recursive backtracking algorithm that enumerates all the maximal cliques in a graph. These maximal cliques can then be ordered by size to identify the maximum clique. Pruning optimisations introduced by Tomita, Tanaka, and Takahashi (2006) have reduced the computational complexity of this algorithm to $O(3^{(m/3)})$. This complexity cannot be improved because in the general case there are a maximum of $O(3^{(m/3)})$ maximal cliques in a graph that all have to be enumerated (Moon and Moser 1965).

However for greedy clique decomposition it is only necessary to identify a single maximum clique at each iteration. Therefore if an upper bound on the size of the maximum clique is known, the Bron–Kerbosch algorithm can be terminated early once a maximal clique of the desired size is found. While not affecting the worst case runtime of the algorithm, in the average case this can provide a significant speed up.

With each iteration of the decomposition, edges are only ever removed from the graph. As a result the size of the clique selected in the current iteration forms an upper bound on the maximum clique found in the next iteration. The added overhead of tracking the size of the enumerated cliques can cause a small performance loss in some cases with small overlap matrices. However, as show in Table 5.11, in most cases the use of an upper bound to detect when a maximum clique has been found leads to a significant improvement in the clique decomposition runtime.

Table 5.11: Runtime Comparison of Exact Greedy Clique Decomposition with and without Upper Bound.

Overlap Matrix Size	Bin Strategy	No Upper Bound Mean Runtime (s)	Upper Bound Mean Runtime (s)
10x10	equi-width	0.0360	0.0358
10x10	equi-frequency	0.0351	0.0339
100x100	equi-width	7.1000	4.5600
100x100	equi-frequency	5.7800	3.6900

This table compares the runtime of the exact greedy clique decomposition of 4 relatively small overlap matrices both with and without the use of an upper bound when searching for maximum cliques. The overlap matrices were generated to approximate the Euclidean distance between binned values.

Bomze et al. (1999) presents several strategies for selecting large maximal cliques from a graph. These strategies mostly fall under the categories of sequential greedy strategies and search based strategies. The sequential greedy strategies consist of repeatedly adding nodes to a clique based on a score assigned to each node. The search based strategies consist of examining the neighbours, within the search space, of a clique to identify potentially larger cliques. As a alternative to the Bron–Kerbosch algorithm I have investigated how using a heuristic clique selection strategy affects the tokens that are produced. One example of each strategy is described below.

A very fast, simple, heuristic for selecting large maximal cliques is the MIN algorithm

(Harant, Ryjacek, and Schiermeyer 2002). This is an example of a sequential greedy heuristic algorithm. The nodes in the graph are sorted according to the degree of the node, that is, the number of edges that are connected to the node. A clique is initially formed from a single node by selecting the node with the highest degree. A tie is broken via random selection. The clique is then enlarged by the selection of the next highest degree node that is connected to all members of the clique. This process is repeated until the clique cannot be enlarged further.

An alternative approach is the Penalty–Evaporation (PE) heuristic (St-Louis, Gendron, and Ferland 2004). This is an example of a search based heuristic based on the Tabu search strategy (Glover 1989). The PE algorithm works by repeating a two step process. First a node is selected and added to a working set of nodes. Next, any nodes in the set that are not adjacent to the newly added node are removed. This forces the working set to be a clique at the end of each iteration. If the clique is larger than any previously seen clique, it is remembered as the best observed clique. The search terminates after a set number of iterations have been reached without improving upon the size of the best observed clique.

Clearly the process of selecting which node to add at each iteration is critical to the success of this algorithm. In order to select a node a score is calculated based on how many edges are shared with nodes in the current clique and the number of iterations since the node was previously included in the current clique (St-Louis, Gendron, and Ferland 2004), the node with the highest score at each iteration is chosen.

5.2.4.2 Evaluation

As previously stated in Section 4.2, the ideal token would be short and have a small weight.

Another desirable property is that the tokens can be generated in a reasonable amount of time. It is not feasible to generate optimal tokens via optimal clique decomposition due to the computational complexity of the problem (Golumbic 2004). Experiments to compute optimal tokens for toy sized overlap matrices showed that the greedy decomposition strategy produced tokens of equal weight and length to the optimal decomposition strategy, see Table 5.12. However a lack of computation resources prevents an analysis of the size and type of overlap matrix that causes the optimal and greedy strategies to diverge, even with a small overlap matrix the runtime of the optimal decomposition is 4 orders of magnitude higher than the slowest greedy decomposition. The pairwise strategy, while being extremely fast to execute, produces tokens that are 5.5 times longer and require 3.4 times more bits to be set than either the optimal or greedy clique decomposition strategies. For these reasons, I restrict my analysis to the greedy clique decomposition strategy in the rest of this Section.

Three algorithms for identifying large maximal cliques, ideally maximum cliques, to remove at each iteration have been described in this Section. These are: the exact method derived from the clique enumeration algorithm of Bron–Kerbosch (EXACT), the greedy sequential algorithm MIN (MIN) and the Tabu search based Penalty Evaporation algorithm (PE).

In order to evaluate these three algorithms I have considered the length, weight, and to a lesser extend the padding required of the tokens produced from each algorithm. In addition I have also considered the runtime of the clique decomposition using each algorithm since

Table 5.12: Properties of tokens generated via Optimal, Greedy and Pairwise clique decomposition.

Clique Startegy	Token Length	Token Weight	Runtime
optimal	34	9	72.86s
pairwise	187	31	<0.01s
greedy (exact)	34	9	0.02s
greedy (penalty_evaporation)	34	9	0.03s
greedy (min)	43	10	0.02s

This table compares the properties of tokens generated from an overlap matrix representing the Euclidean distance between 10 equi–frequency bins in the range (-1, 1) with a maximum overlap of 5. This is a common configuration for encoding a single value token.

this is very important with respect to scaling the token generation to encode a large number of values. In the following sections I present an evaluation of the performance of each of the strategies across overlap matrices of varying complexity and sparsity.

5.2.4.2.1 Euclidean distance Equi–width bins The most simple overlap matrix is that which represents the Euclidean distance between equi–width bins. A total of 21 overlap matrices were generated using the methods described in Section 4.3.2. The properties of these overlap matrices and the parameters used to generate them are given in Table 5.13 and an example of one these overlap matrices is given in Figure 5.9.

Table 5.13: Parameters for Equi–width Euclidean Overlap Matrices

ID	Bins	Max Overlap	Density
EW0	10	5	0.700
EW1	10	10	0.900
EW2	20	5	0.430
EW3	20	10	0.730
EW4	20	15	0.900
EW5	20	20	0.950
EW6	30	5	0.320
EW7	30	10	0.550
EW8	30	15	0.730
EW9	30	20	0.870
EW10	30	25	0.940
EW11	40	5	0.230
EW12	40	10	0.430
EW13	40	15	0.600
EW14	40	20	0.740
EW15	40	25	0.840
EW16	50	5	0.190
EW17	50	10	0.360
EW18	50	15	0.500
EW19	50	20	0.630
EW20	50	25	0.740

This table shows the parameters used to generate overlap matrices used to evaluate clique selection strategies. Density refers to the proportion of non–zero values in the matrix. The data range is $(-1, 1)$ and the maximum distance is 1.0

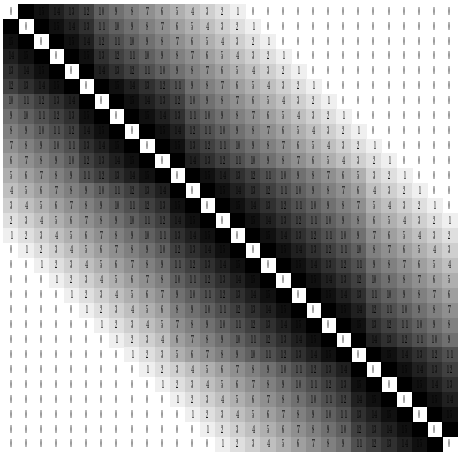


Figure 5.9: Example Overlap Matrix approximating Euclidean distance for Equi–width bins.

This Figure shows an overlap matrix approximating the Euclidean distance between 30 equi–width bins and a maximum overlap of 16. Cells a coloured on a linear scale with black cells having an overlap of 16 and white cells having no overlap.

With the exception of the overlap matrices EW1 and EW9 the token properties from the three algorithms are identical, i.e. the length, weight and required padding are the same.

The results for EW1 and EW9 are given in Table 5.14. For EW1 the MIN algorithm performs significantly worse than the EXACT algorithm. The length of the MIN tokens is 40% worse than the EXACT or PE tokens, in addition the MIN tokens require an extra two bits to be set. However the results for EW9 provide an example where the exact clique decomposition is worse than the heuristic clique decomposition. In this case MIN yields tokens that are 30 bits shorter than the EXACT method and requires a weight with one less bit set.

Table 5.14: Token Properties for Overlap Matrices EW1 and EW9

ID	Property	EXACT	MIN	PE
EW1	Length	30	48	30
	Weight	13	15	13
	Padding	8	24	8
	Time (s)	0.11	<0.01	0.02
EW9	Length	179	149	179
	Weight	27	26	27
	Padding	105	75	105
	Time (s)	1.51	0.09	0.37

This table compares the token properties and execution times for the overlap matrices EW1 and EW9 when generated using the EXACT, MIN and PE clique selection algorithms.

There is a clear ordering in terms of the execution time for each of the algorithms with MIN completing fastest, followed by PE and finally the EXACT algorithm, this can be seen in Table 5.15. Considering the relative speed of MIN and the quality of the tokens it produces in comparison to the EXACT algorithm, MIN is clearly the algorithm of choice for this type of simple overlap matrices despite is poor performance with EW0, especially when the number of bins to encode is large. However for small overlap matrices such as EW1 the runtime penalty of simply using the EXACT algorithm will be negligible for most use cases.

Table 5.15: Mean Clique Decomposition Execution Times for Euclidean Overlap Matrices

Algorithm	Mean Execution Time (s)
EXACT	1.12
MIN	0.07
PE	0.37

This table shows the mean execution time for clique decomposition of the overlap matrices specified in Table 5.13 using the EXACT, MIN and PE clique selection algorithms.

5.2.4.2.2 Euclidean distance Equi–frequency bins Overlap matrices that approximate Euclidean distance between equi–frequency bins are less regular than the overlap matrices from equi–width bins. In order to evaluate the three algorithms with less regular

overlap matrices, a total of 21 overlap matrices were generated using the methods described in Section 4.3.2. The properties of these overlap matrices and the parameters used to generate them are given in Table 5.16 and an example of these overlap matrices is given in Figure 5.10. In addition to being less regular than the equi–width overlap matrices the equi–frequency overlap matrices are also slightly sparser, this can be seen in Table 5.16.

Table 5.16: Parameters for Equi–frequency Euclidean Overlap Matrices

ID	Bins	Max Overlap	Density
EF0	10	5	0.66
EF1	10	10	0.88
EF2	20	5	0.39
EF3	20	10	0.67
EF4	20	15	0.84
EF5	20	20	0.92
EF6	30	5	0.26
EF7	30	10	0.47
EF8	30	15	0.65
EF9	30	20	0.78
EF10	30	25	0.87
EF11	40	5	0.19
EF12	40	10	0.37
EF13	40	15	0.52
EF14	40	20	0.65
EF15	40	25	0.75
EF16	50	5	0.16
EF17	50	10	0.29
EF18	50	15	0.42
EF19	50	20	0.54
EF20	50	25	0.64

This table shows the parameters used to generate overlap matrices used to evaluate clique selection strategies. Density refers to the proportion of non–zero values in the matrix. The data range is $(-1, 1)$ and the maximum distance is 1.0

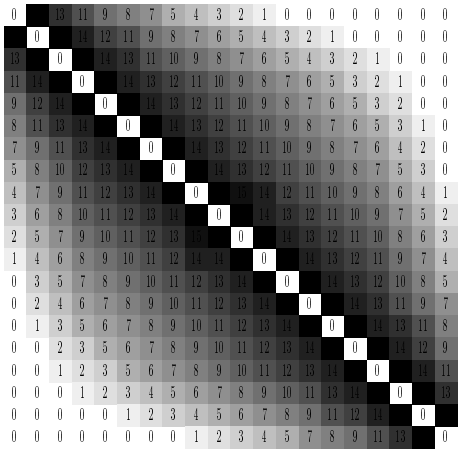


Figure 5.10: Example Overlap Matrix approximating Euclidean distance for Equi–frequency bins.

This Figure shows an overlap matrix approximating the Euclidean distance between 30 equi–frequency bins and a maximum overlap of 16. Cells a coloured on a linear scale with black cells having an overlap of 16 and white cells having no overlap.

In comparison to the equi–width overlap matrices, there is much greater variety between the performance of the three methods. The result can be viewed in Tables 5.17 and 5.18. With respect to the execution times, overall the MIN algorithm is fastest. MIN has a mean execution time that is 3.48 times faster that the EXACT method and 6.11 times faster than PE.

No strategy yielded the best tokens (shortest with lowest weight) for all overlap matrices. Indeed each of the methods provides the best tokens for several of the overlap matrices. Overall MIN results in the shortest tokens for 16 of the overlap matrices in comparison to 11 for EXACT and 7 for PE. While all three strategies provide the lowest weight for 16 of the overlap matrices.

Considering the weight of the tokens, the largest difference between the best and worst tokens for any overlap matrix is 3 bits, as a result the performance difference between using tokens generated by any of the strategies will be very small. As a result I believe overall that for the equi–frequency overlap matrices, the significant advantage in execution time observed for MIN combined with the providing the shortest tokens for most overlap matrices makes it the best clique selection strategy to use.

Table 5.17: Token Properties for Equi–Frequency Overlap Matrices

ID	Property	EXACT	MIN	PE
EF0	Length	34	43	34
	Weight	9	10	9
	Time (s)	0.010	0.009	0.011
EF2	Length	64	64	64
	Weight	9	9	9
	Time (s)	0.020	0.010	0.070
EF1	Length	39	39	39
	Weight	14	14	14
	Time (s)	0.020	0.009	0.021
EF3	Length	84	120	85
	Weight	15	17	15
	Time (s)	0.050	0.020	0.090
EF4	Length	123	137	142
	Weight	22	23	23
	Time (s)	0.100	0.040	0.120
EF5	Length	147	126	166
	Weight	28	27	29
	Time (s)	0.130	0.040	0.140
EF6	Length	124	124	124
	Weight	10	10	10
	Time (s)	0.031	0.020	0.140
EF7	Length	151	148	150
	Weight	16	16	16
	Time (s)	0.070	0.030	0.190
EF8	Length	156	156	156
	Weight	21	21	21
	Time (s)	0.150	0.050	0.240
EF9	Length	293	338	263
	Weight	31	33	30
	Time (s)	0.250	0.070	0.300
EF10	Length	388	310	414
	Weight	40	38	41
	Time (s)	0.350	0.100	0.411
EF11	Length	242	242	243
	Weight	12	12	12
	Time (s)	0.040	0.010	0.231

This table compares the token properties and execution times for the Equi–Frequency overlap matrices when generated using the EXACT, MIN and PE clique selection algorithms.

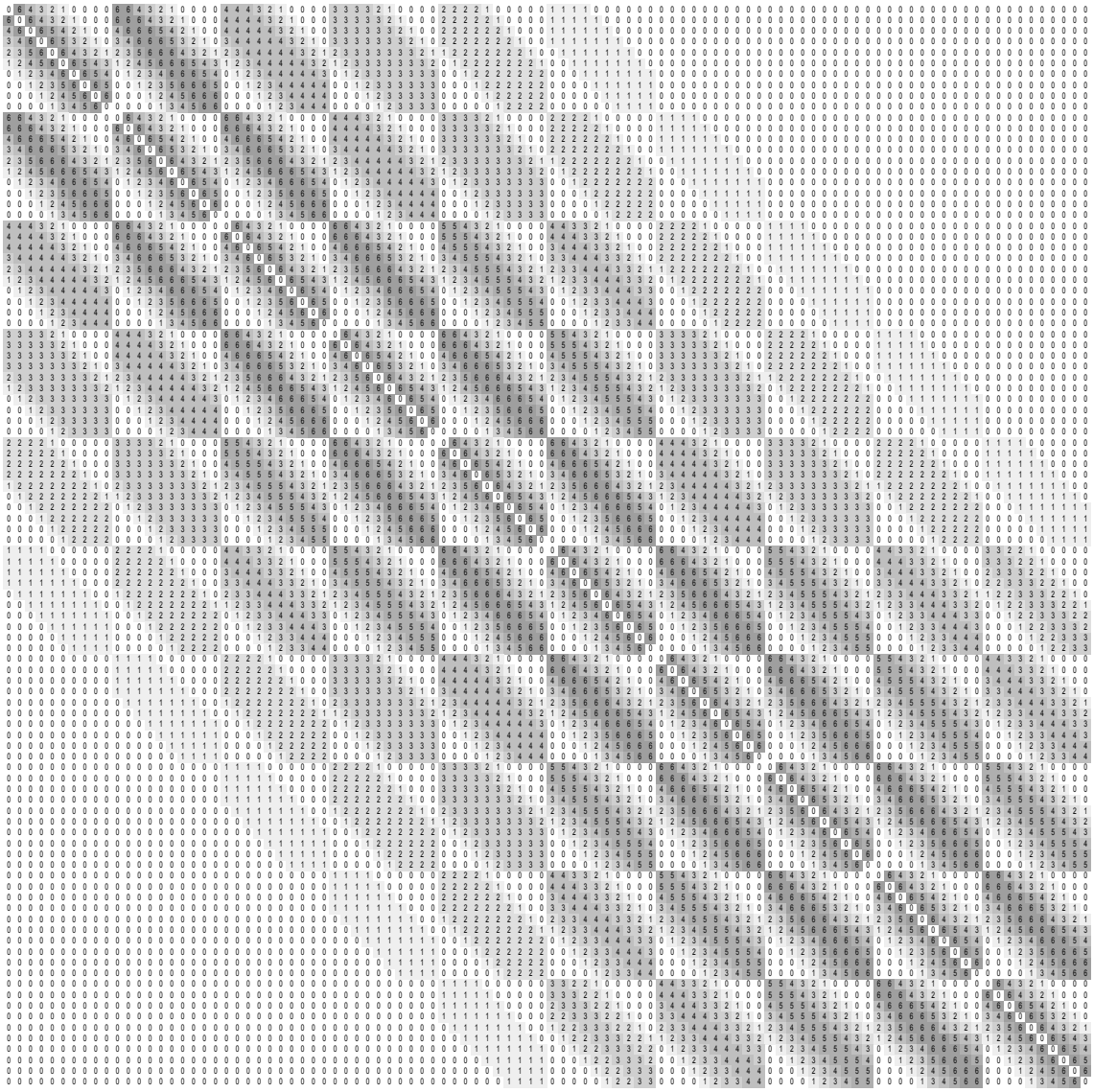
Table 5.18: Token Properties for Equi–Frequency1 Overlap Matrices

ID	Property	EXACT	MIN	PE
EF12	Length	234	230	236
	Weight	17	17	17
	Time (s)	0.110	0.030	0.340
EF13	Length	347	335	309
	Weight	25	25	24
	Time (s)	0.220	0.070	0.430
EF14	Length	562	341	518
	Weight	36	31	35
	Time (s)	0.360	0.100	0.571
EF15	Length	515	521	555
	Weight	40	42	41
	Time (s)	0.540	0.140	0.670
EF16	Length	303	302	303
	Weight	12	12	12
	Time (s)	0.050	0.020	0.370
EF17	Length	344	324	344
	Weight	18	18	18
	Time (s)	0.140	0.040	0.470
EF18	Length	334	404	336
	Weight	23	25	23
	Time (s)	0.270	0.080	0.600
EF19	Length	342	334	343
	Weight	28	28	28
	Time (s)	0.470	0.130	0.720
EF20	Length	676	596	725
	Weight	41	40	42
	Time (s)	0.700	0.170	1.020

This table compares the token properties and execution times for the Equi–Frequency overlap matrices when generated using the EXACT, MIN and PE clique selection algorithms.

5.2.4.2.3 2–Tuple Euclidean distance Equi–frequency bins The multi–feature overlap matrices generated via the n–tuple method produce significantly larger and more complicated overlap matrices. As a result, the execution times for the EXACT greedy clique decomposition of these makes it infeasible to experiment with the same number of overlap matrices as in the previous sections. The parameters used for the overlap matrices in this section are given in Table 5.19 and example of these overlap matrices is given in Figure 5.11.

Figure 5.11: Example Overlap Matrix approximating 2–Tuple Euclidean distance for Equi–frequency bins.



This Figure shows an overlap matrix approximating the Euclidean distance between 2–tuples of 10 equi–frequency bins and a maximum overlap of 16. Cells a coloured on a linear scale with black cells having an overlap of 16 and white cells having no overlap.

Table 5.19: Parameters for 2–Tuple Euclidean Overlap Matrices

ID	Tuple Size	Bins	Max Overlap	Bin Strategy	Sparsity
EW21	2	10	5	equi_width	0.60
EF21	2	10	5	equi_frequency	0.53
EW22	2	10	10	equi_width	0.99
EF22	2	10	10	equi_frequency	0.94
EW23	2	20	10	equi_wdith	0.55

This table shows the parameters used to generate overlap matrices used to evaluate clique selection strategies. Density refers to the proportion of non–zero values in the matrix.

The results of the token generation for these overlap matrices can be seen in Table 5.20. With regards to the execution time, it is clear that the EXACT method scales very poorly. A maximum overlap of 5 bits allows the EXACT algorithm to be competitive with PE however an overlap of 10 bits shows the EXACT algorithm taking around three times longer to complete. In the case of the 20 bin overlap matrix (EW23), the EXACT method takes 4 days to complete. As expected however, MIN is by far the fastest algorithm across all the instances. It is important to note that MIN required less than 5 seconds to generate tokens for EW23, this is several orders of magnitude faster than the other approaches.

The PE method clearly yields the worst tokens in terms of length and weight. However is not as clear whether MIN or EXACT perform better. EXACT has the smallest weight in the most overlap matrices (EF21, EW22 and EW23). However MIN has the smallest weight in EF22 and EW21. In addition, MIN required the fewest padding bits in all overlap matrices. In the case of EW22 this reduction in padding is sufficient to give the shortest tokens overall despite having a greater weight than the EXACT tokens.

Table 5.20: Token Properties for 2–tuple Overlap Matrices

ID	Weight	Length	Padding	Clique Algorithm	Time (s)
EF21	31	1,556	913	EXACT	4.06
EF21	35	1,643	817	MIN	0.20
EF21	32	1,643	981	PE	4.47
EF22	73	4,542	3,446	EXACT	23.56
EF22	62	2,811	1,445	MIN	0.53
EF22	64	3,776	2,804	PE	7.87
EW21	43	2,684	2,007	EXACT	5.28
EW21	38	1,876	1,036	MIN	0.20
EW21	40	2,399	1,733	PE	4.69
EW22	47	2,232	1,462	EXACT	25.70
EW22	54	2,057	799	MIN	0.60
EW22	49	2,474	1,712	PE	6.50
EW23	177	43,337	29,678	EXACT	345,669.91
EW23	199	44,876	27,757	MIN	4.99
EW23	225	61,664	47,251	PE	456.14

This table compares the token properties and execution times for the 2–tuple overlap matrices EF21, EF22, EW21, EW22 and EW23 when generated using the EXACT, MIN and PE clique selection algorithms.

5.2.4.3 Conclusions

For small overlap matrices the choice of algorithm does not have a significant effect on the outcome. However, with few exceptions, the MIN algorithm performs better than or equals the other algorithms while having an execution time that is at least an order of magnitude faster than the other algorithms for the large 2–tuple overlap matrices. However, considering the relatively short execution times for all algorithms on the small overlap matrices, the best results can be obtained by running all three algorithms and choosing the best tokens.

For larger overlap matrices, such as those encoding two or more values, this approach quickly becomes infeasible due to the required execution time for the EXACT and PE algorithms. Considering that 4 days were required to encode a relatively small 2–tuple overlap matrix of 20 bins it is necessary to exclude the EXACT method.

The PE algorithm is intended to search for cliques that are as close as possible to the size of the maximum cliques enumerated by the EXACT method. As a result it appears that the best result achieved with PE is rarely better than that of the EXACT method.

The MIN algorithm yields an inferior approximation to the EXACT method while selecting maximal cliques. However this appears to be sufficiently different from the EXACT method that it enables a different path through the clique decomposition and as a result can often yield shorter tokens with a lower weight and shorter total length than the EXACT tokens. In addition, MIN typically requires fewer padding bits than the other approaches.

In situations where MIN performs worse than the EXACT algorithm, in the worst observed case MIN tokens require 15% more bits to be set and 60% greater total length. The 60% increase in length appears to be an outlier caused by the small total length, 30

for EXACT tokens compared to 48 for MIN tokens, for large overlap matrices the total length difference is at worst 6%. Despite this potential for worse performance than the EXACT method, the speed of MIN in comparison to the other methods clearly outweighs any penalty to the token properties. For the largest overlap matrix evaluated, the EXACT method required 4 days, PE required 7 minutes and MIN required only 5 seconds to generate tokens. In this case MIN had only 12% more bits set and 3% longer tokens than the EXACT method. Clearly the speed of MIN combined with producing tokens that are comparatively short and low weight makes MIN the best clique selection algorithm to use for generating binary tokens from large overlap matrices.

5.2.4.4 Summary

In this Section, the process for converting an overlap matrix into fixed weight tokens for each of the m values was discussed.

The overlap matrix is considered as an undirected weighted graph with m nodes corresponding to the m values and the edges corresponding to similarity between the values. Initially clique decomposition is used to generate the tokens with the required Hamming distance between each value, the tokens are then padded to ensure that all tokens have a fixed weight.

Finding the optimal clique decomposition was considered infeasible for any overlap matrices of a useful size. Therefore analysis was focussed on greedy clique decomposition as it has been shown to produce relatively short tokens.

Three algorithms for selecting cliques during clique decomposition of the overlap matrix were evaluated. These are: the EXACT algorithm is based on the Bron–Kerbosch recursive backtracking maximal clique enumeration algorithm, the Penalty-Evaporation (PE) algorithm which uses a relaxed Tabu search to add and remove nodes while searching for a maximum clique; and the Min (MIN) algorithm which greedily adds the node with the most number of edges to a clique until it is maximal.

The execution time of the EXACT algorithm was shown to scale poorly as the size of the overlap matrix and the maximum overlap allowed between tokens increases. As a result it is considered infeasible to use the EXACT algorithm for large overlap matrices.

The PE algorithm typically performed worse than the EXACT algorithm, however the execution time scales up much better than the EXACT algorithm. This makes PE feasible for large overlap matrices.

However the MIN algorithm, despite being the most simple clique selection algorithm, in the majority of cases produced tokens that outperform the PE tokens and often producing tokens that are better than the EXACT tokens. In addition, the execution time of the MIN algorithm is an order to magnitude lower than the PE algorithm.

As a result the MIN algorithm is considered to provide the best trade–off between the generated token properties, such as length and weight, and the execution time required to generate the tokens.

5.2.5 Summary

In this Section I have detailed the modifications to the OBCC token generation process that I have investigated.

The first step of the OBCC process is to bin the samples. I performed an evaluation of the Equi–width and Equi–frequency binning strategies for generating OBCC tokens. The Equi–frequency binning strategy scales better with an increasing number of bins. However to ensure that the CMM remains small and therefore fast to query, the number of bins must be minimised and the difference in token length and weight between the binning strategies when using a small number of bins is very small. In addition, the Equi–width strategy provides a better representation of the samples for the datasets evaluated. As a result, the continued use of Equi–width binning is the most reasonable choice.

With regards to using OBCC to encode multiple features, the concatenation method and the n–tuple methods have been investigated. The n–tuple methods has several theoretical advantages, in particular, it is able to more accurately represent distance metrics that combine multiple features. However it scales extremely poorly as the number of feature to be encoded increases. As a result the n–tuple method is likely to be unsuitable for the number of features typically handled to the AURA kNN algorithm and so the concatenation method must be used.

The effects on the generated token length and weight of the maximum distance and maximum overlap parameters were investigated. The maximum distance parameter is important for controlling the computational requirements of the OBCC process and ideally should be kept as small as possible. While the length and weight of the generated tokens appear to scale linearly with the maximum overlap parameter.

Finally the process of generating tokens from an overlap matrix was investigated. This process is achieved by a clique decomposition of the overlap matrix. Three clique selection algorithms where evaluated with respect to the runtime requirements of the clique decomposition and the length and weight of the generated tokens. These algorithms were: an optimised version of the standard exact Bron–Kerbosch clique selection algorithm and two heuristic algorithms, MIN and Penalty Evaporation. Overall MIN was determined to be the best clique selection algorithm as a result of providing the best trade off between execution times and the length and weight of the generated tokens.

5.3 Weighted Overlap Code Construction

One of the objectives in replacing the Parabolic Kernel tokens as the input tokens for AURA kNN is to improve training and query times. However Section 5.2.4 showed that even relatively small overlap matrices can result in tokens that have a large weight in comparison to the Parabolic Kernel tokens.

For example, consider the overlap matrix EW0 from Table 5.13, the best OBCC tokens generated for EW0 has a weight of 13 bits. Since the Parabolic Kernel token is only weighted for query tokens (Section 4.3.1) only a small number of bits have to be set in the training token. The Parabolic Kernel training token generated for EW0 would only have a weight of 1 bit .

As a result there are several problems for using OBCC tokens with AURA kNN. Firstly, the additional bits have to be stored by the CMM, this means that the training time of the CMM will be increased. In addition, the CMM will be more densely populated and will therefore consume more memory while also being slower to query because there are

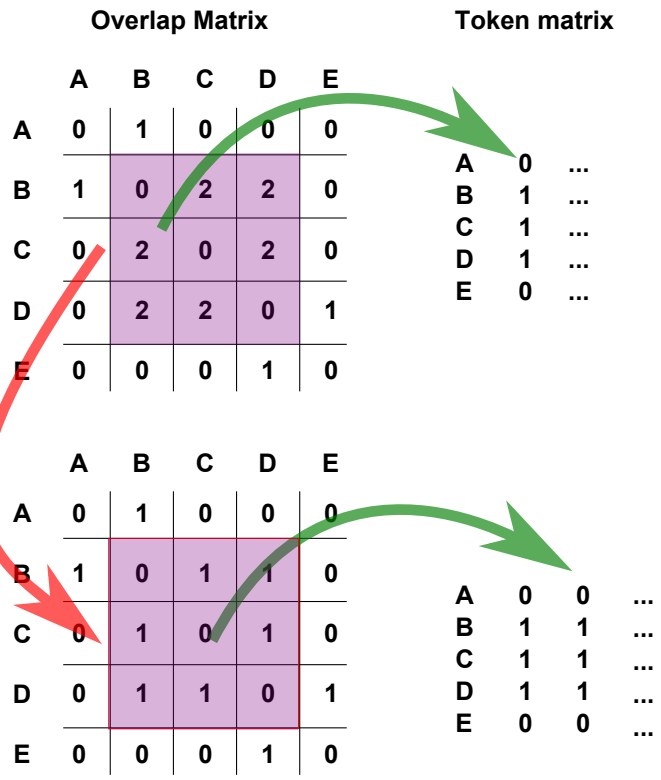


Figure 5.12: Example of Clique Decomposition Selecting Identical Cliques

This Figure illustrates how the same clique can be selected repeatedly from an overlap matrix, resulting in identical columns in the token matrix.

more bits on each matching row that have to be decoded. These problems are observed in the experiments of Chapter 6.

However because OBCC tokens have a number of theoretical advantages over Parabolic Kernel tokens, particularly the ability to encode arbitrary distance metrics, I have sought to reduce the length and particularly the weight of the OBCC tokens. This was achieved through a modification of the OBCC process called Weighted Overlap Code Construction (WOCC).

AURA CMMs are only able to store binary input and output tokens. However they are also able to support non–binary weights on the rows of an input token during a query. It is this ability to apply input weights that the Parabolic Kernel tokens exploit to indicate how much a partial match should contribute to the output score for each column. The WOCC process modifies OBCC to make use of this capability of AURA CMMs to apply weights to the input tokens.

5.3.1 Changes to the OBCC Process

Recall that during the OBCC process, cliques are selected from the graph that represents the overlap matrix. The clique is then removed from the graph and added to the token matrix as a new column. However it is possible that a clique consisting of the same set of nodes can be selected twice from the graph. This results in duplicate columns in the token matrix. An example of this can be observed in Figure 5.12

As part of the WOCC process, these duplicate columns condensed into a single column

with a weight equal to the original number of columns. There are two stages in the OBCC process where merging of columns in the token matrix can occur:

Clique Selection In the WOCC process, once a clique has been identified the minimum edge weight within the clique is found. This minimum edge weight is the weight assigned to the new column in the token matrix. When the clique is removed from the overlap matrix, the minimum edge weight is subtracted from all the edges between the clique nodes that remain in the overlap matrix.

A consequence of removing the minimum edge weight rather than the edge weight of 1 as in OBCC is that the same clique will not be repeatedly selected as the largest clique. This is because the weight of at least one edge in the clique will become zero and therefore be removed from the graph. As a result, the set of nodes that formed the previous clique will no longer be a clique in the next iteration of the graph decomposition.

Another consequence is that every clique removed with a minimum edge weight greater than 1 will reduce the overall number of bits that need to be set in the token.

Padding The padding phase of the OBCC process adds significant length to the generated tokens. However it does not add additional weight. The padding phase in WOCC consists of a single column for each row in the token matrix that is not already max weight. Each column has a single bit with a sufficient weight to make the token weight equal to max weight. As a result while the weight of the token remains the same, the number of bits that are required to be set is considerably reduced. Table 5.20 shows the padding required for overlap matrix EW23 with OBCC is 47,251 bits, in contrast, using WOCC no more than 400 padding bits will be needed.

5.3.2 Evaluation

The changes introduced by the WOCC process essentially allow a weighted CMM to be represented by a binary CMM. This is because the weight applied to a bit in position i is the same for every token that is generated from a particular overlap matrix. Therefore only one copy of the weights need to be retained.

Another benefit is that fewer bits need to be set in the CMM in order to store tokens with the same fixed weight as the equivalent OBCC tokens. Note that because the bits of a WOCC token are weighted, the weight of a WOCC token is not necessarily equal to the number of bits that are set in the token.

Tables 5.21 and 5.22 show the total length and mean number of bits set for tokens generated from the overlap matrices EW0–EW22 and EF0–EF22 with both WOCC and OBCC. The weight of the tokens generated from a specific overlap matrix is identical for each process.

Table 5.21: WOCC vs OBCC for Equi–width Overlap Matrices

ID	OBCC Length	WOCC Length	OBCC Bits	WOCC Bits
EW0	35	29	8.0	7.1
EW1	72	39	15.0	11.7
EW2	53	47	8.0	7.7
EW3	58	52	13.0	12.7
EW4	63	57	18.0	17.7
EW5	103	84	24.0	22.4
EW6	95	90	8.0	7.7
EW7	100	94	13.0	12.5
EW8	105	99	18.0	17.5
EW9	224	119	26.0	23.8
EW10	505	162	37.0	30.7
EW11	93	87	8.0	7.9
EW12	98	92	13.0	12.9
EW13	103	97	18.0	17.9
EW14	108	102	23.0	22.9
EW15	113	107	28.0	27.9
EW16	114	108	8.0	7.9
EW17	119	113	13.0	12.9
EW18	124	118	18.0	17.9
EW19	129	123	23.0	22.9
EW20	134	128	28.0	27.9
EW21	2,912	943	38.0	26.7
EW22	2,856	1,281	54.0	43.5

This table compares the length and mean number of bits set across all tokens generated via WOCC and OBCC with Equi–width binning. Note that the bits for WOCC tokens are weighted, therefore the number of bits required in each token can differ while retaining a fixed weight for the token.

Table 5.22: WOCC vs OBCC for Equi–frequency Overlap Matrices

ID	OBCC Length	WOCC Length	OBCC Bits	WOCC Bits
EF0	63	37	10.0	7.8
EF1	51	37	14.0	10.8
EF2	84	68	9.0	8.1
EF3	186	78	17.0	12.7
EF4	207	91	23.0	18.7
EF5	186	89	27.0	20.8
EF6	178	114	10.0	8.1
EF7	217	116	16.0	12.6
EF8	228	124	21.0	17.7
EF9	566	147	33.0	23.7
EF10	479	182	38.0	30.8
EF11	386	150	12.0	7.8
EF12	345	168	17.0	13.6
EF13	538	188	25.0	19.0
EF14	520	215	31.0	24.6
EF15	818	270	42.0	32.2
EF16	484	188	12.0	7.8
EF17	498	216	18.0	13.4
EF18	637	232	25.0	18.0
EF19	506	226	28.0	22.8
EF20	973	279	40.0	28.6
EF21	2,460	913	35.0	25.6
EF22	4,256	1,331	62.0	43.9

This table compares the length and mean number of bits set across all tokens generated via WOCC and OBCC with Equi–frequency binning. Note that the bits for WOCC tokens are weighted, therefore the number of bits required in each token can differ while retaining a fixed weight for the token.

It is clear that WOCC generates tokens that are significantly shorter in all cases. Across all the overlap matrices investigated, the WOCC tokens are a mean of 63% shorter than the equivalent OBCC token.

In addition, WOCC require a mean of 85% fewer bits to be set in the tokens that it generates.

5.3.3 Conclusion

The WOCC process requires a CMM implementation that is capable of supporting weighted query tokens. When weighted query tokens are not supported then standard OBCC tokens have to be used. However if such an implementation is available then the token generated by WOCC are clearly superior to OBCC tokens. This is because they are shorter and require fewer bits to be set.

As a result it is expected that using WOCC will translate into faster CMM training and query times. In addition, since the tokens for both processes are generated from the same overlap matrix, the accuracy of a CMM using either token is expected to be identical. The effects of using weighted inputs on both the training and query performance, in addition

to verifying the accuracy of WOCC tokens in comparison to OBCC tokesn is investigated as part of the experiments in Chapter 6.

5.4 Multi Bit Output Tokens

The current version of AURA kNN uses unary output tokens to represent the samples to be learned by the CMM. There are two major advantages to this approach.

Firstly the use of unary tokens allows the result of the L–max threshold on the output scores be easily decoded into the L samples that closest match the query sample.

In addition, unary tokens are orthogonal to each other and therefore eliminate the effect of crosstalk within the CMM. Crosstalk occurs when a column within the CMM contains information about multiple samples that have been stored in the CMM and refers to the noise introduced by storing information about multiple samples in a column. The more crosstalk that is present, the greater the likelihood that a query will return an erroneous result (Hobson 2011).

However, as a result of using unary tokens, the AURA kNN query essentially becomes a linear scan of the binary encoded samples. As a result the only major advantage of AURA kNN over a simple linear scan is that the binary comparison is less computationally expensive than computing the Euclidean distance between samples.

If, instead of using unary tokens, the weight of the output token is increased, then the token is able to represent a larger volume of information. Similarly it means that a given amount of information can be stored in a shorter token if a larger weight is used. A shorter token yields a smaller CMM and results in faster query times.

Unfortunately increasing the weight of the output token will also increase the amount of crosstalk in the CMM and will therefore reduce the accuracy of the kNN query.

5.4.1 Baum Codes

Baum Codes (Baum, Moody, and Wilczek 1988), described in Section 4.4.2, provide a method for generating output tokens that balance the desire to increase the amount information that can be stored in a token of a given length while also minimising the amount of crosstalk.

Figure 5.13 shows how using Baum coded output tokens with a weight of 2 to represent samples to be stored in the CMM requires considerably shorter tokens in comparison to the Unary tokens currently used by AURA kNN.

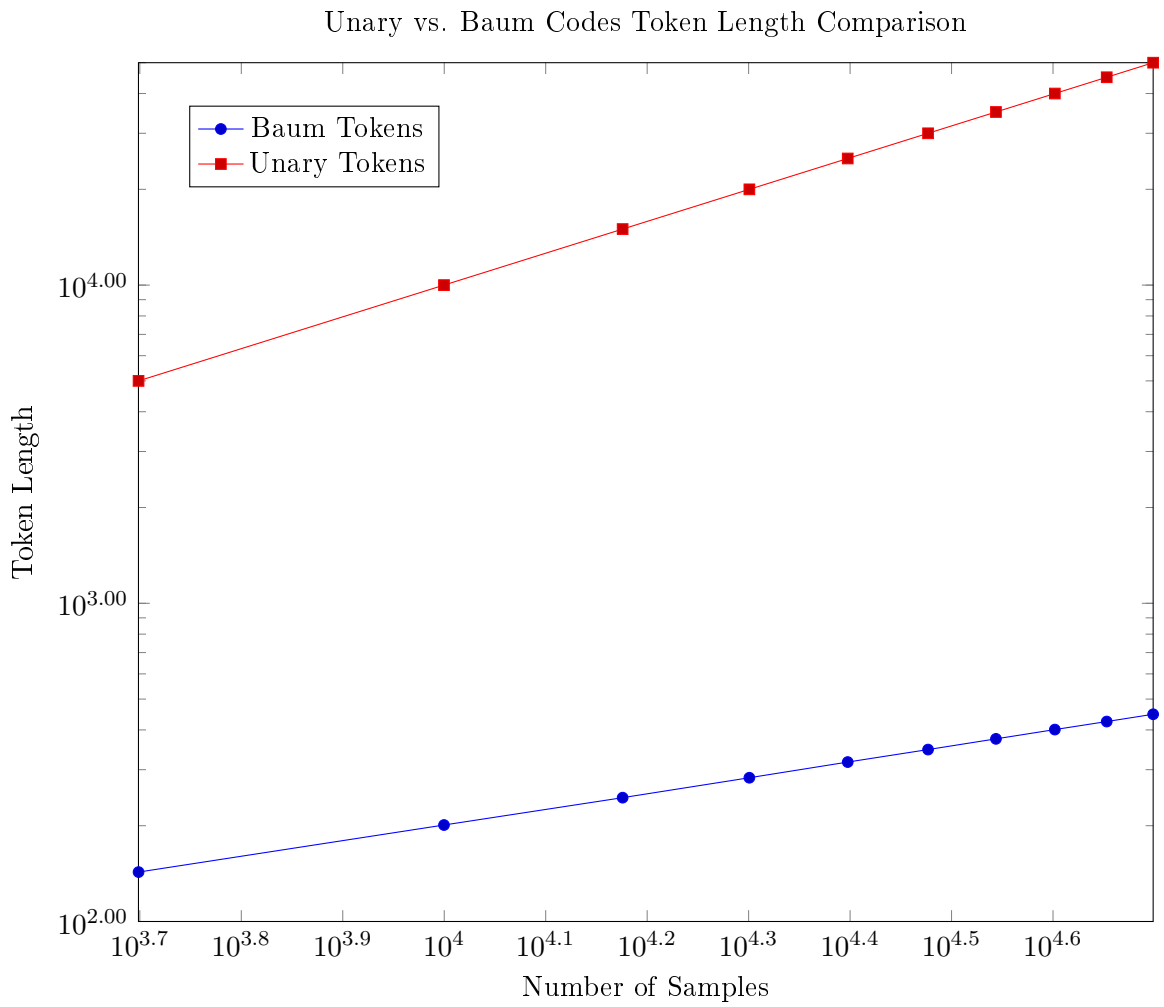


Figure 5.13: A comparison of the token length between Unary tokens and 2 Bit Baum Coded Tokens encoding an increasing number of samples.

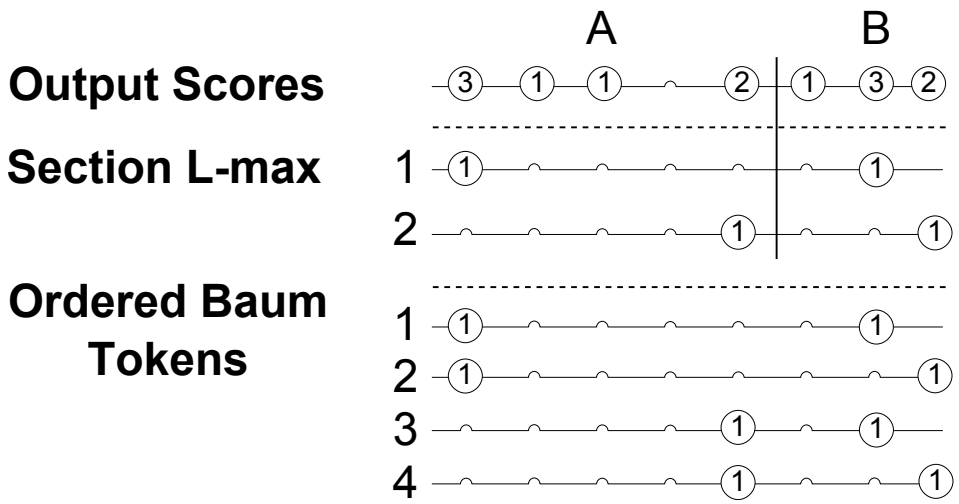
It is clear from Figure 5.13 that the 2–bit Baum Tokens are considerably shorter than the Unary Tokens. A 2–bit Baum token of length 143 is needed to uniquely represent 5,000 samples while the equivalent Unary token requires length 5,000. However while it is clear that the length of the 2–bit Baum tokens grow much more slowly than Unary tokens, they still appear to grow linearly with the number of samples to be encoded. A tenfold increase in the number of samples to be encoded results in a tenfold increase in the length of the 2–Bit Baum token with 50,000 samples requiring a token of length 1,415 to be uniquely encoded.

Despite this, the slower growth of the 2–Bit Baum tokens means that a CMM can store large datasets with an order of magnitude fewer columns than using Unary tokens. This will result in a must faster query of the CMM.

5.4.2 Output Threshold

As part of the CMM query it is necessary to threshold the output scores so that the best matches can be retrieved. Hobson (2011) observed that the standard threshold algorithms, L–max (Austin 1996) and Willshaw (Willshaw, Buneman, and Longuet-Higgins 1969), will often retrieve tokens that are not valid Baum tokens, despite using Baum tokens as the output tokens in the CMM. As a result the L–WTA algorithm (Hobson 2011), described

Figure 5.14: Illustration of the L–max–Baum threshold



This Figure illustrates the steps required for the L–max–Baum threshold. The output scores are first divided into 2 sections, Section A of length 5 and Section B of length 3. The L–max threshold, $l = 2$ is applied to each of the sections yielding two subtokens for each section ordered by match score. The highest matching subtokens are combined to form Baum token 1, Baum token 2 is formed by combining the highest match from A with the second match from B, Baum token 3 is formed by combining the second match from A with the highest match from B and the lowest matching subtokens are combined to form Baum token 4.

in Section 4.5, was introduced to restrict the results of the threshold step to valid Baum tokens. L–WTA provided a reduction in the number of query errors observed when using Baum Coded output tokens.

However the L–WTA threshold only provides the single best match in the CMM. In order to implement a kNN search it is necessary to be able to retrieve at least k potential matches. L–max–Baum is a new threshold function that combines the ability of the L–max threshold function to retrieve multiple matches and the ability of the L–WTA threshold function only provide valid Baum tokens.

Similar to L–WTA, the L–max–Baum threshold function divides the output scores into multiple sections of equal length to the sections originally used to generate the Baum codes, see Section 4.4.2. The L–max threshold is then applied individually to each of the sections. The desired Baum codes are then constructed by combining the items retrieved from each of the sections. In order to identify the L best matches, it is necessary to impose an ordering on the Baum codes that are produced. This ordering is defined by the match score of each of its constituent sections. An example of this process is given in Figure 5.14.

5.4.3 Summary

In this Section, the effect of increasing the weight of the output tokens was examined. This is known to cause crosstalk between samples since multiple samples are superimposed onto each column of a CMM. Since Baum codes can be used to generate a set of tokens that are relatively orthogonal to each other, it was decided that Baum tokens should be used as the output tokens in order to manage this crosstalk.

The use of 2–Bit Baum tokens requires significantly fewer columns to be needed by a CMM. Using 2–Bit Baum tokens, only 1415 columns are needed to store 50,000 unique

samples, while the Unary tokens would require a column for each sample to be stored. As a result the CMM query can be substantially faster due to the use of a smaller CMM.

However existing threshold functions are not suited to decoding the multiple Baum coded tokens needed to implement a kNN search. Therefore the L–max–Baum threshold was presented in order to provide this capability.

5.5 Summary

In this Chapter, several changes to the process for selecting input and output tokens used for the CMM component of AURA kNN have been investigated. Initially the process of generating OBCC tokens was examined and by using the max distance parameter to limit computation and the heuristic clique selection algorithm, MIN, it is feasible to generate OBCC tokens for a large range of values.

Weighted Overlap Code Construction (WOCC) was introduced as a method for generating tokens that are equivalent to OBCC tokens but that are shorter and require fewer bits to be set.

Finally, the use of Baum coded output tokens was investigated to reduce the size of a CMM needed to store a given dataset. This required a new threshold function, L–max–Baum, so that Baum tokens can be used as part of AURA kNN.

Chapter 6

Baseline Nearest Neighbour Experiments for Timeseries Data

6.1 Introduction

This Chapter consists of three Sections. In Section 6.2 the experiments are described that are used in both this Chapter and the following Chapters to evaluate: the standard kNN algorithms described in Section 3.11; the standard AURA kNN algorithm described in Chapter 4; and the modifications to the standard AURA kNN algorithm presented in Chapter 5; with a view to determining whether there is any advantage to replacing the kNN algorithm used within AURA Alert.

Section 6.3 contains the results of these experiments for the exact kNN algorithms: Linear Scan, KD–Tree (Bentley 1975) and Dual KD–Tree (Gray and Moore 2000).

Finally Section 6.4 contains the results of these experiments for the LSH algorithm (Indyk and Motwani 1998), a state of the art approximate kNN algorithm.

The results in these sections are compared to the expected results that are predicted by the relevant literature. This is to verify that the experiments have been implemented correctly. In addition these results form a baseline with which to compare the AURA kNN algorithm.

6.2 Experiment Design

In this Section, the experiment design is presented for the experiments that follow in the rest of the Chapter.

As described in Section 4.6, AURA Alert models the normal behaviour of a complex system by performing a fast approximate kNN search across a historical database of system states to determine whether a new reading of the system state differs significantly from its previous readings. The current implementation of AURA Alert is based on the AURA kNN algorithm, described in Chapter 4, using Parabolic Kernel (PK) input tokens and Unary output tokens. The purpose of these experiments is to determine whether there are alternative kNN algorithms that would provide superior performance within the context of AURA Alert.

6.2.1 Objectives

In order to identify whether an alternative kNN algorithm is likely to perform better as part of AURA Alert. The experiments have been designed in order to answer the following questions:

1. How is the execution time of each algorithm affected by the number of samples in the dataset?
2. How is the execution time of each algorithm affected by the number of features in each sample?
3. How does the execution time compare between each of the algorithms?

When considering the approximate nearest neighbour algorithms:

1. How is the accuracy of the retrieved neighbours affected by the number of samples in the dataset?
2. How is the accuracy of the retrieved neighbours affected by the number of features possessed by each sample?

In order to compare the methods for generating input tokens used for the CMM query as part of AURA kNN, the following questions are to be answered:

1. Which of the methods for generating input tokens results in the most accurate version of AURA kNN?
2. Which of the methods for generating input tokens results in the fastest version of AURA kNN?

Finally, since Section 5.2.1.2 showed that the length and weight of the input tokens are significantly affected by the number of bins used in representing a data range:

1. What effect does the number of bins have on the execution time of AURA kNN?
2. What effect does the number of bins have on the accuracy of AURA kNN?

The rest of this Section will describe the experiments that have been designed in order to answer these questions.

6.2.2 Evaluation Criteria

The questions presented in Section 6.2.1 require that the execution time and the accuracy of the algorithms be evaluated.

The accuracy of an algorithm will be measured by comparing the mean feature distance from a query sample to the set of 25 neighbours returned by the query with the mean feature distance from the query sample to the known true set of neighbours given by the Linear Scan algorithm, the most basic exact kNN algorithm.

The method for computing the feature distance is given in Equation 6.1. The mean feature distance is simply the mean of the feature distances between the query sample and each of the retrieved neighbours.

The feature distance was chosen as the measurement for comparison rather than, for example, the mean squared error (Hastie et al. 2009), because it aids in the comparison between results from datasets with different numbers of features by presenting a measure that represents the contribution of a single feature to the observed distance.

$$feature\_distance = \frac{distance(query, neighbour)}{n\_features} \tag{6.1}$$

In all cases a small feature distance is desired. However when comparing approximate algorithms, it is desired that the mean feature distance be close to the mean feature distance of an exact algorithm.

In evaluating the execution time of the algorithms it is necessary to consider both the time required to perform a query and the time required to perform any setup required to facilitate the query. These are termed the query time and the training time respectively. The results of these experiments are reported in seconds and the best results are those that complete in the shortest period of time.

6.2.3 Datasets

This Section provides details about the datasets used by the kNN experiments.

6.2.3.1 Synthetic Datasets

The synthetic datasets have been generated so that it is easier to evaluate the effects of specific dataset characteristics. With this in mind synthetic datasets have been generated consisting of $1,000$, $10,000$, $100,000$ and $1,000,000$ samples. In addition the number of features varies with datasets containing 2, 10, 100, and 1000 features. All values in the synthetic datasets are randomly generated, with two datasets for each combination of dataset size and number of features.

One dataset consists of samples drawn from both a random Uniform distribution and another dataset with samples drawn from a Gaussian distribution, both with a mean of 0 and values drawn such that three standard deviations from the mean falls within the range $[-1, 1]$. The characteristics of all the synthetic datasets are provided in Table 6.1. Note that there are no datasets of 1000 features and 1,000,000 samples, this is simply due to memory constraints on the compute cluster used to perform the experiments.

The Uniform random datasets were created to evaluate the algorithms when the data is spread out across a large portion of the data space.

The random Gaussian datasets were created because this is more typical of the type of data that is likely to be queried by AURA Alert. This is because, when normalised, time series data typically has a Gaussian distribution (Lin et al. 2003) (See Appendix A).

It is expected that most uses of AURA Alert will require less than 100 features. However the 1,000 feature datasets are included because it is possible that future applications of AURA Alert could potentially require large numbers of features.

Table 6.1: Properties of the Synthetic Datasets

ID	Number of Samples	Number of Features	Random Distribution
SYNTH01	1,000	2	Uniform
SYNTH02	1,000	2	Gaussian
SYNTH03	1,000	10	Uniform
SYNTH04	1,000	10	Gaussian
SYNTH05	1,000	100	Uniform
SYNTH06	1,000	100	Gaussian
SYNTH07	1,000	1,000	Uniform
SYNTH08	1,000	1,000	Gaussian
SYNTH09	10,000	2	Uniform
SYNTH10	10,000	2	Gaussian
SYNTH11	10,000	10	Uniform
SYNTH12	10,000	10	Gaussian
SYNTH13	10,000	100	Uniform
SYNTH14	10,000	100	Gaussian
SYNTH15	10,000	1,000	Uniform
SYNTH16	10,000	1,000	Gaussian
SYNTH17	100,000	2	Uniform
SYNTH18	100,000	2	Gaussian
SYNTH19	100,000	10	Uniform
SYNTH20	100,000	10	Gaussian
SYNTH21	100,000	100	Uniform
SYNTH22	100,000	100	Gaussian
SYNTH23	100,000	1,000	Uniform
SYNTH24	100,000	1,000	Gaussian
SYNTH25	1,000,000	2	Uniform
SYNTH26	1,000,000	2	Gaussian
SYNTH27	1,000,000	10	Uniform
SYNTH28	1,000,000	10	Gaussian
SYNTH29	1,000,000	100	Uniform
SYNTH30	1,000,000	100	Gaussian

This table shows the properties of the 30 synthetic datasets generated for the k–Nearest Neighbours experiments.

The number of samples generated for each of the datasets was chosen to represent a variety of dataset sizes within the range typically handled by AURA Alert.

A matching test dataset consisting of samples is generated with the same number of features and from the same random distribution. These test datasets are used to provide the query samples for nearest neighbour search.

6.2.3.2 Real World Datasets

The real world datasets were chosen so that the results from the synthetic datasets can be validated on realistic data. For this reason the UCR Time Series Classification (Keogh et al. 2011) datasets were chosen as a set of datasets that are frequently used when comparing kNN algorithms that operate on time series data. These are small datasets and are therefore primarily of use for validating the accuracy of the methods rather than their runtime performance. There are a total of 48 datasets of various sizes and numbers of features. These are listed in Table 6.2.

6.2.4 Experiment Hardware

All experiments were performed on the York Advanced Research Computing Cluster (YARCC). A node in the cluster consisted of a single Intel Xeon E5-2670 CPU at 2.6 GHz and 8GB of RAM.

Each node was configured to exclusively run a single experiment. This is to prevent other workloads on the system interfering with the timing results. The YARCC Cluster currently consists of 27 Nodes.

6.2.5 Experiment Parameters

In order to ensure the accuracy and consistency of the experiment results, every experiment, consisting of a dataset and an algorithm, was executed with 10 repetitions. The standard error of the mean for these results is presented via error bars on the charts that plot the results. Any significant deviation across the repetitions will be noted when the results are discussed.

Table 6.2: Properties of the UCR Time Series Classification Datasets

ID	Training Samples	Query Samples	Features
50Words	450	455	270
Adiac	390	391	176
Beef	30	30	470
Car	60	60	577
CBF	30	900	128
ChlorineConcentration	467	3840	166
CinC_ECG_torso	40	1380	1639
Coffee	28	28	286
Cricket_X	390	390	300
Cricket_Y	390	390	300
Cricket_Z	390	390	300
DiatomSizeReduction	16	306	345
ECG	100	100	96
ECGFiveDays	23	861	136
Face (all)	560	1690	131
Face (four)	24	88	350
FacesUCR	200	2050	131
Fish	175	175	463
Gun-Point	50	150	150
Haptics	155	308	1092
InlineSkate	100	550	1882
ItalyPowerDemand	67	1029	24
Lightning-2	60	61	637
Lightning-7	70	73	319
MALLAT	55	2345	1024
MedicalImages	381	760	99
MoteStrain	20	1252	84
Non-Invasive Fetal ECG Thorax1	1800	1965	750
Non-Invasive Fetal ECG Thorax2	1800	1965	750
OliveOil	30	30	570
OSU Leaf	200	242	427
Plane	105	105	144
SonyAIBORobot Surface	20	601	70
SonyAIBORobot SurfaceII	27	953	65
StarLightCurves	1000	8236	1024
Swedish Leaf	500	625	128
Symbols	25	995	398
Synthetic Control	300	300	60
Trace	100	100	275
Two Patterns	1000	4000	128
TwoLeadECG	23	1139	82
uWaveGestureLibrary_X	896	3582	315
uWaveGestureLibrary_Y	896	3582	315
uWaveGestureLibrary_Z	896	3582	315
Wafer	1000	6174	152
WordsSynonyms	267	638	270
Yoga	300	3000	426

This table shows the properties of the 48 real world datasets used for the k–Nearest Neighbours experiments.

6.3 Exact Algorithm Results

In this Section the performance of the exact algorithms, Linear search, KD–Tree and Dual KD-Tree, has been evaluated with respect to the real world and synthetic datasets detailed in Sections 6.2.3. Given that the performance and accuracy of the exact algorithms can be predicted from their theoretical properties, the purpose of this evaluation is primarily to demonstrate that these algorithms are operating as expected and verify the experimental setup. This is important because the exact algorithms form the baseline comparison for the evaluation of approximate algorithms that follow.

Accuracy

Since the Linear, KD–Tree and Dual KD–Tree algorithms are all exact algorithms, it is expected that the accuracy will be identical across all the datasets.

The results of the Linear, KD–Tree and Dual KD–Tree experiments can be seen in Tables 6.3 and 6.4. As expected, the mean features distance is identical across all the algorithms.

As a result it is reasonable to conclude that the implementations of the Linear, KD–Tree and Dual KD–Tree algorithms are correct.

Table 6.3: Mean Feature Distance for Exact Algorithms (Synthetic)

ID	Features	Samples	Linear	KD Tree	Dual KD Tree
SYNTH01	2	1,000	0.062	0.062	0.062
SYNTH02	2	1,000	0.065	0.065	0.065
SYNTH03	10	1,000	0.142	0.142	0.142
SYNTH04	10	1,000	0.124	0.124	0.124
SYNTH05	100	1,000	0.071	0.071	0.071
SYNTH06	100	1,000	0.061	0.061	0.061
SYNTH07	1,000	1,000	0.025	0.025	0.025
SYNTH08	1,000	1,000	0.021	0.021	0.021
SYNTH09	2	10,000	0.019	0.019	0.019
SYNTH10	2	10,000	0.022	0.022	0.022
SYNTH11	10	10,000	0.107	0.107	0.107
SYNTH12	10	10,000	0.096	0.096	0.096
SYNTH13	100	10,000	0.067	0.067	0.067
SYNTH14	100	10,000	0.058	0.058	0.058
SYNTH15	1,000	10,000	0.024	0.024	0.024
SYNTH16	1,000	10,000	0.021	0.021	0.021
SYNTH17	2	100,000	0.006	0.006	0.006
SYNTH18	2	100,000	0.007	0.007	0.007
SYNTH19	10	100,000	0.082	0.082	0.082
SYNTH20	10	100,000	0.076	0.076	0.076
SYNTH21	100	100,000	0.064	0.064	0.064
SYNTH22	100	100,000	0.055	0.055	0.055
SYNTH23	1,000	100,000	0.024	0.024	0.024
SYNTH24	1,000	100,000	0.021	0.021	0.021
SYNTH25	2	1,000,000	0.002	0.002	0.002
SYNTH26	2	1,000,000	0.002	0.002	0.002
SYNTH27	10	1,000,000	0.063	0.063	0.063
SYNTH28	10	1,000,000	0.060	0.060	0.060
SYNTH29	100	1,000,000	0.062	0.062	0.062
SYNTH30	100	1,000,000	0.053	0.053	0.053

This table shows the mean feature distance for the Linear, KD–Tree and Dual KD–Tree algorithms. Since these are all exact algorithms, as expected, the results are identical.

Table 6.4: Mean Feature Distance for Exact Algorithms (Real)

ID	Features	Samples	Linear	KD Tree	Dual KD Tree
50words	270	450	0.046	0.046	0.046
Adiac	176	390	0.005	0.005	0.005
Beef	470	30	0.002	0.002	0.002
CBF	128	30	0.093	0.093	0.093
ChlorineConcentration	166	467	0.011	0.011	0.011
CinC_ECG_torso	1,639	40	0.032	0.032	0.032
Coffee	286	28	0.208	0.208	0.208
Cricket_X	300	390	0.044	0.044	0.044
Cricket_Y	300	390	0.044	0.044	0.044
Cricket_Z	300	390	0.044	0.044	0.044
DiatomSizeReduction	345	16	0.010	0.010	0.010
ECG200	96	100	0.053	0.053	0.053
ECGFiveDays	136	23	0.081	0.081	0.081
FaceAll	131	560	0.072	0.072	0.072
FaceFour	350	24	0.063	0.063	0.063
FacesUCR	131	200	0.079	0.079	0.079
Gun_Point	150	50	0.034	0.034	0.034
Haptics	1,092	155	0.012	0.012	0.012
InlineSkate	1,882	100	0.013	0.013	0.013
ItalyPowerDemand	24	67	0.074	0.074	0.074
Lighting2	637	60	0.039	0.039	0.039
Lighting7	319	70	0.052	0.052	0.052
MALLAT	1,024	55	0.009	0.009	0.009
MedicalImages	99	381	0.045	0.045	0.045
MoteStrain	84	20	0.110	0.110	0.110
NonInvasiveFatalECG_Thorax1	750	1,800	0.004	0.004	0.004
NonInvasiveFatalECG_Thorax2	750	1,800	0.004	0.004	0.004
OSULeaf	427	200	0.044	0.044	0.044
OliveOil	570	30	0.000	0.000	0.000
SonyAIBORobot_Surface	70	20	0.089	0.089	0.089
SonyAIBORobot_SurfaceII	65	27	0.131	0.131	0.131
StarLightCurves	1,024	1,000	0.007	0.007	0.007
SwedishLeaf	128	500	0.029	0.029	0.029
Symbols	398	25	0.054	0.054	0.054
Trace	275	100	0.027	0.027	0.027
TwoLeadECG	82	23	0.055	0.055	0.055
Two_Patterns	128	1,000	0.083	0.083	0.083
WordsSynonyms	270	267	0.051	0.051	0.051
fish	463	175	0.009	0.009	0.009
synthetic_control	60	300	0.106	0.106	0.106
test	3	10	0.379	0.379	0.379
uWaveGestureLibrary_X	315	896	0.031	0.031	0.031
uWaveGestureLibrary_Y	315	896	0.028	0.028	0.028
uWaveGestureLibrary_Z	315	896	0.030	0.030	0.030
wafer	152	1,000	0.029	0.029	0.029
yoga	426	300	0.019	0.019	0.019

This table shows the mean feature distance for the Linear, KD–Tree and Dual KD–Tree algorithms. Since these are all exact algorithms, as expected, the results are identical.

Query Performance

The mean query times for the Linear, KD–Tree and Dual KD–Tree algorithms on the synthetic datasets can be found in Table 6.5. The query times for the real world datasets are not evaluated because the datasets are not large enough to draw meaningful conclusions about the performance of the algorithms. Specifically, some of the datasets contain so few

samples that the execution time of the query is less than the resolution of the system clock and any variances observed at this level are more likely to be caused by operating system overheads than differences in the algorithms.

Table 6.5: Mean Query Time for Exact Algorithms (Synthetic)

ID	Features	Samples	Linear	KD Tree	Dual KD Tree
SYNTH01	2	1,000	0.123	0.005	0.007
SYNTH02	2	1,000	0.139	0.006	0.013
SYNTH03	10	1,000	0.154	0.030	0.029
SYNTH04	10	1,000	0.160	0.028	0.026
SYNTH05	100	1,000	0.438	0.165	0.147
SYNTH06	100	1,000	0.466	0.183	0.166
SYNTH07	1,000	1,000	3.585	1.902	1.167
SYNTH08	1,000	1,000	3.519	1.812	1.307
SYNTH09	2	10,000	1.154	0.007	0.044
SYNTH10	2	10,000	1.236	0.008	0.052
SYNTH11	10	10,000	1.411	0.254	0.212
SYNTH12	10	10,000	1.445	0.229	0.218
SYNTH13	100	10,000	4.524	3.959	1.447
SYNTH14	100	10,000	4.624	4.114	1.359
SYNTH15	1,000	10,000	34.654	19.448	12.863
SYNTH16	1,000	10,000	34.189	18.718	12.216
SYNTH17	2	100,000	11.931	0.016	0.319
SYNTH18	2	100,000	11.862	0.011	0.324
SYNTH19	10	100,000	14.250	1.245	1.898
SYNTH20	10	100,000	14.834	2.267	2.105
SYNTH21	100	100,000	44.913	37.610	13.699
SYNTH22	100	100,000	44.026	46.588	13.688
SYNTH23	1,000	100,000	345.050	211.882	119.184
SYNTH24	1,000	100,000	352.514	217.079	130.298
SYNTH25	2	1,000,000	116.661	0.018	2.684
SYNTH26	2	1,000,000	116.297	0.020	3.093
SYNTH27	10	1,000,000	143.728	2.922	20.675
SYNTH28	10	1,000,000	144.761	6.058	19.216
SYNTH29	100	1,000,000	450.798	422.941	148.958
SYNTH30	100	1,000,000	433.448	411.861	140.706

This table shows the query time in seconds for the Linear, KD–Tree and Dual KD–Tree algorithms.

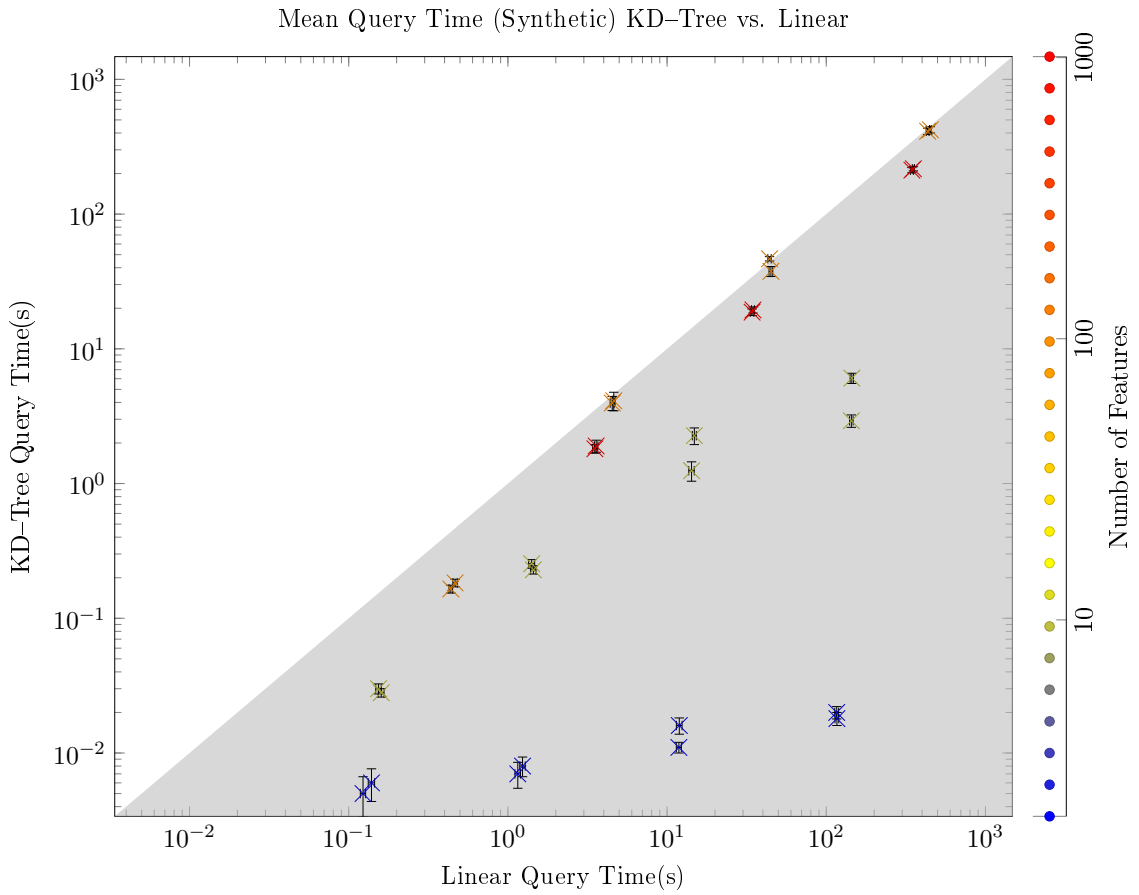


Figure 6.1: A comparison of the mean query time in seconds for each of the synthetic datasets. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A smaller query time is better. The shaded area indicates where the KD–Tree performance is superior to the Linear algorithm.

Figure 6.1 provides a comparison between the query times of the Linear algorithm and the KD–Tree algorithm. It is clear that the performance characteristics of the KD–Tree algorithm are as expected. When the number of features in the dataset is low, the KD–Tree vastly outperforms the Linear algorithm. The mean execution time of the KD–Tree algorithm on the 1,000,000 sample, 2 feature datasets is 0.016% of the Linear Scan execution time.

However as the number of features increases, KD–Tree performance degrades to match that of the Linear algorithm. The 100 feature, 1,000,000 sample datasets have a mean execution time of 94.4% of the Linear Scan execution time. It is worth noting that even for 10 features, a relatively small number of features, the performance degradation is very significant.

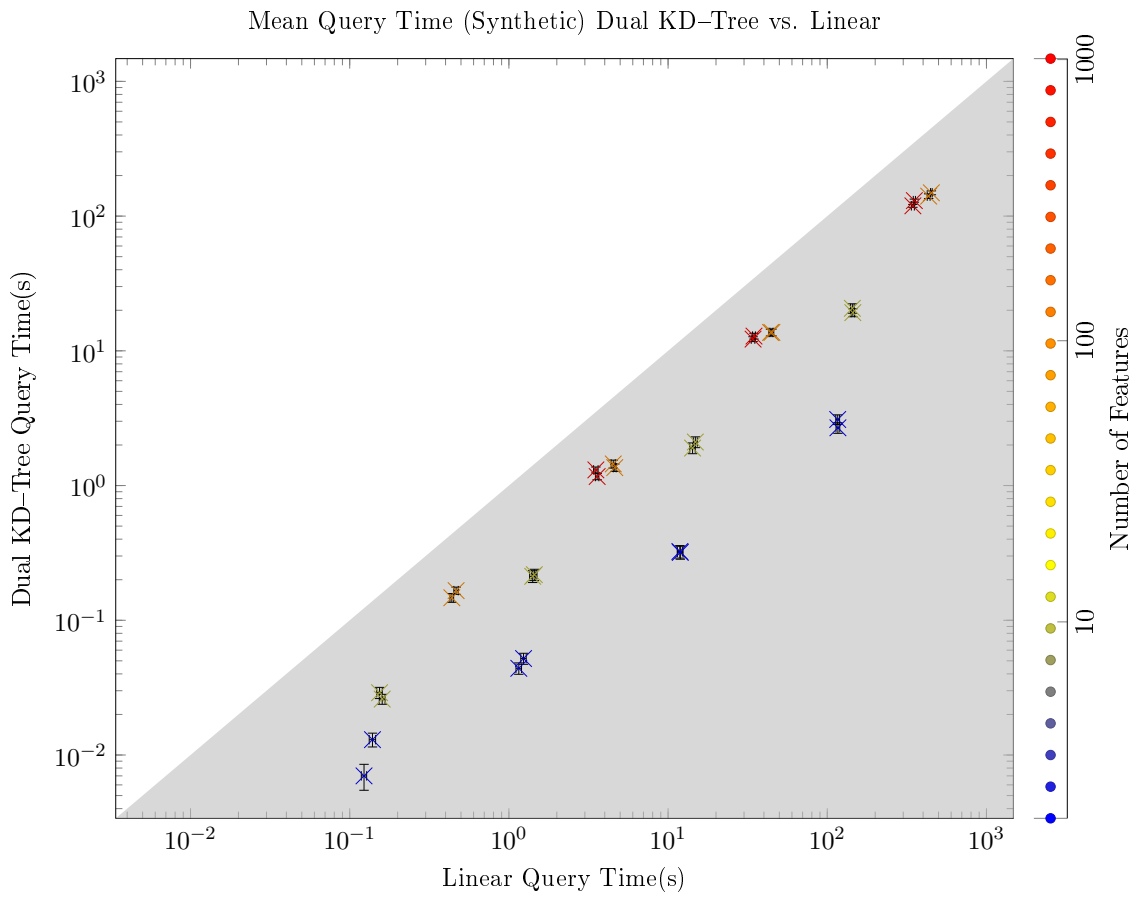


Figure 6.2: A comparison of the mean query time in seconds for each of the synthetic datasets. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A smaller query time is better. The shaded area indicates where the Dual KD–Tree performance is superior to the Linear algorithm.

Figure 6.2 provides a comparison between the query times of the Linear algorithm and the Dual KD–Tree algorithm. It is clear that the Dual KD–Tree algorithm is much faster for all of the synthetic datasets. Across all the datasets, the mean execution time of the Dual KD–Tree algorithm is just 21% of the mean execution time for the Linear Scan.

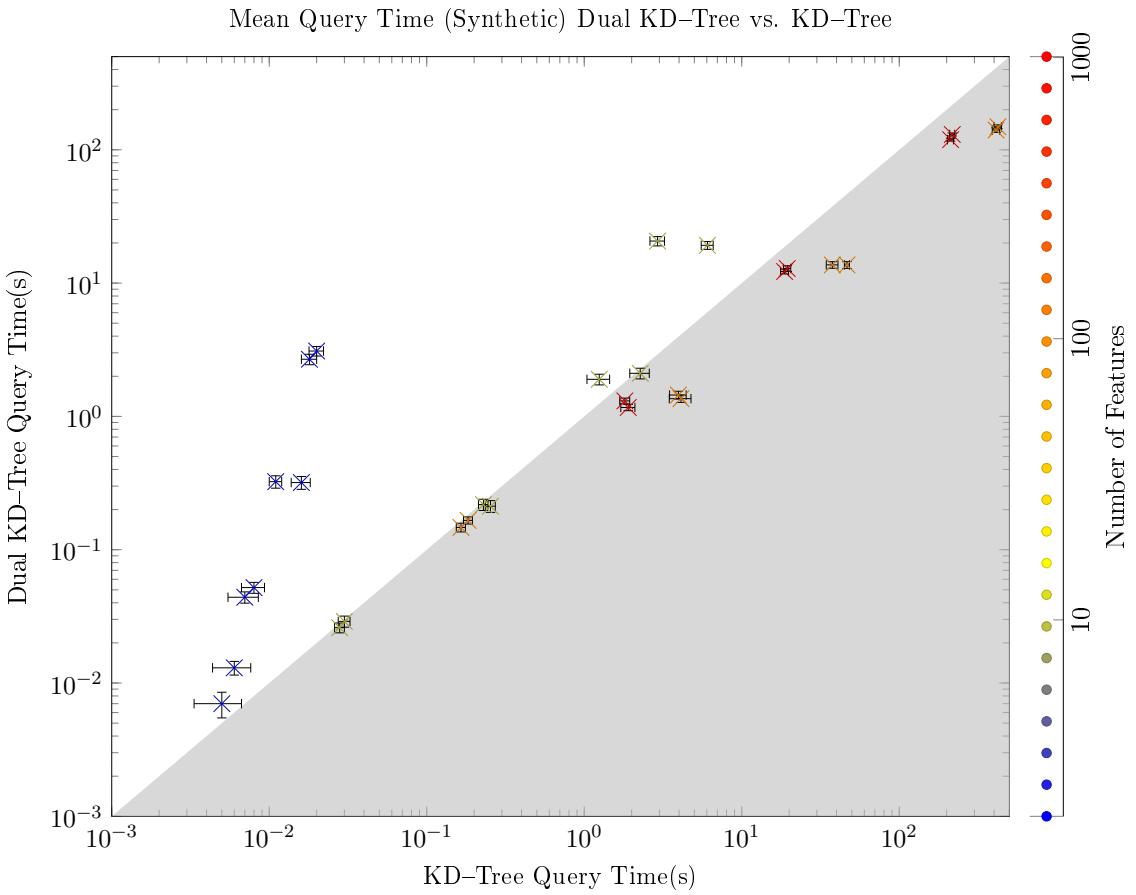


Figure 6.3: A comparison of the mean query time in seconds for each of the synthetic datasets. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A smaller query time is better. The shaded area indicates where the Dual KD–Tree performance is superior to the Linear algorithm.

The advantage of the Dual KD–Tree over the standard KD–Tree for datasets with many features can clearly be seen in Figure 6.3. For datasets SYNTH29 and SYNTH30, the rightmost markers, each consisting of 100 features and 1,000,000 samples the Dual KD–Tree algorithm has a mean execution time that is only 35% of the execution time for the standard KD–Tree.

However it is clear that the overhead incurred in this experiment for datasets with few features is sufficient to negate the theoretical advantages of this approach over the standard KD–Tree. In the worst case, datasets SYNTH25 and SYNTH26, consisting of 2 features and 1,000,00 samples, the execution time of the Dual KD–Tree algorithm is 152 longer than the standard KD–Tree algorithm.

Training Performance

The mean training times in seconds for the Linear, KD–Tree and Dual KD–Tree algorithms are provided in Table 6.6. As expected for the Linear algorithm this step completes almost instantly because there is no preprocessing that is required to be performed, the actual time taken by the experiment is less than the millisecond resolution of the machines that the experiments are performed on.

The training times for KD–Tree and Dual KD–Tree are very similar, this was also to

be expected. The median training times for the KD–Tree and Dual KD–Tree algorithms are 0.08s and 0.08s with the difference in distribution of training times between the two algorithms being statistically insignificant (Mann–Whitney–U = 44758, $n_1 = n_2 = 300$, $P > 0.05$ two tailed).

In addition, both algorithms exhibit the expected increase in training times as the number of features and samples in a dataset increases.

Table 6.6: Mean Training Time for Exact Algorithms (Synthetic)

ID	Features	Samples	Linear	KD Tree	Dual KD Tree
SYNTH01	2	1,000	0.000	0.000	0.000
SYNTH02	2	1,000	0.000	0.000	0.000
SYNTH03	10	1,000	0.000	0.000	0.001
SYNTH04	10	1,000	0.000	0.000	0.000
SYNTH05	100	1,000	0.000	0.003	0.005
SYNTH06	100	1,000	0.000	0.004	0.002
SYNTH07	1,000	1,000	0.000	0.026	0.023
SYNTH08	1,000	1,000	0.000	0.026	0.031
SYNTH09	2	10,000	0.000	0.004	0.005
SYNTH10	2	10,000	0.000	0.005	0.003
SYNTH11	10	10,000	0.000	0.006	0.007
SYNTH12	10	10,000	0.000	0.006	0.008
SYNTH13	100	10,000	0.000	0.062	0.058
SYNTH14	100	10,000	0.000	0.068	0.057
SYNTH15	1,000	10,000	0.000	0.474	0.534
SYNTH16	1,000	10,000	0.000	0.415	0.497
SYNTH17	2	100,000	0.000	0.059	0.057
SYNTH18	2	100,000	0.000	0.066	0.075
SYNTH19	10	100,000	0.000	0.148	0.172
SYNTH20	10	100,000	0.000	0.159	0.182
SYNTH21	100	100,000	0.000	0.931	1.183
SYNTH22	100	100,000	0.000	1.402	1.084
SYNTH23	1,000	100,000	0.000	12.699	13.359
SYNTH24	1,000	100,000	0.000	14.159	15.501
SYNTH25	2	1,000,000	0.000	2.622	2.442
SYNTH26	2	1,000,000	0.000	2.017	2.046
SYNTH27	10	1,000,000	0.000	2.732	3.165
SYNTH28	10	1,000,000	0.000	3.006	3.127
SYNTH29	100	1,000,000	0.000	25.089	24.611
SYNTH30	100	1,000,000	0.000	23.284	23.537

This table shows the training time in seconds for the Linear, KD–Tree and Dual KD–Tree algorithms.

Summary

In this section the Linear, KD–Tree and Dual KD–Tree algorithms have been shown to behave as expected. All three exact algorithms agree on the nearest neighbours for every dataset examined. The Linear algorithm has the slowest query time in every dataset. The KD–Tree algorithm has the fastest query time for very small numbers of features

but degrades to Linear performance as the number of features increases. In contrast the overhead incurred by the Dual KD–Tree for a small number of features makes it slower than the standard KD–Tree, however as the number of features increases, the benefits to query time of the Dual Tree approach are easily observed. With respect to training, the Linear algorithm requires no training step and therefore completes instantly and the KD–Tree and Dual KD–Tree algorithms have similar training times required to construct their trees. The training time for both tree based algorithms increases significantly with both the number of samples and number of features in the datasets.

6.4 Locality Sensitive Hashing (LSH) Results

In this Section the performance of Locality Sensitive Hashing (LSH) approximate nearest neighbour algorithm has been evaluated with respect to the real world and synthetic datasets detailed in Sections 6.2.3. LSH is widely used for high performance nearest neighbour search for datasets with both a large number of features and a large number of samples (Sundaram et al. 2013). Therefore the performance of this approximate algorithm forms the baseline for an approximate nearest neighbour algorithm to which the AURA methods will be compared.

For these experiments the number of hash tables used was fixed at 10 as suggested by (Bawa, Condie, and Ganesan 2005). Increasing the number of hash tables will increase the execution time of the algorithm and reduce the probability of a neighbour being excluded from the results. Reducing the number of hash tables will reduce the execution time and increase the probability of a neighbour being excluded. 10 hash tables is sufficient to ensure a good covering of the random bins and not so large as to impose a significant computational overhead.

Accuracy

Since the LSH algorithm is not an exact nearest neighbour algorithm, it is expected that the mean features distance for the LSH results will be greater than that of the exact algorithms. In these experiments, a mean features distance equal to the exact algorithm is considered to be a perfect result. A larger difference between the exact mean feature distance and the LSH mean feature distance indicates more error in the approximate results. In this Section, the results for both the synthetic and real world datasets are evaluated in comparison to the exact results.

Table 6.7: Mean Feature Distance for LSH Algorithm (Synthetic)

ID	Features	Samples	Exact	LSH
SYNTH01	2	1,000	0.06249	0.06250
SYNTH02	2	1,000	0.06456	0.06457
SYNTH03	10	1,000	0.14206	0.14371
SYNTH04	10	1,000	0.12374	0.12542
SYNTH05	100	1,000	0.07080	0.07250
SYNTH06	100	1,000	0.06061	0.06214
SYNTH07	1,000	1,000	0.02476	0.02476
SYNTH08	1,000	1,000	0.02136	0.02136
SYNTH09	2	10,000	0.01919	0.01919
SYNTH10	2	10,000	0.02229	0.02229
SYNTH11	10	10,000	0.10738	0.11075
SYNTH12	10	10,000	0.09582	0.09949
SYNTH13	100	10,000	0.06722	0.07120
SYNTH14	100	10,000	0.05752	0.06122
SYNTH15	1,000	10,000	0.02442	0.02479
SYNTH16	1,000	10,000	0.02101	0.02137
SYNTH17	2	100,000	0.00607	0.00607
SYNTH18	2	100,000	0.00725	0.00725
SYNTH19	10	100,000	0.08181	0.08596
SYNTH20	10	100,000	0.07569	0.08035
SYNTH21	100	100,000	0.06426	0.07023
SYNTH22	100	100,000	0.05485	0.06027
SYNTH23	1,000	100,000	0.02413	0.02476
SYNTH24	1,000	100,000	0.02075	0.02135
SYNTH25	2	1,000,000	0.00191	0.00191
SYNTH26	2	1,000,000	0.00238	0.00238
SYNTH27	10	1,000,000	0.06313	0.06730
SYNTH28	10	1,000,000	0.06011	0.06498
SYNTH29	100	1,000,000	0.06158	0.06920
SYNTH30	100	1,000,000	0.05268	0.05958

This table shows the mean feature distance for the LSH Algorithm, the mean feature distance for the Exact algorithms are provided for comparison.

Table 6.7 shows the mean feature distance for each of the synthetic datasets. It is clear that the difference between the LSH algorithm and the exact algorithms is very small. Across all the synthetic datasets the mean feature distance of the LSH algorithm is only 3.54% higher than the exact algorithms.

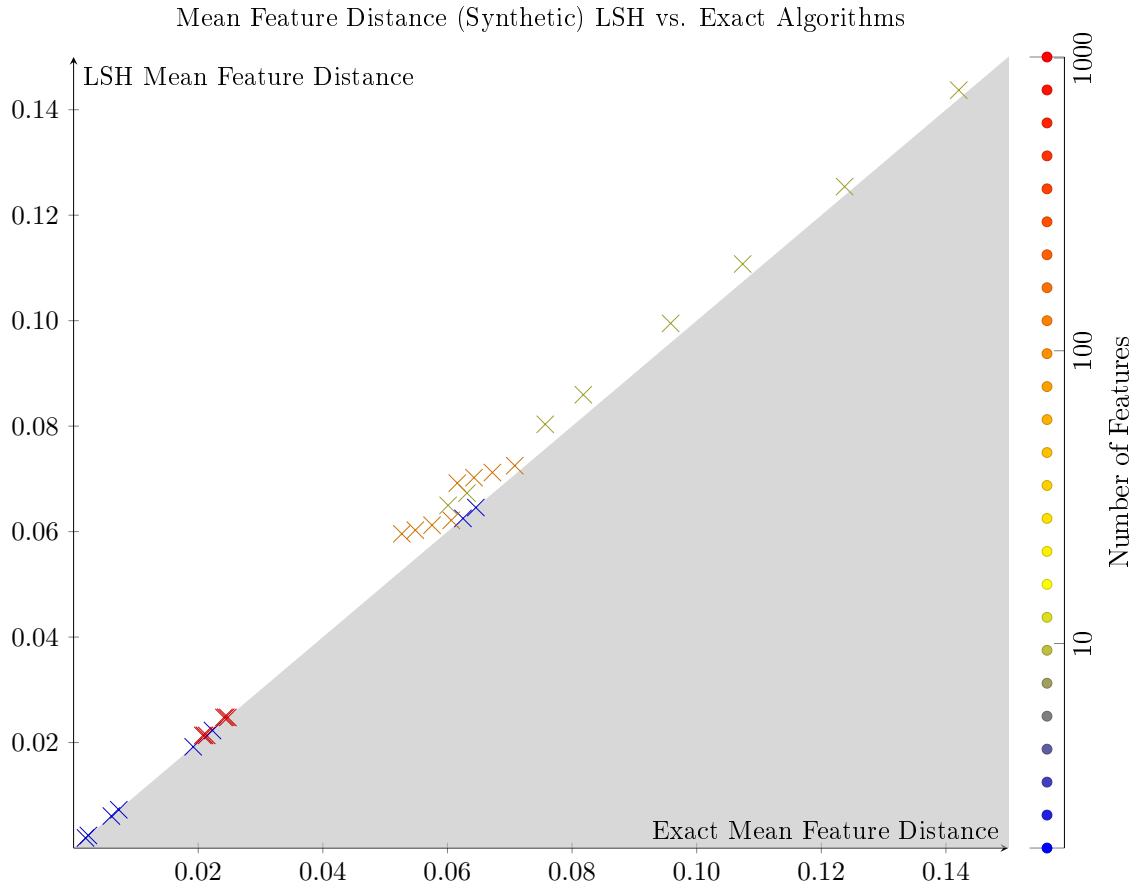


Figure 6.4: A comparison of the mean feature distance for each of the synthetic datasets. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A good result for LSH is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that LSH returns the same results as the exact algorithms.

However Figure 6.4 shows that the majority of the difference in mean feature distance is accounted for by the datasets with 10 or 100 features. The datasets with 2 or 1000 features are typically much closer to the results of the exact algorithms. Indeed the mean increase in feature distance for the 1000 feature datasets using the LSH algorithm is only 0.9%.

Table 6.8: Mean Feature Distance for LSH Algorithm (Real)

ID	Features	Samples	Exact	LSH
50words	270	450	0.04583	0.04601
Adiac	176	390	0.00543	0.00544
Beef	470	30	0.00233	0.00233
CBF	128	30	0.09348	0.09348
ChlorineConcentration	166	467	0.01130	0.01131
CinC_ECG_torso	1,639	40	0.03179	0.03179
Coffee	286	28	0.20785	0.20785
Cricket_X	300	390	0.04368	0.04381
Cricket_Y	300	390	0.04375	0.04389
Cricket_Z	300	390	0.04376	0.04391
DiatomSizeReduction	345	16	0.00983	0.00983
ECG200	96	100	0.05256	0.05256
ECGFiveDays	136	23	0.08129	0.08129
FaceAll	131	560	0.07213	0.07305
FaceFour	350	24	0.06316	0.06316
FacesUCR	131	200	0.07933	0.07948
Gun_Point	150	50	0.03413	0.03413
Haptics	1,092	155	0.01153	0.01153
InlineSkate	1,882	100	0.01281	0.01281
ItalyPowerDemand	24	67	0.07384	0.07384
Lighting2	637	60	0.03891	0.03891
Lighting7	319	70	0.05242	0.05242
MALLAT	1,024	55	0.00904	0.00904
MedicalImages	99	381	0.04450	0.04472
MoteStrain	84	20	0.10961	0.10961
NonInvasiveFatalECG_Thorax1	750	1,800	0.00431	0.00434
NonInvasiveFatalECG_Thorax2	750	1,800	0.00411	0.00414
OSULeaf	427	200	0.04397	0.04397
OliveOil	570	30	0.00018	0.00018
SonyAIBORobot_Surface	70	20	0.08912	0.08912
SonyAIBORobot_SurfaceII	65	27	0.13130	0.13130
StarLightCurves	1,024	1,000	0.00688	0.00688
SwedishLeaf	128	500	0.02878	0.02910
Symbols	398	25	0.05374	0.05374
Trace	275	100	0.02738	0.02738
TwoLeadECG	82	23	0.05481	0.05481
Two_Patterns	128	1,000	0.08252	0.08483
WordsSynonyms	270	267	0.05110	0.05110
fish	463	175	0.00862	0.00862
synthetic_control	60	300	0.10573	0.10627
uWaveGestureLibrary_X	315	896	0.03133	0.03182
uWaveGestureLibrary_Y	315	896	0.02756	0.02783
uWaveGestureLibrary_Z	315	896	0.02954	0.03001
wafer	152	1,000	0.02948	0.02978
yoga	426	300	0.01895	0.01895

This table shows the mean feature distance for the LSH Algorithm, the mean feature distance for the Exact algorithms are provided for comparison.

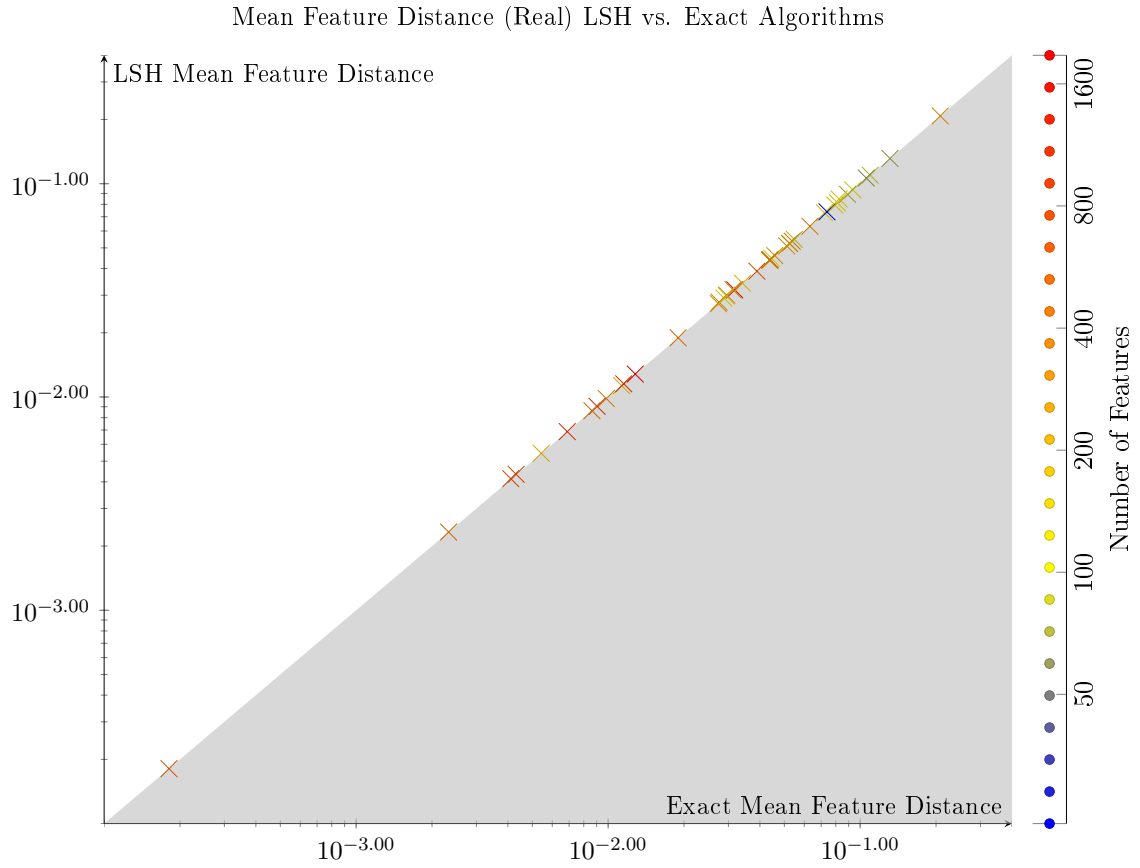


Figure 6.5: A comparison of the mean feature distance for each of the real world datasets. Each marker represents the mean feature distance for a single dataset, the markers are coloured according to the number of features in the dataset. A good result for LSH is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that LSH returns the same results as the exact algorithms.

The results for the real world datasets are given in Table 6.8 and plotted in Figure 6.5. These results are clearly superior to those of the synthetic datasets. Indeed, the mean increase in feature distance across all the real world datasets using the LSH algorithm is only 0.32%. Considering only the datasets with less than 500 features, this increase rises to 0.37%. In contrast the datasets with more than 500 features have an increase in mean feature distance of only 0.15%.

Despite being an approximate algorithm, it is clear that in general, accuracy of the LSH algorithm is very good both for the large and the small datasets. In both the real world dataset and the synthetic datasets, it appears that the accuracy of the LSH algorithm is superior for datasets with a large number of features and it is for this reason that LSH is commonly used for datasets consisting of many features (Sundaram et al. 2013).

Query Performance

In this Section, the query performance of the LSH algorithm will be evaluated with respect to the Dual KD–Tree algorithm. This is because the Dual KD–Tree algorithm was typically the fastest of the exact algorithms, particularly on the large datasets to which the LSH algorithm is best suited.

The mean query times for the LSH algorithm on the synthetic datasets can be found in Table 6.9. The query times for the real world datasets are not provided because the

datasets are not large enough to draw meaningful conclusions about the performance of the algorithms.

Table 6.9: Mean Query Time for the LSH Algorithm (Synthetic)

ID	Features	Samples	Dual KD Tree	LSH
SYNTH01	2	1,000	0.007	0.068
SYNTH02	2	1,000	0.013	0.075
SYNTH03	10	1,000	0.029	0.076
SYNTH04	10	1,000	0.026	0.073
SYNTH05	100	1,000	0.147	0.143
SYNTH06	100	1,000	0.166	0.137
SYNTH07	1,000	1,000	1.167	1.818
SYNTH08	1,000	1,000	1.307	1.973
SYNTH09	2	10,000	0.044	0.089
SYNTH10	2	10,000	0.052	0.086
SYNTH11	10	10,000	0.212	0.116
SYNTH12	10	10,000	0.218	0.124
SYNTH13	100	10,000	1.447	0.233
SYNTH14	100	10,000	1.359	0.241
SYNTH15	1,000	10,000	12.863	1.749
SYNTH16	1,000	10,000	12.216	1.889
SYNTH17	2	100,000	0.319	0.095
SYNTH18	2	100,000	0.324	0.108
SYNTH19	10	100,000	1.898	0.145
SYNTH20	10	100,000	2.105	0.165
SYNTH21	100	100,000	13.699	0.294
SYNTH22	100	100,000	13.688	0.290
SYNTH23	1,000	100,000	119.184	1.919
SYNTH24	1,000	100,000	130.298	1.857
SYNTH25	2	1,000,000	2.684	0.110
SYNTH26	2	1,000,000	3.093	0.123
SYNTH27	10	1,000,000	20.675	0.201
SYNTH28	10	1,000,000	19.216	0.190
SYNTH29	100	1,000,000	148.958	0.308
SYNTH30	100	1,000,000	140.706	0.300

This table shows the query time in seconds for the Locality Sensitive Hashing (LSH) algorithm.

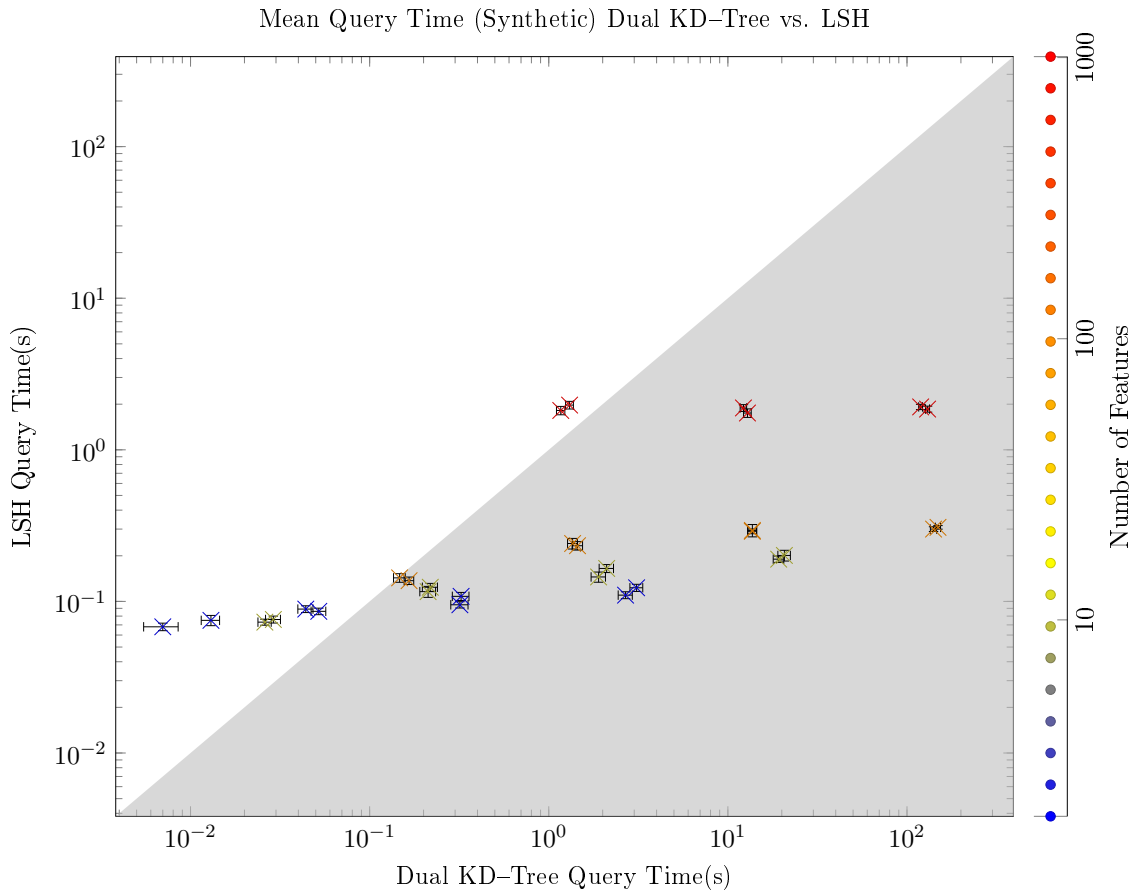


Figure 6.6: A comparison of the mean query time in seconds for each of the synthetic datasets. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A smaller query time is better. The shaded area indicates where the LSH performance is superior to the Dual KD–Tree algorithm.

Figure 6.6 shows a comparison between the mean query times of the LSH algorithm and the Dual KD–Tree algorithm. It is clear that the number of features in the dataset has a significant effect on the query time of the dataset. The total execution time of the 1000 feature datasets is 5.76 times longer than the total execution time of the 100 feature datasets.

However, it is also apparent from Table 6.9 that the query time does not change significantly as the number of samples increases. The mean query time only doubles as the number of samples in the 100 feature datasets increases from 1,000 to 1,000,000. This excellent ability to scale well with increasing dataset sized is expected and is one of the major reasons why LSH is widely used.

While Dual KD–Tree is faster than the LSH for the small datasets, SYNTH01–SYNTH09, that consist of only 1000 samples, LSH is substantially faster than the Dual KD–Tree algorithm for the larger datasets. For datasets SYNTH29 and SYNTH30, consisting of 1,000,000 samples, the LSH algorithm is faster by three orders of magnitude.

Training Performance

In this Section, the training performance of the LSH algorithm will be evaluated with respect to the Dual KD–Tree algorithm. This is to be consistent with the query time

comparison above. In addition the training times for KD–Tree and Dual KD-Tree were found to be very similar and a comparison with the non–existent training of the Linear algorithm is not useful.

Table 6.10: Mean Training Time for the LSH Algorithm (Synthetic)

ID	Features	Samples	Dual KD Tree	LSH
SYNTH01	2	1,000	0.000	0.032
SYNTH02	2	1,000	0.000	0.034
SYNTH03	10	1,000	0.001	0.055
SYNTH04	10	1,000	0.000	0.048
SYNTH05	100	1,000	0.005	0.191
SYNTH06	100	1,000	0.002	0.209
SYNTH07	1,000	1,000	0.023	0.365
SYNTH08	1,000	1,000	0.031	0.405
SYNTH09	2	10,000	0.005	0.351
SYNTH10	2	10,000	0.003	0.331
SYNTH11	10	10,000	0.007	0.655
SYNTH12	10	10,000	0.008	0.734
SYNTH13	100	10,000	0.058	3.762
SYNTH14	100	10,000	0.057	3.932
SYNTH15	1,000	10,000	0.534	20.380
SYNTH16	1,000	10,000	0.497	23.184
SYNTH17	2	100,000	0.057	3.692
SYNTH18	2	100,000	0.075	4.200
SYNTH19	10	100,000	0.172	8.848
SYNTH20	10	100,000	0.182	10.319
SYNTH21	100	100,000	1.183	60.299
SYNTH22	100	100,000	1.084	66.571
SYNTH23	1,000	100,000	13.359	387.808
SYNTH24	1,000	100,000	15.501	398.625
SYNTH25	2	1,000,000	2.442	49.638
SYNTH26	2	1,000,000	2.046	57.752
SYNTH27	10	1,000,000	3.165	176.161
SYNTH28	10	1,000,000	3.127	188.207
SYNTH29	100	1,000,000	24.611	1,055.715
SYNTH30	100	1,000,000	23.537	1,042.559

This table shows the training time in seconds for the Locality Sensitive Hashing (LSH) algorithm.

The mean training times for each of the synthetic datasets are provided in Table 6.10. The LSH training times are substantially longer than for the Dual KD–Tree algorithm. In all cases it takes several orders of magnitude longer to perform the training for the LSH. In addition Figures 6.7 and 6.8 appear to show the training time increasing linearly with both the number of samples in the dataset and the number of features in each sample.

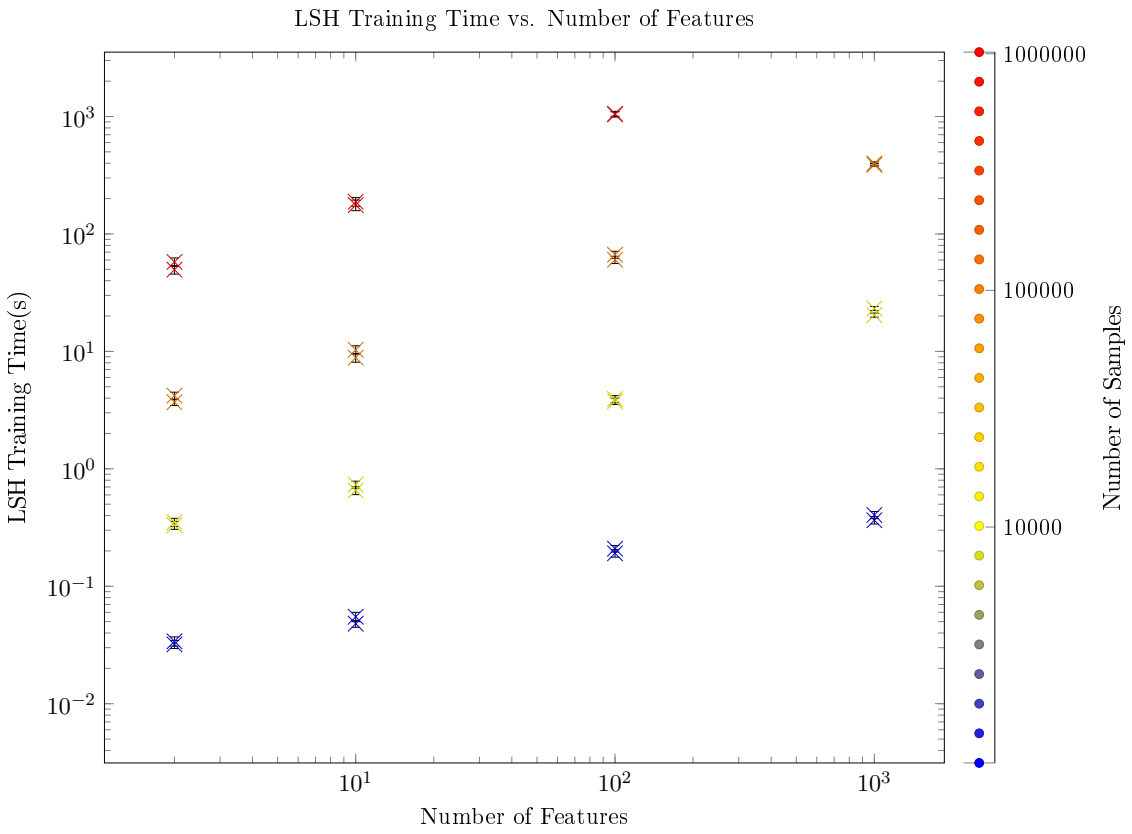


Figure 6.7: A comparison of the mean training time in seconds with the number of features in a dataset. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of samples in the dataset.

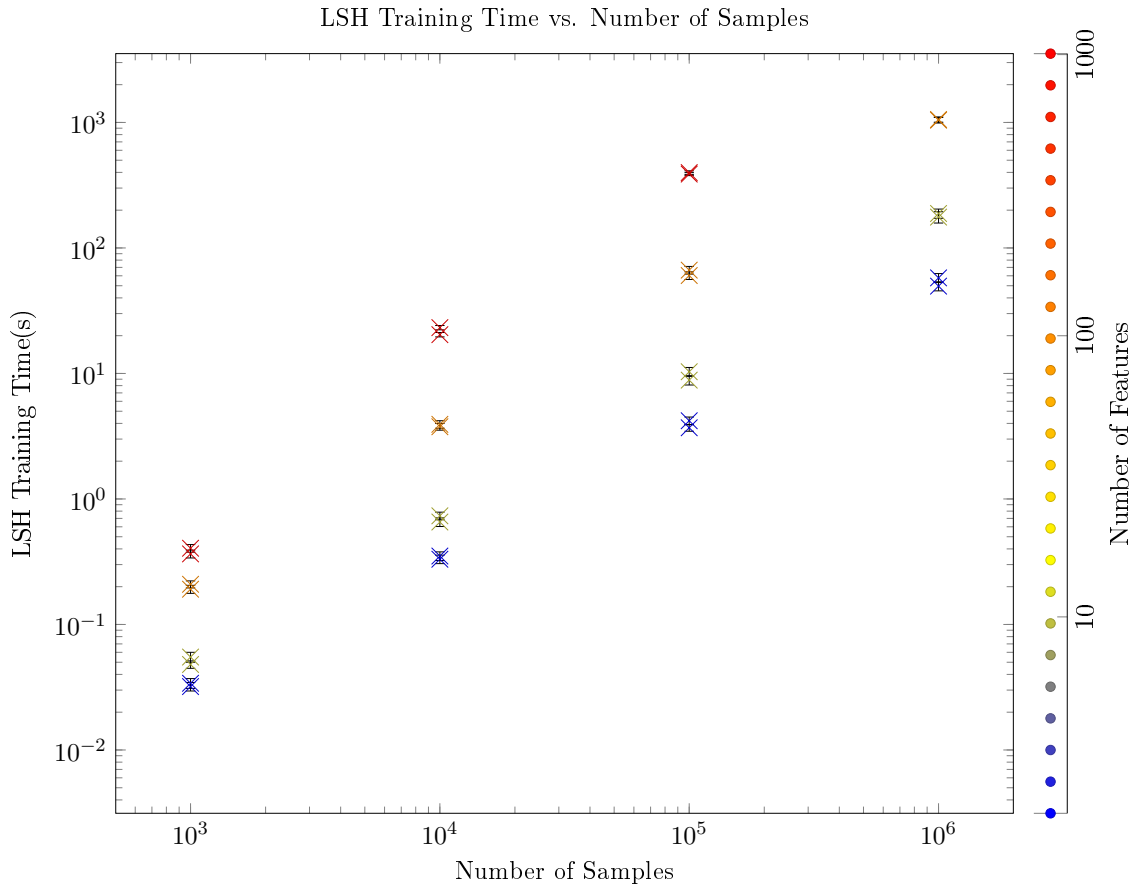


Figure 6.8: A comparison of the mean training time in seconds with the number of samples in a dataset. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset.

Summary

In this Section, the LSH algorithm has been evaluated with respect to the exact Dual KD–Tree algorithm. The accuracy of the LSH algorithm is very similar to that of the exact algorithm. On the real world datasets the mean feature distance for the LSH algorithm is only 0.32% larger than the results for the exact algorithms. It is likely that this level of accuracy would be sufficient for most applications of AURA Alert.

The query time of LSH is the primary advantage of the algorithm. Within the range of datasets used for this experiment, there does not appear to be a significant increase in the query time for datasets with many samples in comparison to those with few samples. As a result, while the LSH algorithm may not be the best algorithm for small datasets, i.e. those with fewer than 10,000 samples, it is several orders of magnitude faster than any of the exact algorithms for the largest datasets.

This major advantage in query speed comes at the cost of the training time. In comparison to Dual KD–Tree, LSH requires considerably longer to train. This training time appears to grow linearly with both the number of samples and the number of features in the dataset. As a result, in the worst case, LSH required a mean training time of 1056 seconds to train dataset SYNTH29 consisting of 1,000,000 samples and 100 features while the Dual KD–Tree required only 25 seconds.

In conclusion, the LSH algorithm would be well suited to applications that require

frequent querying of very large datasets where this would enable the higher cost of the training to be out weighed by the significantly faster query times.

Chapter 7

AURA Unary Output Token Experiments

In this Chapter the AURA kNN algorithm is investigated using a unary output token. The purpose of these experiments is to determine the best input token method to use with AURA kNN and to evaluate how the AURA kNN compares to the baseline algorithms examined in Section 6.3.

The Parabolic Kernel (PK) input token with a unary output token is currently the standard configuration of AURA kNN used by AURA Alert. However this configuration has previously only been compared to the Linear Scan algorithm (Hodge and Austin 2005) and it is unknown how this configuration will preform in comparison to faster exact algorithms such as Dual KD–Tree or the approximate LSH algorithm. This is the first comparison to be published that considers either the OBCC or WOCC input tokens.

In Section 7.1 the effect of varying the number of bins used to encode the input tokens is considered. This is followed by Section 7.2 where the performance of the three input token methods is compared with the baseline algorithms.

7.1 Number of Bins

Section 5.2.1.2 demonstrated how increasing the number of bins adversely affect the length and weight of OBCC tokens. However using more bins allows tokens to be generated that provide a greater level of discrimination between each token.

The purpose of this Section is to determine whether there are any improvements in the accuracy of the AURA kNN algorithm by using more bins and whether it is worth the performance cost caused by the resulting larger input tokens.

Accuracy

In this Section the effect on the accuracy of increasing the number of bins is investigated for each of the three input token methods, PK, OBCC and WOCC. The table of results for these experiments on the synthetic datasets is given below, the results on the real world datasets can be found in Appendix B.

Table 7.1: Mean Feature Distance for Number of Bins with PK AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.062	0.062	0.062	0.063	0.063
SYNTH02	2	1,000	0.065	0.065	0.065	0.065	0.065
SYNTH03	10	1,000	0.142	0.149	0.149	0.149	0.149
SYNTH04	10	1,000	0.124	0.128	0.128	0.128	0.128
SYNTH05	100	1,000	0.071	0.072	0.072	0.072	0.072
SYNTH06	100	1,000	0.061	0.062	0.062	0.062	0.062
SYNTH07	1,000	1,000	0.025	0.025	0.025	0.025	0.025
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.020	0.019	0.019	0.019
SYNTH10	2	10,000	0.022	0.023	0.022	0.022	0.023
SYNTH11	10	10,000	0.107	0.110	0.110	0.110	0.110
SYNTH12	10	10,000	0.096	0.099	0.098	0.097	0.097
SYNTH13	100	10,000	0.067	0.069	0.069	0.069	0.069
SYNTH14	100	10,000	0.058	0.059	0.059	0.059	0.059
SYNTH15	1,000	10,000	0.024	0.025	0.025	0.025	0.025
SYNTH16	1,000	10,000	0.021	0.021	0.021	0.021	0.021
SYNTH17	2	100,000	0.006	0.006	0.006	0.006	0.006
SYNTH18	2	100,000	0.007	0.008	0.008	0.008	0.007
SYNTH19	10	100,000	0.082	0.084	0.082	0.082	0.082
SYNTH20	10	100,000	0.076	0.080	0.078	0.078	0.077
SYNTH21	100	100,000	0.064	0.066	0.066	0.066	0.066
SYNTH22	100	100,000	0.055	0.057	0.057	0.057	0.057
SYNTH23	1,000	100,000	0.024	0.024	0.024	0.024	0.024
SYNTH24	1,000	100,000	0.021	0.021	0.021	0.021	0.021
SYNTH25	2	1,000,000	0.002	0.002	0.002	0.002	0.002
SYNTH26	2	1,000,000	0.002	0.002	0.002	0.003	0.003
SYNTH27	10	1,000,000	0.063	0.066	0.064	0.063	0.063
SYNTH28	10	1,000,000	0.060	0.065	0.063	0.062	0.062
SYNTH29	100	1,000,000	0.062	0.064	0.064	0.064	0.064
SYNTH30	100	1,000,000	0.053	0.055	0.055	0.055	0.055

This table shows the mean feature distance for the AURA algorithm with PK input tokens, Unary output tokens and a varying numbers of bins.

Table 7.1 gives the results for the AURA Nearest Neighbours algorithm using the PK input tokens and Unary output tokens. This is the configuration that has traditionally been used as part of AURA Alert. It is expected that increasing the number of bins will correspond to a decrease in the mean feature distances observed. While it appears that this is the case from these results, because the mean feature distance has a small decrease as the number of bins increases, there is no statistically significant difference between the observed feature distances for 10 bins and for 80 bins (One–Way ANOVA, $f = 0.012$, $p = 0.998 > 0.05$) despite the large increase in the number of bins. The real world datasets also did not display any statistically significant differences (One–Way ANOVA, $f = 0.003$, $p = 1.000 > 0.05$) between the 10 bin results and the 80 bin results.

Table 7.2: Mean Feature Distance for Number of Bins with OBCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.062	0.062	0.063	0.096	0.226
SYNTH02	2	1,000	0.065	0.065	0.065	0.090	0.177
SYNTH03	10	1,000	0.142	0.181	0.190	0.197	0.201
SYNTH04	10	1,000	0.124	0.159	0.166	0.172	0.175
SYNTH05	100	1,000	0.071	0.075	0.076	0.077	0.077
SYNTH06	100	1,000	0.061	0.066	0.066	0.067	0.067
SYNTH07	1,000	1,000	0.025	0.025	0.025	0.025	0.025
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.020	0.019	0.019	0.030
SYNTH10	2	10,000	0.022	0.023	0.022	0.023	0.031
SYNTH11	10	10,000	0.107	0.155	0.165	0.177	0.183
SYNTH12	10	10,000	0.096	0.134	0.144	0.154	0.158
SYNTH13	100	10,000	0.067	0.073	0.074	0.075	0.075
SYNTH14	100	10,000	0.058	0.064	0.065	0.065	0.066
SYNTH15	1,000	10,000	0.024	0.025	0.025	0.025	0.025
SYNTH16	1,000	10,000	0.021	0.022	0.022	0.022	0.022
SYNTH17	2	100,000	0.006	0.006	0.006	0.006	0.006
SYNTH18	2	100,000	0.007	0.008	0.008	0.008	0.008
SYNTH19	10	100,000	0.082	0.126	0.142	0.157	0.166
SYNTH20	10	100,000	0.076	0.111	0.125	0.138	0.145
SYNTH21	100	100,000	0.064	0.072	0.073	0.074	0.074
SYNTH22	100	100,000	0.055	0.062	0.063	0.064	0.065
SYNTH23	1,000	100,000	0.024	0.025	0.025	0.025	0.025
SYNTH24	1,000	100,000	0.021	0.022	0.022	0.022	0.022
SYNTH25	2	1,000,000	0.002	0.002	0.002	0.002	0.002
SYNTH26	2	1,000,000	0.002	0.002	0.002	0.003	0.003
SYNTH27	10	1,000,000	0.063	0.098	0.120	0.140	0.151
SYNTH28	10	1,000,000	0.060	0.089	0.106	0.123	0.132
SYNTH29	100	1,000,000	0.062	0.070	0.071	0.072	0.073
SYNTH30	100	1,000,000	0.053	0.061	0.062	0.063	0.064

This table shows the mean feature distance for the AURA algorithm with OBCC input tokens, unary output tokens and a varying numbers of bins.

Table 7.2 gives the results for the AURA Nearest Neighbours algorithm using the OBCC input tokens and unary output tokens. Here we can see that, counter–intuitively, there is a significant increase in the mean feature distance for OBCC tokens with more bins (One–Way ANOVA, $f = 6.17$, $p = 0.000 < 0.05$). A potential reason for this result could be due to the limits placed on the maximum allowed overlap between input tokens during the generation of the OBCC tokens that is required to ensure that the tokens remain a manageable length. Within the real world datasets, this degradation in accuracy is not observed, the accuracy on these datasets did not display any statistically significant differences (One–Way ANOVA, $f = 0.006$, $p = 0.999 > 0.05$) as the number of bins increased.

Table 7.3: Mean Feature Distance for Number of Bins with WOCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.062	0.062	0.063	0.096	0.226
SYNTH02	2	1,000	0.065	0.065	0.065	0.090	0.177
SYNTH03	10	1,000	0.142	0.181	0.190	0.197	0.201
SYNTH04	10	1,000	0.124	0.159	0.166	0.172	0.175
SYNTH05	100	1,000	0.071	0.075	0.076	0.077	0.077
SYNTH06	100	1,000	0.061	0.066	0.066	0.067	0.067
SYNTH07	1,000	1,000	0.025	0.025	0.025	0.025	0.025
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.020	0.019	0.019	0.030
SYNTH10	2	10,000	0.022	0.023	0.022	0.023	0.031
SYNTH11	10	10,000	0.107	0.155	0.165	0.177	0.183
SYNTH12	10	10,000	0.096	0.134	0.144	0.154	0.158
SYNTH13	100	10,000	0.067	0.073	0.074	0.075	0.075
SYNTH14	100	10,000	0.058	0.064	0.065	0.065	0.066
SYNTH15	1,000	10,000	0.024	0.025	0.025	0.025	0.025
SYNTH16	1,000	10,000	0.021	0.022	0.022	0.022	0.022
SYNTH17	2	100,000	0.006	0.006	0.006	0.006	0.006
SYNTH18	2	100,000	0.007	0.008	0.008	0.008	0.008
SYNTH19	10	100,000	0.082	0.126	0.142	0.157	0.166
SYNTH20	10	100,000	0.076	0.111	0.125	0.138	0.145
SYNTH21	100	100,000	0.064	0.072	0.073	0.074	0.074
SYNTH22	100	100,000	0.055	0.062	0.063	0.064	0.065
SYNTH23	1,000	100,000	0.024	0.025	0.025	0.025	0.025
SYNTH24	1,000	100,000	0.021	0.022	0.022	0.022	0.022
SYNTH25	2	1,000,000	0.002	0.002	0.002	0.002	0.002
SYNTH26	2	1,000,000	0.002	0.002	0.002	0.003	0.003
SYNTH27	10	1,000,000	0.063	0.098	0.120	0.140	0.151
SYNTH28	10	1,000,000	0.060	0.089	0.106	0.123	0.132
SYNTH29	100	1,000,000	0.062	0.070	0.071	0.072	0.073
SYNTH30	100	1,000,000	0.053	0.061	0.062	0.063	0.064

This table shows the mean feature distance for the AURA algorithm with WOCC input tokens, unary output tokens and a varying numbers of bins.

Table 7.3 gives the results for the AURA Nearest Neighbours algorithm using the WOCC input tokens and unary output tokens. As expected, the accuracy of these tokens is identical to that of the OBCC tokens and similarly, there is a significant increase in the mean feature distance as the number of bins increases (One–Way ANOVA, $f = 6.17$, $p = 0.000 < 0.05$). As with the OBCC tokens, this degradation in accuracy is not observed within the real world datasets, the accuracy on these datasets did not display any statistically significant differences (One–Way ANOVA, $f = 0.006$, $p = 0.999 > 0.05$) as the number of bins increased.

Query Performance

This Section investigates the effect on query time of increasing the number of bins for each of the three input token methods, PK, OBCC and WOCC. The results of these experiments are given below.

Table 7.4: Mean Query Time for Number of Bins with PK AURA (Synthetic)

ID	Features	Samples	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.096	0.085	0.099	0.106
SYNTH02	2	1,000	0.097	0.088	0.095	0.110
SYNTH03	10	1,000	0.127	0.119	0.125	0.124
SYNTH04	10	1,000	0.112	0.119	0.129	0.129
SYNTH05	100	1,000	0.210	0.219	0.247	0.291
SYNTH06	100	1,000	0.216	0.235	0.253	0.302
SYNTH07	1,000	1,000	1.142	1.199	1.421	1.878
SYNTH08	1,000	1,000	1.314	1.299	1.550	2.056
SYNTH09	2	10,000	0.135	0.118	0.160	0.340
SYNTH10	2	10,000	0.118	0.119	0.158	0.310
SYNTH11	10	10,000	0.656	0.692	0.758	0.787
SYNTH12	10	10,000	0.621	0.714	0.801	0.816
SYNTH13	100	10,000	1.459	1.530	1.542	1.584
SYNTH14	100	10,000	1.561	1.621	1.688	1.708
SYNTH15	1,000	10,000	12.455	10.752	10.393	10.684
SYNTH16	1,000	10,000	12.900	11.925	11.914	11.747
SYNTH17	2	100,000	0.390	0.216	0.219	0.472
SYNTH18	2	100,000	0.465	0.247	0.230	0.466
SYNTH19	10	100,000	6.636	7.383	8.605	9.331
SYNTH20	10	100,000	6.071	7.340	8.728	9.455
SYNTH21	100	100,000	17.344	20.662	24.509	26.250
SYNTH22	100	100,000	18.543	21.676	25.869	27.797
SYNTH23	1,000	100,000	119.093	144.322	171.968	200.853
SYNTH24	1,000	100,000	119.534	159.917	190.666	195.135
SYNTH25	2	1,000,000	4.385	1.181	0.412	0.518
SYNTH26	2	1,000,000	6.132	1.555	0.546	0.559
SYNTH27	10	1,000,000	65.392	79.352	94.646	99.622
SYNTH28	10	1,000,000	66.379	80.838	90.038	105.464
SYNTH29	100	1,000,000	178.781	213.471	241.205	281.756
SYNTH30	100	1,000,000	182.917	219.228	262.427	297.216

This table shows the mean query time in seconds for the AURA algorithm with PK input tokens, unary output tokens and a varying numbers of bins.

Table 7.4 gives the results for the AURA Nearest Neighbours algorithm using PK input tokens and unary output tokens. This is the configuration that has traditionally been used as part of AURA Alert. It appears that the results are generally as expected. Increasing bins yields an increased query time (One–Way ANOVA, $f = 3.26$, $p = 0.021 < 0.05$). Datasets SYNTH25 and SYNTH26 are clearly outliers to this trend, if they are excluded, the mean query time increases by 24.7% as the number of bins increases from 10 to 80. However for datasets SYNTH25 and SYNTH26 with only 2 features and 1,000,000 samples fewer bins does not result in faster query times but rather a 90% reduction in query time.

Table 7.5: Mean Query Time for Number of Bins with OBCC AURA (Synthetic)

ID	Features	Samples	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.101	0.094	0.098	0.088
SYNTH02	2	1,000	0.103	0.095	0.099	0.090
SYNTH03	10	1,000	0.156	0.126	0.133	0.129
SYNTH04	10	1,000	0.167	0.135	0.137	0.132
SYNTH05	100	1,000	0.688	0.449	0.405	0.406
SYNTH06	100	1,000	0.862	0.512	0.431	0.434
SYNTH07	1,000	1,000	6.047	4.932	5.034	5.758
SYNTH08	1,000	1,000	8.401	5.585	5.384	5.688
SYNTH09	2	10,000	0.116	0.118	0.140	0.197
SYNTH10	2	10,000	0.119	0.122	0.138	0.193
SYNTH11	10	10,000	0.989	0.737	0.736	0.670
SYNTH12	10	10,000	1.168	0.811	0.764	0.718
SYNTH13	100	10,000	5.923	3.752	3.129	2.755
SYNTH14	100	10,000	6.524	4.234	3.489	3.125
SYNTH15	1,000	10,000	50.060	32.044	25.711	23.236
SYNTH16	1,000	10,000	60.062	37.869	30.182	26.617
SYNTH17	2	100,000	0.405	0.197	0.191	0.292
SYNTH18	2	100,000	0.505	0.231	0.205	0.275
SYNTH19	10	100,000	9.878	7.464	7.283	6.595
SYNTH20	10	100,000	12.101	8.151	7.631	6.740
SYNTH21	100	100,000	52.479	36.954	31.454	28.430
SYNTH22	100	100,000	62.582	42.282	35.477	31.922
SYNTH23	1,000	100,000	481.428	320.311	269.623	249.017
SYNTH24	1,000	100,000	572.691	377.277	310.774	284.489
SYNTH25	2	1,000,000	4.679	1.102	0.397	0.335
SYNTH26	2	1,000,000	6.034	1.509	0.521	0.355
SYNTH27	10	1,000,000	99.923	80.627	74.732	70.248
SYNTH28	10	1,000,000	107.512	85.985	80.636	76.428
SYNTH29	100	1,000,000	926.413	637.902	580.114	626.345
SYNTH30	100	1,000,000	1,185.169	756.547	673.358	742.845

This table shows the mean query time in seconds for the AURA algorithm with OBCC input tokens, unary output tokens and a varying numbers of bins.

Table 7.5 gives the results for the AURA Nearest Neighbours algorithm using OBCC input tokens and unary output tokens. Once again, it appears that OBCC tokens are not performing as predicted. In this case increasing bins yields a significantly reduced query time (One–Way ANOVA, $f = 3.87$, $p = 0.009 < 0.05$), the mean query time is reduced by 33% as the number of bins increases from 10 bins to 80 bins. This is opposite to the expected outcome.

Table 7.6: Mean Query Time for Number of Bins with WOCC AURA (Synthetic)

ID	Features	Samples	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.096	0.095	0.092	0.082
SYNTH02	2	1,000	0.118	0.094	0.100	0.093
SYNTH03	10	1,000	0.129	0.121	0.126	0.117
SYNTH04	10	1,000	0.145	0.148	0.132	0.129
SYNTH05	100	1,000	0.300	0.287	0.362	0.378
SYNTH06	100	1,000	0.342	0.371	0.412	0.415
SYNTH07	1,000	1,000	2.588	2.894	4.180	4.934
SYNTH08	1,000	1,000	2.828	3.477	5.198	5.302
SYNTH09	2	10,000	0.117	0.116	0.129	0.190
SYNTH10	2	10,000	0.132	0.126	0.137	0.189
SYNTH11	10	10,000	0.636	0.637	0.680	0.649
SYNTH12	10	10,000	0.804	0.705	0.747	0.698
SYNTH13	100	10,000	2.066	2.080	2.791	2.568
SYNTH14	100	10,000	2.709	2.794	3.322	3.107
SYNTH15	1,000	10,000	15.633	15.522	23.168	22.307
SYNTH16	1,000	10,000	18.951	20.288	27.545	25.733
SYNTH17	2	100,000	0.572	0.221	0.189	0.268
SYNTH18	2	100,000	0.453	0.254	0.208	0.268
SYNTH19	10	100,000	6.218	5.982	6.600	6.449
SYNTH20	10	100,000	7.956	6.703	7.242	6.897
SYNTH21	100	100,000	19.127	21.437	27.966	27.931
SYNTH22	100	100,000	24.743	23.878	32.857	31.281
SYNTH23	1,000	100,000	147.735	155.731	234.371	226.136
SYNTH24	1,000	100,000	178.000	177.837	282.805	274.388
SYNTH25	2	1,000,000	4.194	1.254	0.374	0.341
SYNTH26	2	1,000,000	5.274	1.599	0.518	0.347
SYNTH27	10	1,000,000	73.649	64.859	71.481	66.555
SYNTH28	10	1,000,000	76.556	73.847	75.691	75.080
SYNTH29	100	1,000,000	207.231	213.543	441.144	523.540
SYNTH30	100	1,000,000	251.387	242.896	537.618	657.671

This table shows the mean query time in seconds for the AURA algorithm with WOCC input tokens, unary output tokens and a varying numbers of bins.

Table 7.6 gives the results for the AURA Nearest Neighbours algorithm using WOCC input tokens and unary output tokens. As with the PK tokens, increasing the number of bins also leads to a significant increase in the query time (One–Way ANOVA, $f = 6.49$, $p = 0.00 < 0.05$) for the majority of datasets. Excluding the 2 feature datasets, the 80 bin query times are 18% slower than the 10 bin query times. As with the PK tokens, the two feature datasets, particularly those with many samples such as SYNTH25 and SYNTH26, can be queried a mean of 25% faster with 80 bins rather than 10 bins.

Training Performance

This Section investigates the effect on the training time of increasing the number of bins for the PK, OBCC and WOCC input tokens. The results of these experiments are given below.

Table 7.7: Mean Training Time for Number of Bins with PK AURA (Synthetic)

ID	Features	Samples	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.000	0.001	0.000	0.002
SYNTH02	2	1,000	0.000	0.001	0.002	0.001
SYNTH03	10	1,000	0.004	0.003	0.002	0.003
SYNTH04	10	1,000	0.003	0.004	0.003	0.005
SYNTH05	100	1,000	0.012	0.015	0.018	0.022
SYNTH06	100	1,000	0.011	0.014	0.016	0.022
SYNTH07	1,000	1,000	0.119	0.148	0.181	0.231
SYNTH08	1,000	1,000	0.128	0.141	0.178	0.209
SYNTH09	2	10,000	0.004	0.005	0.007	0.011
SYNTH10	2	10,000	0.007	0.003	0.009	0.010
SYNTH11	10	10,000	0.024	0.027	0.029	0.031
SYNTH12	10	10,000	0.024	0.025	0.029	0.033
SYNTH13	100	10,000	0.129	0.157	0.183	0.212
SYNTH14	100	10,000	0.123	0.149	0.177	0.205
SYNTH15	1,000	10,000	1.330	1.520	1.851	2.364
SYNTH16	1,000	10,000	1.249	1.392	1.852	2.306
SYNTH17	2	100,000	0.042	0.048	0.055	0.067
SYNTH18	2	100,000	0.043	0.049	0.056	0.064
SYNTH19	10	100,000	0.295	0.283	0.322	0.325
SYNTH20	10	100,000	0.255	0.272	0.316	0.330
SYNTH21	100	100,000	1.353	1.577	1.883	2.156
SYNTH22	100	100,000	1.327	1.484	1.824	2.094
SYNTH23	1,000	100,000	12.703	14.823	19.147	23.814
SYNTH24	1,000	100,000	11.536	14.452	18.345	21.192
SYNTH25	2	1,000,000	0.455	0.494	0.569	0.676
SYNTH26	2	1,000,000	0.529	0.490	0.577	0.643
SYNTH27	10	1,000,000	2.668	3.096	3.452	3.455
SYNTH28	10	1,000,000	2.728	2.918	3.066	3.509
SYNTH29	100	1,000,000	15.489	15.786	18.477	21.358
SYNTH30	100	1,000,000	13.033	15.136	18.023	20.388

This table shows the mean training time in seconds for the AURA algorithm with PK input tokens, unary output tokens and a varying numbers of bins.

Table 7.7 gives the results for the AURA Nearest Neighbours algorithm using PK input tokens and unary output tokens. This is the configuration that has traditionally been used as part of AURA Alert. As with the query time, the training time generally increases as the number of bins increases. With the mean training time increasing by 38% between the 10 bin and 80 bin configurations. This is a statistically significant increase (One–Way ANOVA, $f = 3.374$, $p = 0.018 < 0.05$) and is consistent regardless of the number of features in the dataset.

Table 7.8: Mean Training Time for Number of Bins with OBCC AURA (Synthetic)

ID	Features	Samples	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.000	0.001	0.001	0.004
SYNTH02	2	1,000	0.001	0.000	0.001	0.005
SYNTH03	10	1,000	0.008	0.010	0.012	0.016
SYNTH04	10	1,000	0.009	0.009	0.014	0.016
SYNTH05	100	1,000	0.087	0.092	0.131	0.176
SYNTH06	100	1,000	0.097	0.093	0.130	0.182
SYNTH07	1,000	1,000	0.896	1.329	1.941	2.729
SYNTH08	1,000	1,000	1.128	1.105	1.820	2.418
SYNTH09	2	10,000	0.005	0.006	0.010	0.025
SYNTH10	2	10,000	0.005	0.005	0.010	0.024
SYNTH11	10	10,000	0.106	0.104	0.146	0.177
SYNTH12	10	10,000	0.114	0.104	0.142	0.177
SYNTH13	100	10,000	1.047	1.009	1.433	1.936
SYNTH14	100	10,000	1.000	0.998	1.362	1.899
SYNTH15	1,000	10,000	11.382	13.847	19.515	27.868
SYNTH16	1,000	10,000	12.095	13.953	20.111	26.812
SYNTH17	2	100,000	0.043	0.048	0.058	0.085
SYNTH18	2	100,000	0.045	0.049	0.059	0.085
SYNTH19	10	100,000	1.089	1.099	1.502	1.867
SYNTH20	10	100,000	1.233	1.073	1.489	1.796
SYNTH21	100	100,000	10.019	10.455	15.085	20.508
SYNTH22	100	100,000	10.317	10.520	14.497	20.086
SYNTH23	1,000	100,000	115.619	136.090	202.406	276.971
SYNTH24	1,000	100,000	116.010	136.788	208.260	280.658
SYNTH25	2	1,000,000	0.515	0.455	0.576	0.628
SYNTH26	2	1,000,000	0.452	0.480	0.589	0.642
SYNTH27	10	1,000,000	10.873	11.316	15.014	18.655
SYNTH28	10	1,000,000	10.740	11.014	14.730	18.399
SYNTH29	100	1,000,000	82.237	87.665	118.900	157.882
SYNTH30	100	1,000,000	91.697	90.478	117.299	148.335

This table shows the mean training time in seconds for the AURA algorithm with OBCC input tokens, unary output tokens and a varying numbers of bins.

Table 7.8 gives the results for the AURA Nearest Neighbours algorithm using OBCC input tokens and unary output tokens. As expected, the training time has a significant increase (One–Way ANOVA, $f = 6.551$, $p = 0.000 < 0.05$) as the number of bins increases. The 10 bin training time is 52% faster than the 80 bin training time.

Table 7.9: Mean Training Time for Number of Bins with WOCC AURA (Synthetic)

ID	Features	Samples	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.001	0.000	0.005	0.005
SYNTH02	2	1,000	0.001	0.000	0.000	0.004
SYNTH03	10	1,000	0.005	0.005	0.010	0.015
SYNTH04	10	1,000	0.005	0.010	0.011	0.016
SYNTH05	100	1,000	0.036	0.052	0.092	0.148
SYNTH06	100	1,000	0.036	0.068	0.095	0.147
SYNTH07	1,000	1,000	0.393	0.745	1.316	2.038
SYNTH08	1,000	1,000	0.405	0.664	1.297	2.008
SYNTH09	2	10,000	0.002	0.006	0.010	0.024
SYNTH10	2	10,000	0.003	0.002	0.009	0.019
SYNTH11	10	10,000	0.041	0.067	0.100	0.148
SYNTH12	10	10,000	0.051	0.065	0.104	0.147
SYNTH13	100	10,000	0.333	0.558	0.996	1.444
SYNTH14	100	10,000	0.379	0.595	0.991	1.545
SYNTH15	1,000	10,000	3.836	6.650	13.654	23.185
SYNTH16	1,000	10,000	3.644	7.290	14.144	21.512
SYNTH17	2	100,000	0.062	0.051	0.056	0.079
SYNTH18	2	100,000	0.042	0.048	0.059	0.077
SYNTH19	10	100,000	0.476	0.663	1.059	1.548
SYNTH20	10	100,000	0.493	0.727	1.095	1.533
SYNTH21	100	100,000	3.204	5.993	10.416	16.269
SYNTH22	100	100,000	3.905	6.018	10.471	16.101
SYNTH23	1,000	100,000	37.234	73.642	145.850	225.989
SYNTH24	1,000	100,000	37.478	68.502	156.347	232.910
SYNTH25	2	1,000,000	0.559	0.514	0.549	0.683
SYNTH26	2	1,000,000	0.472	0.482	0.560	0.650
SYNTH27	10	1,000,000	4.839	7.036	10.825	15.194
SYNTH28	10	1,000,000	5.237	7.479	10.855	15.216
SYNTH29	100	1,000,000	33.303	56.740	93.720	130.491
SYNTH30	100	1,000,000	37.722	57.860	93.955	126.915

This table shows the mean training time in seconds for the AURA algorithm with WOCC input tokens, unary output tokens and a varying numbers of bins.

Table 7.9 gives the results for the AURA Nearest Neighbours algorithm using WOCC input tokens and unary output tokens. As with the OBCC tokens, the training time is 60% faster with 10 bins in comparison to 80 bins. This advantage is statistically significant (One–Way ANOVA, $f = 17.552$, $p = 0.000 < 0.05$).

Evaluation

In this Section, the effect of varying the number of bins used when generating the input tokens for the CMM has been examined.

Surprisingly, with PK input tokens, increasing the number of bins did not have a significant effect on the accuracy of the results for either the synthetic or real world datasets (Appendix B). It did however increase both the training time and the query time for the majority of datasets. The magnitude of the increase in training and query times is particularly large for the largest datasets.

However for some datasets, particularly SYNTH25 and SYNTH26 with only 2 features and 1,000,000 samples, an increase in the number of bins appears to result in a reduction of the query times. The reason for this appears to be that there are not enough bins to discriminate between different samples. As described in Section 4, many samples can

potentially be binned into the same bucket. With a combination of a small number of features, relatively few bins and a large number of samples, the buckets are overfilled. The result of this is that the filtering stage, where the contents of the most similar buckets returned by the CMM are compared with the query sample, has to perform a large number of expensive comparisons. In these situations, although increasing the number of bins slows down the CMM query, it also reduces the number of comparisons that have to be performed by the filtering. The end result is that the overall run times can be reduced.

For both OBCC and WOCC input tokens, increasing the number of bins leads to a significant reduction in the accuracy of the results in the synthetic datasets. As with the PK input tokens, the accuracy of the real world datasets was not significantly affected. A possible reason for the loss in accuracy is the max overlap parameter of the OBCC tokens (described in Section 5.2.3.3) results in a reduction in the discrimination between nearby bins, however further experiments would be needed to verify this. In these experiments the WOCC tokens are designed to emulate the OBCC tokens, and therefore has to honour the max overlap parameter, so it is expected that they will perform similarly.

With respect to the effects on training and query times for OBCC input tokens, it appears that increasing the number of bins actually results in a speed up. A possible explanation for this result is that the larger tokens result in fewer bits being set along each row. AURA performs a linear scan of the bits set in a row before it can set a particular bit on that row in the CMM. This is to prevent a bit being set twice in the sparse representation of the CMM. Yet with many bits being set in row, the overhead of this check can result in a significant slow down in the training of the CMM. Therefore by increasing the number of bins, it spreads the bits across more rows and reduces this overhead to training the CMM. However when the significant loss of accuracy is considered, it does not make sense to advocate using more bins with OBCC tokens.

Finally, the training and query times of the WOCC input tokens behave as expected with both query and training times increasing as the number of bins increases.

Summary

As a results of these experiments, it is clear that it is reasonable to only consider the experiment configurations that make use of 10 bins for the rest of the evaluation in this Section. This is because increasing the number of bins has been shown to have no significant effect on the accuracy of the real world datasets, and has either no significant effect, for PK, or a detrimental effect, for OBCC and WOCC, on the accuracy of the synthetic datasets. In addition, increasing the number of bins imposes a large performance penalty on the PK and WOCC input tokens.

7.2 Input Tokens

The purpose of this Section is to compare the PK, OBCC and WOCC tokens in the context of the AURA kNN algorithm. The accuracy, query time and training time of AURA kNN using each of the token types as input tokens will be compared with a view to determining the best token generation method to use in conjunction with unary output tokens.

Accuracy

In this Section, the accuracy of the PK, OBCC and WOCC input tokens is compared. The results of this experiment are given in Table 7.10.

Table 7.10: Mean Feature Distance for AURA Input Tokens (Synthetic)

ID	Features	Samples	Exact	PK	OBCC	WOCC
SYNTH01	2	1,000	0.062	0.062	0.062	0.062
SYNTH02	2	1,000	0.065	0.065	0.065	0.065
SYNTH03	10	1,000	0.142	0.149	0.181	0.181
SYNTH04	10	1,000	0.124	0.128	0.159	0.159
SYNTH05	100	1,000	0.071	0.072	0.075	0.075
SYNTH06	100	1,000	0.061	0.062	0.066	0.066
SYNTH07	1,000	1,000	0.025	0.025	0.025	0.025
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.020	0.020	0.020
SYNTH10	2	10,000	0.022	0.023	0.023	0.023
SYNTH11	10	10,000	0.107	0.110	0.155	0.155
SYNTH12	10	10,000	0.096	0.099	0.134	0.134
SYNTH13	100	10,000	0.067	0.069	0.073	0.073
SYNTH14	100	10,000	0.058	0.059	0.064	0.064
SYNTH15	1,000	10,000	0.024	0.025	0.025	0.025
SYNTH16	1,000	10,000	0.021	0.021	0.022	0.022
SYNTH17	2	100,000	0.006	0.006	0.006	0.006
SYNTH18	2	100,000	0.007	0.008	0.008	0.008
SYNTH19	10	100,000	0.082	0.084	0.126	0.126
SYNTH20	10	100,000	0.076	0.080	0.111	0.111
SYNTH21	100	100,000	0.064	0.066	0.072	0.072
SYNTH22	100	100,000	0.055	0.057	0.062	0.062
SYNTH23	1,000	100,000	0.024	0.024	0.025	0.025
SYNTH24	1,000	100,000	0.021	0.021	0.022	0.022
SYNTH25	2	1,000,000	0.002	0.002	0.002	0.002
SYNTH26	2	1,000,000	0.002	0.002	0.002	0.002
SYNTH27	10	1,000,000	0.063	0.066	0.098	0.098
SYNTH28	10	1,000,000	0.060	0.065	0.089	0.089
SYNTH29	100	1,000,000	0.062	0.064	0.070	0.070
SYNTH30	100	1,000,000	0.053	0.055	0.061	0.061

This table shows the mean feature distance for the AURA algorithm with PK, OBCC and WOCC input tokens, unary output tokens and 10 bins.

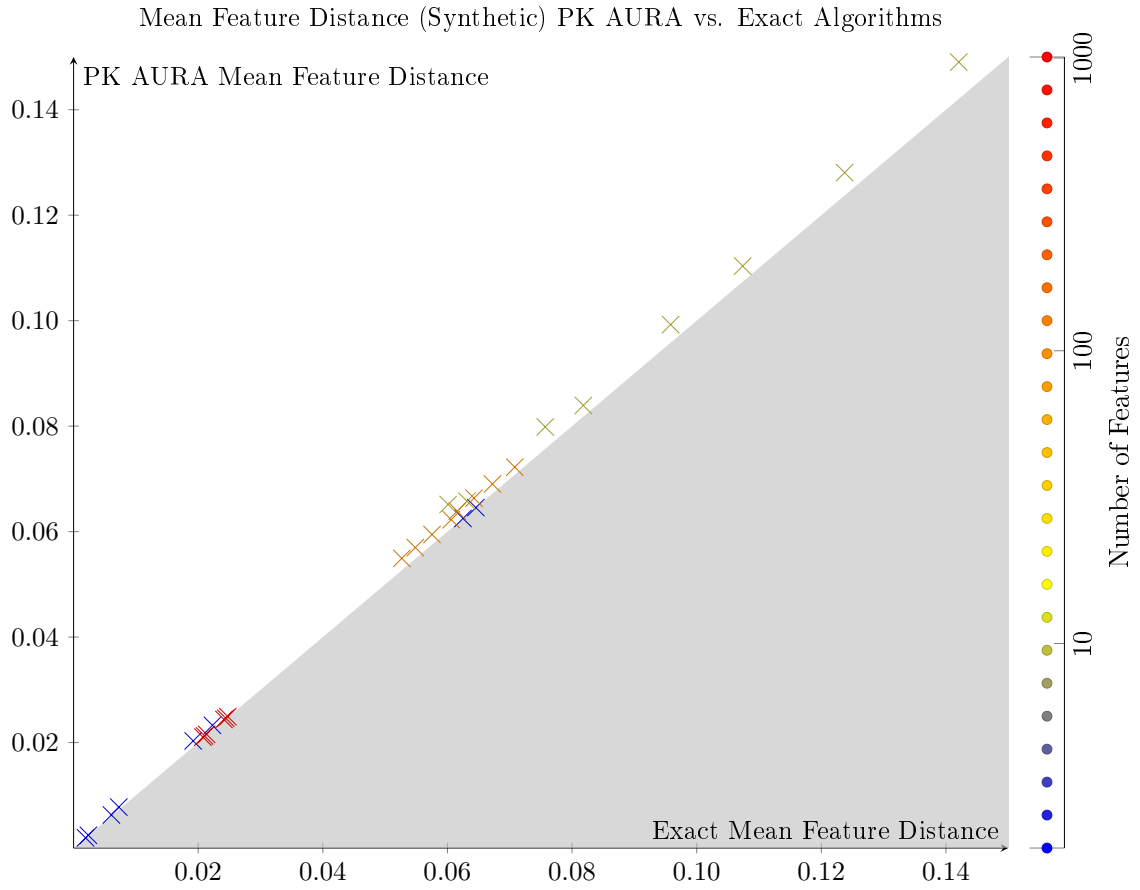


Figure 7.1: A comparison of the mean feature distance for each of the synthetic datasets. The PK input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A good result for PK AURA is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that PK AURA returns the same results as the exact algorithms.

Figure 7.1 shows how the mean feature distance of PK input tokens compared to that of the exact algorithms. The mean feature distance using PK AURA is only 2.9% larger than the exact algorithms. However it appears that PK AURA performs worst on the 10 feature datasets, with the mean feature distance for these datasets alone being 4.2% larger than the exact algorithm.

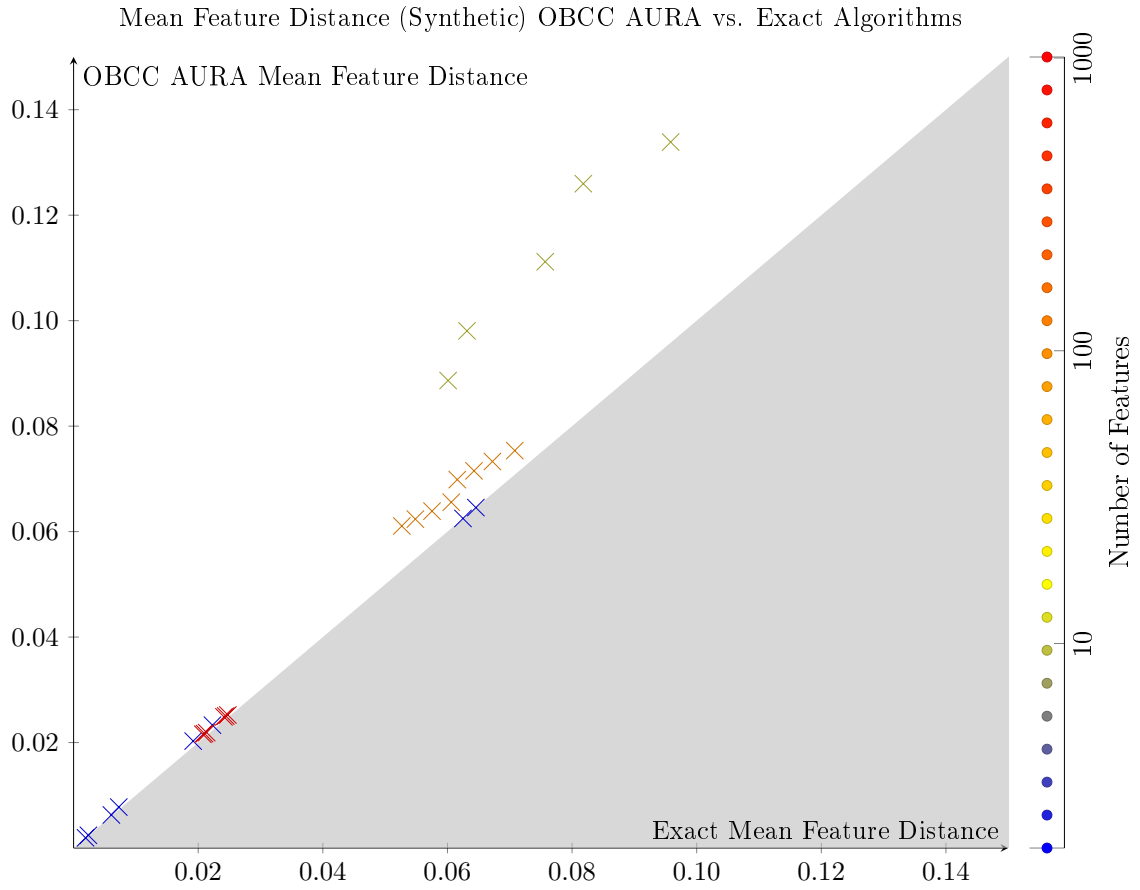


Figure 7.2: A comparison of the mean feature distance for each of the synthetic datasets. The OBCC input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A good result for OBCC AURA is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that OBCC AURA returns the same results as the exact algorithms.

Figure 7.2 shows how the mean feature distance of OBCC input tokens compares to that of the exact algorithms. It is clear that the accuracy of the 10 feature datasets is once again very poor. The mean feature distance of the 10 feature datasets is 30% larger than the exact algorithms. However the accuracy increases as the number of features increases. In particular, the 1000 feature datasets have a mean feature distance that is only 2.8% larger than the exact algorithm. Overall the penalty for using OBCC AURA over an exact algorithm is a mean increase of 11.9% in the mean feature distance across all datasets.

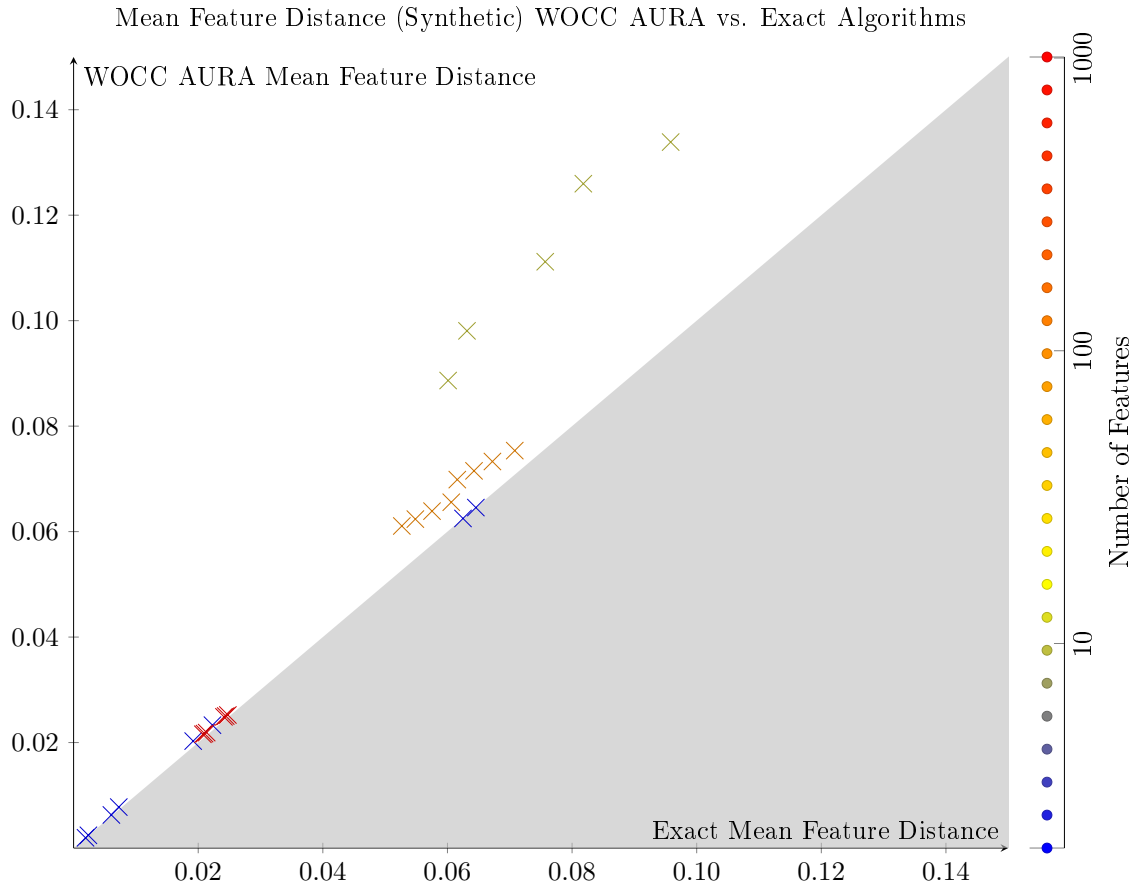


Figure 7.3: A comparison of the mean feature distance for each of the synthetic datasets. The WOCC input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A good result for WOCC AURA is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that WOCC AURA returns the same results as the exact algorithms.

Figure 7.3 shows how the mean feature distance of WOCC input tokens compares to that of the exact algorithms. From this it is clear that the accuracy characteristics are the same as the OBCC tokens. Indeed the mean feature distance observed with WOCC AURA is identical to that of OBCC AURA. This is the expected result.

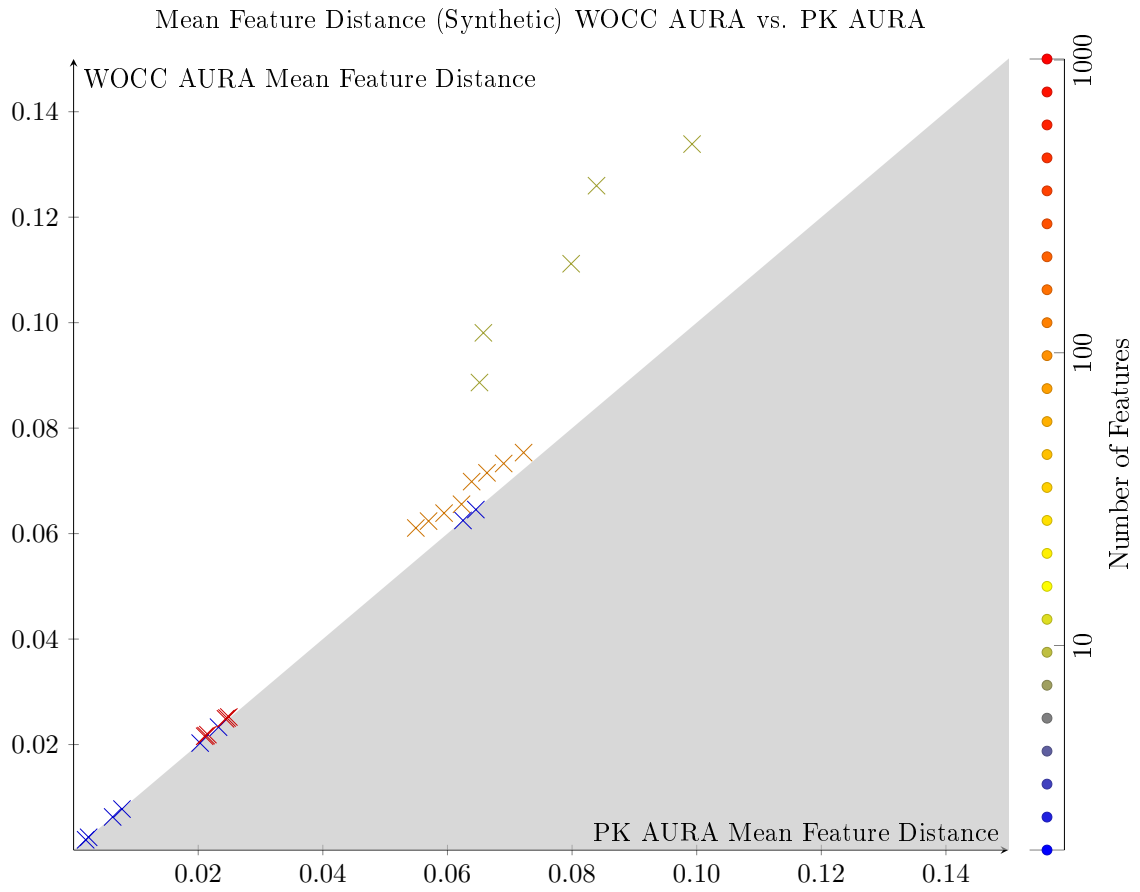


Figure 7.4: A comparison of the mean feature distance for each of the synthetic datasets. Both the WOCC input tokens and the PK input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. Markers in the shaded area indicate datasets where WOCC AURA has a smaller feature distance than PK AURA.

Figure 7.4 provides a comparison of the mean feature distance between AURA kNN using the PK input tokens and WOCC input tokens. With 2 feature and 1000 feature datasets, the accuracy of both methods is very similar with PK AURA producing a mean feature distance that is 0.9% smaller. However for the 10 and 100 feature datasets it is clear that PK input tokens are more accurate with a mean feature distance that is 6.9% lower for 100 feature datasets and 26% lower for 10 feature datasets.

Query Performance

In this Section the performance of the PK, OBCC and WOCC input tokens is considered with respect to query time. The results of this experiment are presented in Table 7.11.

The Dual KD–Tree algorithm was chosen to form the comparison from the exact algorithms because it was the fastest of the exact algorithms on the large datasets, particularly those with many features.

Table 7.11: Mean Query Time for AURA Input Tokens (Synthetic)

ID	Features	Samples	Dual KD–Tree	PK	OBCC	WOCC
SYNTH01	2	1,000	0.007	0.096	0.101	0.096
SYNTH02	2	1,000	0.013	0.097	0.103	0.118
SYNTH03	10	1,000	0.029	0.127	0.156	0.129
SYNTH04	10	1,000	0.026	0.112	0.167	0.145
SYNTH05	100	1,000	0.147	0.210	0.688	0.300
SYNTH06	100	1,000	0.166	0.216	0.862	0.342
SYNTH07	1,000	1,000	1.167	1.142	6.047	2.588
SYNTH08	1,000	1,000	1.307	1.314	8.401	2.828
SYNTH09	2	10,000	0.044	0.135	0.116	0.117
SYNTH10	2	10,000	0.052	0.118	0.119	0.132
SYNTH11	10	10,000	0.212	0.656	0.989	0.636
SYNTH12	10	10,000	0.218	0.621	1.168	0.804
SYNTH13	100	10,000	1.447	1.459	5.923	2.066
SYNTH14	100	10,000	1.359	1.561	6.524	2.709
SYNTH15	1,000	10,000	12.863	12.455	50.060	15.633
SYNTH16	1,000	10,000	12.216	12.900	60.062	18.951
SYNTH17	2	100,000	0.319	0.390	0.405	0.572
SYNTH18	2	100,000	0.324	0.465	0.505	0.453
SYNTH19	10	100,000	1.898	6.636	9.878	6.218
SYNTH20	10	100,000	2.105	6.071	12.101	7.956
SYNTH21	100	100,000	13.699	17.344	52.479	19.127
SYNTH22	100	100,000	13.688	18.543	62.582	24.743
SYNTH23	1,000	100,000	119.184	119.093	481.428	147.735
SYNTH24	1,000	100,000	130.298	119.534	572.691	178.000
SYNTH25	2	1,000,000	2.684	4.385	4.679	4.194
SYNTH26	2	1,000,000	3.093	6.132	6.034	5.274
SYNTH27	10	1,000,000	20.675	65.392	99.923	73.649
SYNTH28	10	1,000,000	19.216	66.379	107.512	76.556
SYNTH29	100	1,000,000	148.958	178.781	926.413	207.231
SYNTH30	100	1,000,000	140.706	182.917	1,185.169	251.387

This table shows the mean query time in seconds for the AURA algorithm with all three input tokens, unary output tokens and 10 bins.

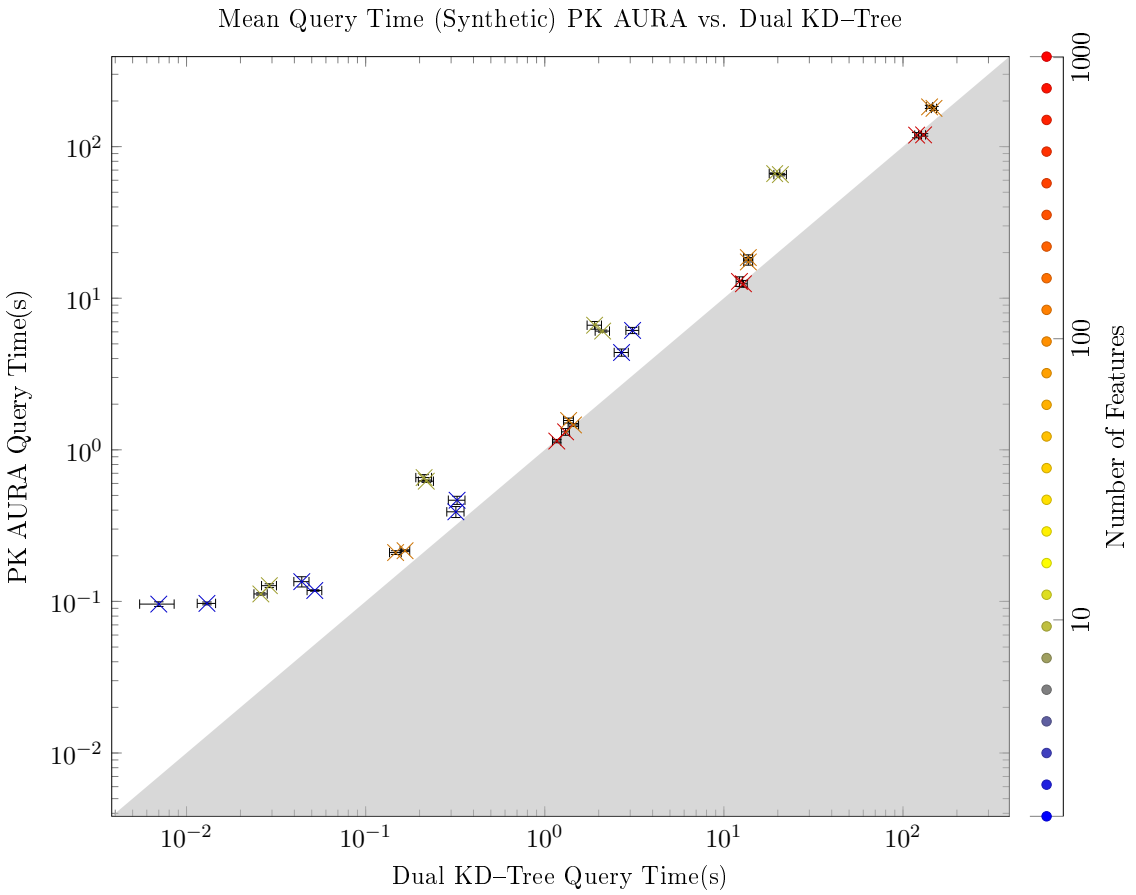


Figure 7.5: A comparison of the mean query time for each of the synthetic datasets. The PK input tokens for this experiment consist of 10 bins. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the PK AURA mean query time is faster.

Figure 7.5 shows a comparison between the mean query times of the datasets using AURA with PK input tokens and the Dual KD–Tree algorithm. It is clear that Dual KD–Tree is faster than the PK approach for most of the datasets, particularly the small datasets, the mean query time of PK AURA is 38% slower than the exact Dual KD–Tree algorithm. However for the 1000 feature datasets, the performance gap is very small with PK AURA being 1.5% faster. Indeed for dataset SYNTH24 PK is 10.8 seconds (9%) faster.

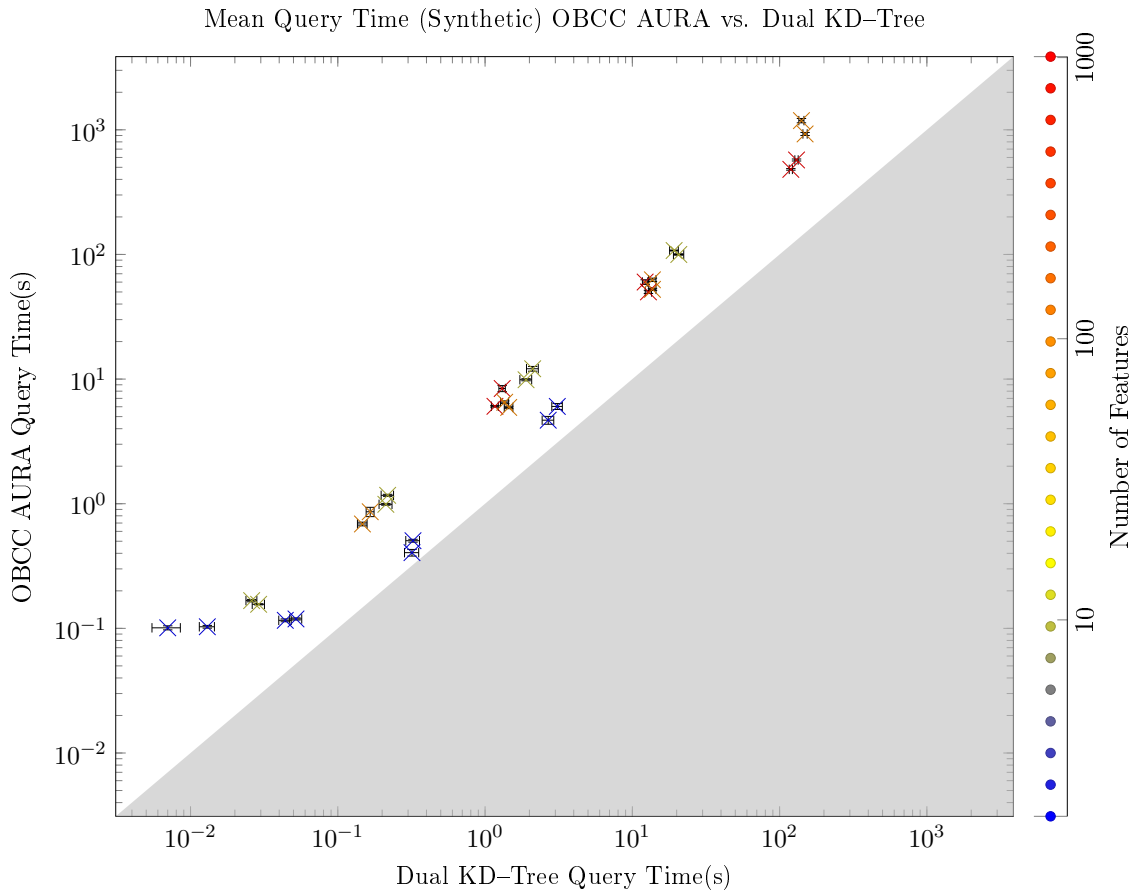


Figure 7.6: A comparison of the mean query time for each of the synthetic datasets. The OBCC input tokens for this experiment consist of 10 bins. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the OBCC AURA mean query time is faster.

Figure 7.6 shows a comparison of the mean query time between AURA with OBCC input tokens and the Dual KD–Tree algorithm. It is clear that OBCC is substantially slower than Dual KD–Tree for all datasets. Overall the mean query time of OBCC AURA is 4.9 times slower than Dual KD–Tree.

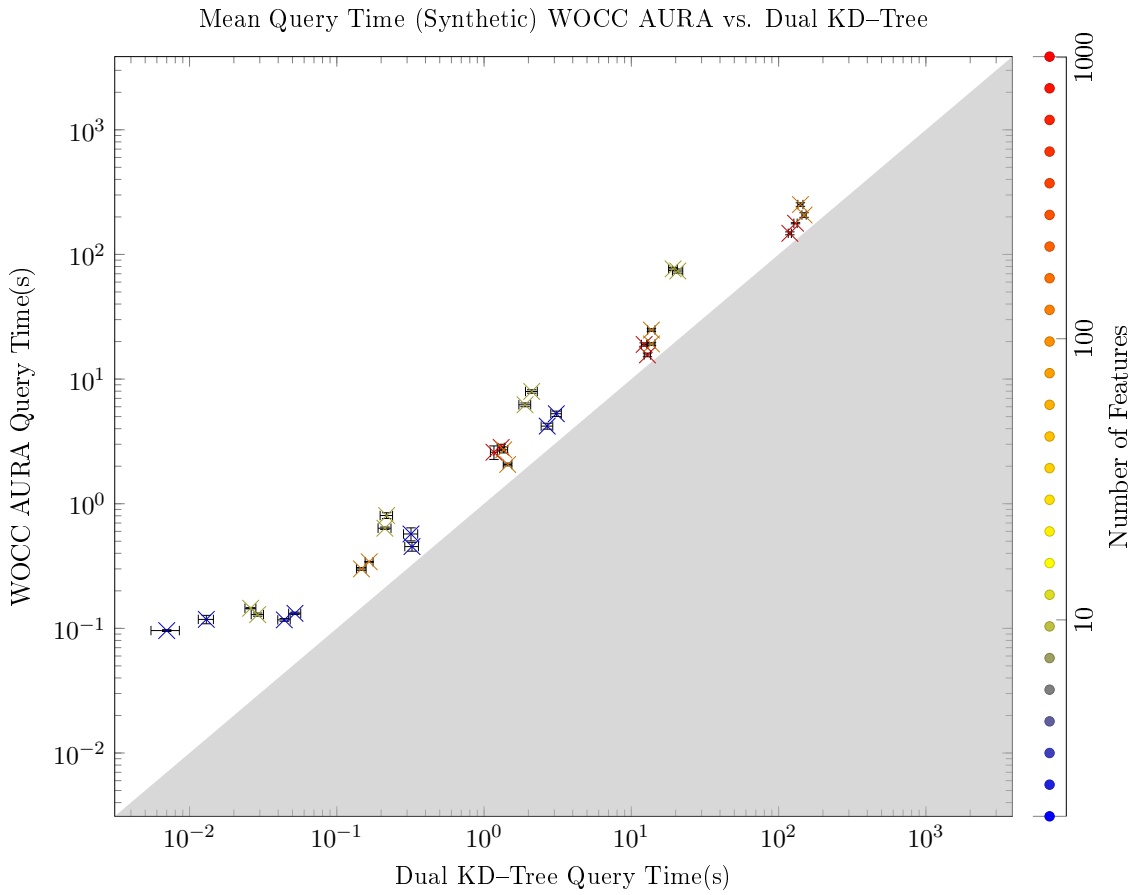


Figure 7.7: A comparison of the mean query time for each of the synthetic datasets. The WOCC input tokens for this experiment consist of 10 bins. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the WOCC AURA mean query time is faster.

Figure 7.7 shows a comparison of the mean query time between AURA with WOCC input tokens and the Dual KD–Tree algorithm. As with OBCC tokens, it is clear that WOCC is substantially slower than Dual KD–Tree for all datasets. The mean query time for WOCC AURA is 2.9 times slower than the Dual KD–Tree algorithm.

Training Performance

In this Section the performance of the PK, OBCC and WOCC input tokens is considered with respect to training time. The results of this experiment are presented in Table 7.12.

Table 7.12: Mean Training Time for AURA Input Tokens (Synthetic)

ID	Features	Samples	Dual KD–Tree	PK	OBCC	WOCC
SYNTH01	2	1,000	0.000	0.000	0.000	0.001
SYNTH02	2	1,000	0.000	0.000	0.001	0.001
SYNTH03	10	1,000	0.001	0.004	0.008	0.005
SYNTH04	10	1,000	0.000	0.003	0.009	0.005
SYNTH05	100	1,000	0.005	0.012	0.087	0.036
SYNTH06	100	1,000	0.002	0.011	0.097	0.036
SYNTH07	1,000	1,000	0.023	0.119	0.896	0.393
SYNTH08	1,000	1,000	0.031	0.128	1.128	0.405
SYNTH09	2	10,000	0.005	0.004	0.005	0.002
SYNTH10	2	10,000	0.003	0.007	0.005	0.003
SYNTH11	10	10,000	0.007	0.024	0.106	0.041
SYNTH12	10	10,000	0.008	0.024	0.114	0.051
SYNTH13	100	10,000	0.058	0.129	1.047	0.333
SYNTH14	100	10,000	0.057	0.123	1.000	0.379
SYNTH15	1,000	10,000	0.534	1.330	11.382	3.836
SYNTH16	1,000	10,000	0.497	1.249	12.095	3.644
SYNTH17	2	100,000	0.057	0.042	0.043	0.062
SYNTH18	2	100,000	0.075	0.043	0.045	0.042
SYNTH19	10	100,000	0.172	0.295	1.089	0.476
SYNTH20	10	100,000	0.182	0.255	1.233	0.493
SYNTH21	100	100,000	1.183	1.353	10.019	3.204
SYNTH22	100	100,000	1.084	1.327	10.317	3.905
SYNTH23	1,000	100,000	13.359	12.703	115.619	37.234
SYNTH24	1,000	100,000	15.501	11.536	116.010	37.478
SYNTH25	2	1,000,000	2.442	0.455	0.515	0.559
SYNTH26	2	1,000,000	2.046	0.529	0.452	0.472
SYNTH27	10	1,000,000	3.165	2.668	10.873	4.839
SYNTH28	10	1,000,000	3.127	2.728	10.740	5.237
SYNTH29	100	1,000,000	24.611	15.489	82.237	33.303
SYNTH30	100	1,000,000	23.537	13.033	91.697	37.722

This table shows the mean training time in seconds for the AURA algorithm with all three input tokens, unary output tokens and 10 bins.

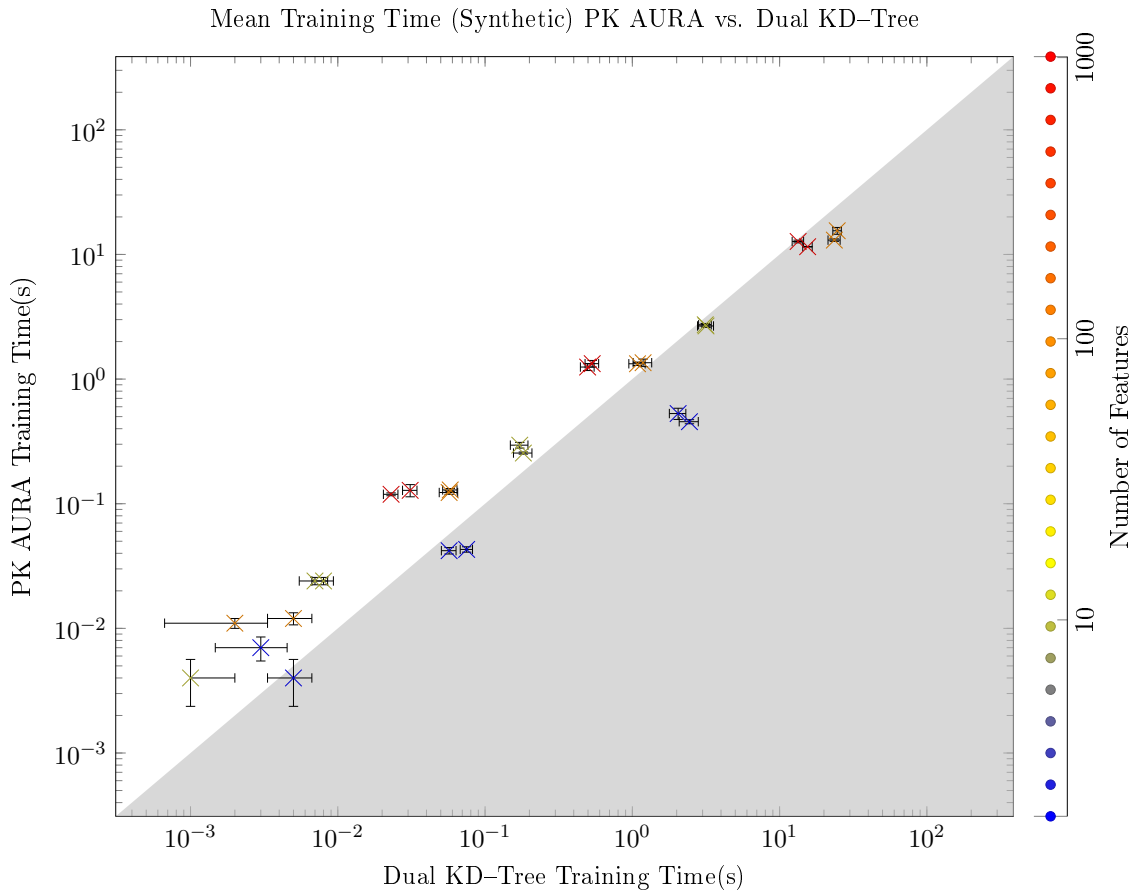


Figure 7.8: A comparison of the mean training time for each of the synthetic datasets. The PK input tokens for this experiment consist of 10 bins. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the PK AURA mean training time is faster.

Figure 7.8 shows a comparison of the mean training time between AURA with PK input tokens and the Dual KD–Tree algorithm. It appears that Dual KD–Tree is faster to train with the smaller datasets, however for bigger datasets PK appears to be faster. The rate at which the cross over point occurs appears to be dependent on the number of features in the dataset. The 10 feature and 100 feature datasets require 1,000,000 samples in order for PK AURA to be faster.

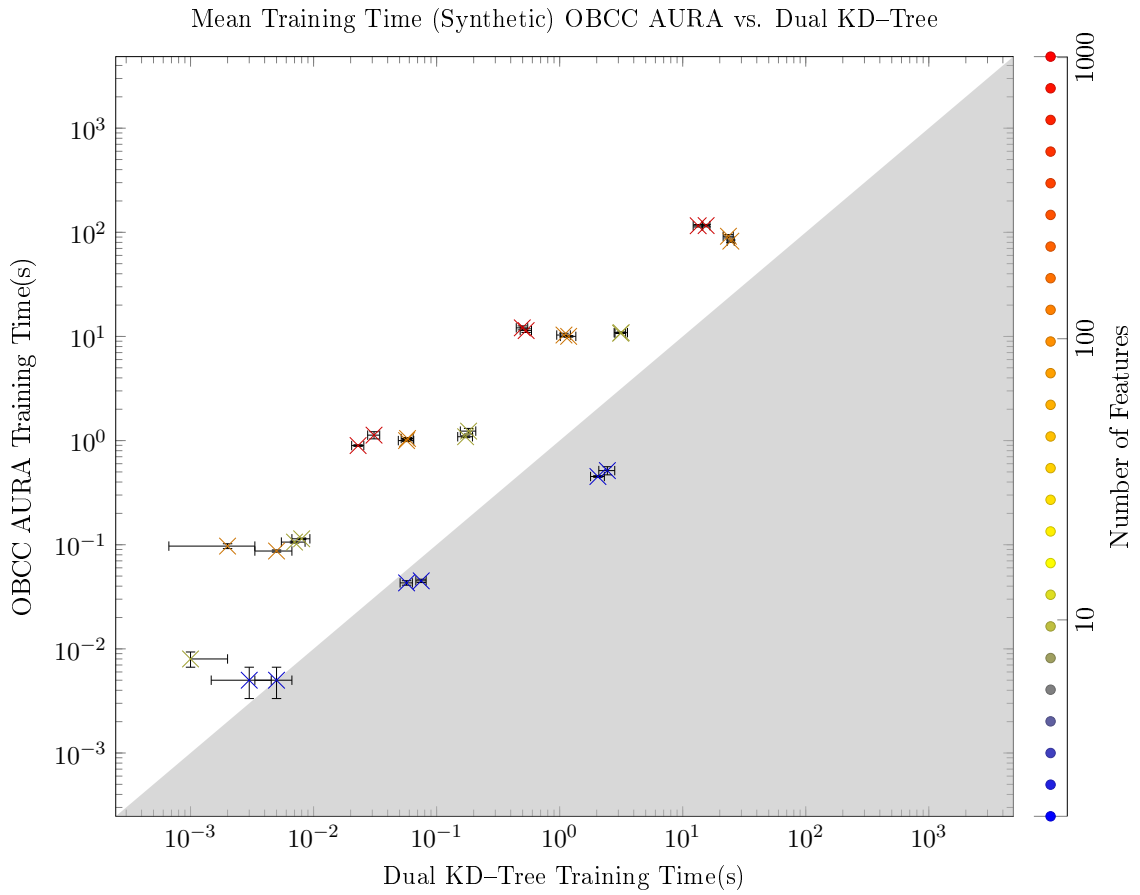


Figure 7.9: A comparison of the mean training time for each of the synthetic datasets. The OBCC input tokens for this experiment consist of 10 bins. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the OBCC AURA mean training time is faster.

Figure 7.9 shows a comparison of the mean training time between AURA with OBCC input tokens and the Dual KD–Tree algorithm. The OBCC tokens show the same general trend as the PK tokens in terms of comparing well with only 2 features. However the overall performance is considerably worse than the Dual KD–Tree algorithm, requiring a mean of 5.4 times longer to train a dataset.

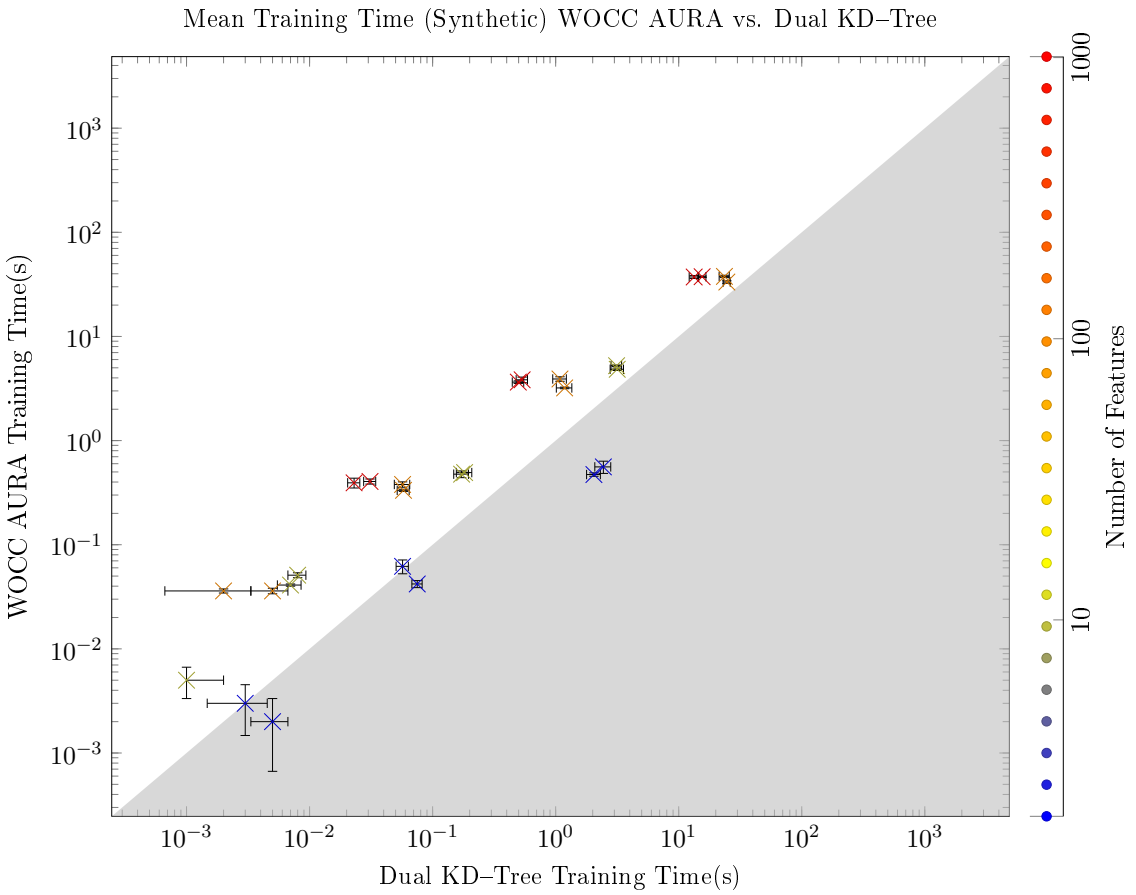


Figure 7.10: A comparison of the mean training time for each of the synthetic datasets. The WOCC input tokens for this experiment consist of 10 bins. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the WOCC AURA mean training time is faster.

Figure 7.10 shows a comparison of the mean training time between AURA with WOCC input tokens and the Dual KD–Tree algorithm. The WOCC tokens show the same general trend as the PK and OBCC tokens. The overall performance is an improvement on OBCC input tokens, despite being considerably worse than PK AURA. The WOCC input tokens require a mean of 2.2 times longer than the Dual KD–Tree algorithm to train a dataset.

Evaluation

In this Section the performance of the PK, OBCC and WOCC input tokens has been examined with 10 bins per feature and unary output tokens for the CMM.

In terms of accuracy it is clear than CMMs with OBCC and WOCC input tokens are able to provide equivalent levels of accuracy. This is expected because the WOCC input tokens were designed to emulate OBCC tokens. However the performance of the WOCC tokens is considerably better than OBCC tokens, as can be seen in both the training and query times. This is because the WOCC input tokens require a smaller CMM and fewer bits to be set within the CMM than OBCC tokens. As a result the WOCC tokens are strictly superior to OBCC input tokens when used within the AURA framework.

The PK input tokens are clearly superior to both the OBCC and WOCC tokens in these experiments both with respect to accuracy and performance. When considering

performance it is clear that the minimal number of bits that have to be set in the CMM (1 per feature) results in a considerable performance advantage over the other input tokens where many more bits have to be set.

However despite being faster and more accurate than the other CMM methods, PK is still generally worse than the exact Dual KD–Tree method. The only datasets where the approximate CMM based kNN algorithms are faster than Dual KD–Tree are those with only 2 features. As has been noted in Section 6.3, these are the datasets where Dual KD–Tree performs the worst and a standard KD–Tree is significantly faster.

Since the LSH algorithm has been shown to be faster than both the Dual KD–Tree and standard KD–Tree with a reasonable accuracy, it is not necessary to present a comparison between the PK AURA with unary output tokens and LSH.

Summary

As a result of these experiments it is clear that the PK input tokens provide significantly better accuracy, query and training performance than either WOCC or OBCC input tokens when used in conjunction with unary output tokens for the CMM based kNN algorithm. However despite this, there are exact methods that have superior accuracy and for the majority of datasets provide better performance.

7.3 Conclusion

The characteristics of the CMM based kNN algorithm using a unary output token have been examined in this Chapter.

In reference to the questions posed in Section 6.2.1 the following observations can be made about the AURA kNN algorithm with a unary output token.

With the PK and WOCC input tokens an increase in the number of bins will increase the training and query time of the algorithm. For the PK tokens, it does not however appear to have a significant effect on the accuracy of the algorithm across the synthetic or real world datasets. Yet for the OBCC and WOCC tokens an increase in the number of bins results in a reduction of the accuracy of the algorithm.

The results show that the best approach for generating the input tokens to be paired with unary output tokens is the PK method as this is faster and more accurate than the other approaches. This confirms that the standard approach used by AURA Alert is best when used with unary output tokens. It also appears that increasing the number of bins does not lead to substantial improvements in accuracy. Therefore the correct strategy appears to be using as few bins as possible without overfilling the discrete states with samples.

However overall the CMM based kNN algorithm has been shown to be slower than an exact method for each of the synthetic datasets. Therefore it appears that, from a technical perspective, a CMM based kNN algorithm with a unary output token is not the best algorithm to use in AURA Alert.

Chapter 8

AURA 2–bit Output Token Experiments

In this Chapter the performance of AURA kNN using 2–bit output tokens is investigated. The output tokens are generated using Baum Codes with two sections as described in Section 4.4.2. Since the length of the output tokens determines the number of columns in the CMM, this results in a CMM with fewer columns and more bits set than the CMMs considered in Chapter 7. Table 8.1 illustrates how the token length of the 2–bit output tokens compares to the unary output tokens for each of the synthetic datasets.

Table 8.1: Output Token Length

ID	Features	Samples	Unary Token Length	2 Bit Token Length
SYNTH01	2	1,000	1,000	21
SYNTH02	2	1,000	1,000	20
SYNTH03	10	1,000	1,000	64
SYNTH04	10	1,000	1,000	64
SYNTH05	100	1,000	1,000	64
SYNTH06	100	1,000	1,000	64
SYNTH07	1,000	1,000	1,000	64
SYNTH08	1,000	1,000	1,000	64
SYNTH09	2	10,000	10,000	21
SYNTH10	2	10,000	10,000	21
SYNTH11	10	10,000	10,000	201
SYNTH12	10	10,000	10,000	201
SYNTH13	100	10,000	10,000	201
SYNTH14	100	10,000	10,000	201
SYNTH15	1,000	10,000	10,000	201
SYNTH16	1,000	10,000	10,000	201
SYNTH17	2	100,000	100,000	21
SYNTH18	2	100,000	100,000	21
SYNTH19	10	100,000	100,000	633
SYNTH20	10	100,000	100,000	633
SYNTH21	100	100,000	100,000	633
SYNTH22	100	100,000	100,000	633
SYNTH23	1,000	100,000	100,000	633
SYNTH24	1,000	100,000	100,000	633
SYNTH25	2	1,000,000	1,000,000	21
SYNTH26	2	1,000,000	1,000,000	21
SYNTH27	10	1,000,000	1,000,000	2,000
SYNTH28	10	1,000,000	1,000,000	2,000
SYNTH29	100	1,000,000	1,000,000	2,001
SYNTH30	100	1,000,000	1,000,000	2,001

This table gives the length of the output tokens used to train each of the synthetic datasets for both unary output tokens and 2–bit Baum Coded output tokens.

The purpose of these experiments is twofold. It is necessary to determine which input token method, PK, OBCC or WOCC, performs best with the 2–bit output tokens. In addition it is necessary to compare the performance of the approximate AURA kNN algorithm using 2–bit output tokens with both the performance of the baseline kNN algorithms and the approximate AURA kNN algorithm using unary output tokens.

The performance of AURA kNN using multiple bits in the output tokens has not previously been investigated. It is expected that the accuracy of this method will suffer due to interference caused by multiple samples being superimposed within the CMM. It is possible that the PK method will suffer particularly because the method by which the weights are applied to each row of the input token will potentially result in an output score that is greater than would be expected by an exact match. In essence, the PK tokens effectively no longer have fixed weight.

The training time is also likely to be increased in comparison to using unary output tokens. This is because the CMM will be less sparse due to the increased number of bits that have to be set and this reduces the size of the CMM. As a result more collisions within the CMM are likely to occur during training and there is a performance penalty in detecting and resolving these collisions imposed by the AURA framework.

However it is hoped that the reduced CMM size caused by the use of more compact output tokens will reduce the query time of the AURA kNN and that this will be sufficient to offset the disadvantages described above.

In Section 8.1 the effect of varying the number of bins used to encode the input tokens is considered. This is followed by Section 8.2 where the performance of the three input token methods are compared to identify the best tokens to use with 2–bit tokens.

8.1 Number of Bins

The purpose of this experiment is to determine whether it is reasonable to consider only the 10 bin configurations in the further 2–bit output token experiments. The experiments with unary output tokens showed that 10 bins provided the best balance of speed and accuracy across all three input token methods. It is necessary to verify that this result still holds when using 2–bit output tokens.

Accuracy

In this Section the effect on the accuracy of increasing the number of bins is investigated for each of the three input token methods, PK, OBCC and WOCC. The table of results for these experiments on the synthetic datasets are given below, the results on the real world datasets can be found in Appendix B.

Table 8.2: Mean Feature Distance for Number of Bins with 2–Bit PK AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.062	0.130	0.222	0.345	0.414
SYNTH02	2	1,000	0.065	0.152	0.217	0.294	0.331
SYNTH03	10	1,000	0.142	0.248	0.244	0.242	0.238
SYNTH04	10	1,000	0.124	0.211	0.209	0.207	0.204
SYNTH05	100	1,000	0.071	0.081	0.080	0.080	0.080
SYNTH06	100	1,000	0.061	0.070	0.070	0.070	0.069
SYNTH07	1,000	1,000	0.025	0.026	0.026	0.026	0.026
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.188	0.138	0.155	0.281
SYNTH10	2	10,000	0.022	0.184	0.146	0.138	0.233
SYNTH11	10	10,000	0.107	0.246	0.247	0.244	0.241
SYNTH12	10	10,000	0.096	0.215	0.211	0.209	0.206
SYNTH13	100	10,000	0.067	0.081	0.081	0.081	0.080
SYNTH14	100	10,000	0.058	0.070	0.070	0.070	0.070
SYNTH15	1,000	10,000	0.024	0.026	0.026	0.026	0.026
SYNTH16	1,000	10,000	0.021	0.022	0.022	0.022	0.022
SYNTH17	2	100,000	0.006	0.299	0.388	0.164	0.121
SYNTH18	2	100,000	0.007	0.329	0.309	0.194	0.126
SYNTH19	10	100,000	0.082	0.243	0.243	0.243	0.246
SYNTH20	10	100,000	0.076	0.213	0.212	0.212	0.211
SYNTH21	100	100,000	0.064	0.081	0.081	0.081	0.081
SYNTH22	100	100,000	0.055	0.070	0.070	0.070	0.070
SYNTH23	1,000	100,000	0.024	0.026	0.026	0.026	0.026
SYNTH24	1,000	100,000	0.021	0.022	0.022	0.022	0.022
SYNTH25	2	1,000,000	0.002	0.353	0.343	0.472	0.372
SYNTH26	2	1,000,000	0.002	0.380	0.359	0.341	0.303
SYNTH27	10	1,000,000	0.063	0.250	0.246	0.246	0.250
SYNTH28	10	1,000,000	0.060	0.210	0.206	0.212	0.211
SYNTH29	100	1,000,000	0.062	0.081	0.081	0.081	0.081
SYNTH30	100	1,000,000	0.053	0.070	0.070	0.070	0.070

This table shows the mean feature distance for the AURA algorithm with PK input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.2 gives the results for the AURA kNN algorithm using the PK input tokens and 2–bit output tokens. As expected, there is no statistically significant difference between the observed feature distances (One–Way ANOVA, $f = 0.084$, $p = 0.969 > 0.05$) despite the huge increase in the number of bins. The real world datasets also did not display any statistically significant differences (One–Way ANOVA, $f = 0.232$, $p = 0.874 > 0.05$) as the number of bins increased.

Table 8.3: Mean Feature Distance for Number of Bins with 2–bit OBCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.062	0.120	0.243	0.379	0.435
SYNTH02	2	1,000	0.065	0.117	0.202	0.307	0.360
SYNTH03	10	1,000	0.142	0.249	0.247	0.246	0.246
SYNTH04	10	1,000	0.124	0.214	0.211	0.210	0.210
SYNTH05	100	1,000	0.071	0.081	0.081	0.081	0.081
SYNTH06	100	1,000	0.061	0.070	0.070	0.070	0.070
SYNTH07	1,000	1,000	0.025	0.026	0.026	0.026	0.026
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.203	0.134	0.160	0.317
SYNTH10	2	10,000	0.022	0.182	0.144	0.148	0.253
SYNTH11	10	10,000	0.107	0.246	0.248	0.247	0.248
SYNTH12	10	10,000	0.096	0.213	0.211	0.212	0.211
SYNTH13	100	10,000	0.067	0.081	0.081	0.081	0.081
SYNTH14	100	10,000	0.058	0.070	0.070	0.070	0.070
SYNTH15	1,000	10,000	0.024	0.026	0.026	0.026	0.026
SYNTH16	1,000	10,000	0.021	0.022	0.022	0.022	0.022
SYNTH17	2	100,000	0.006	0.464	0.392	0.239	0.150
SYNTH18	2	100,000	0.007	0.411	0.327	0.218	0.151
SYNTH19	10	100,000	0.082	0.243	0.243	0.244	0.249
SYNTH20	10	100,000	0.076	0.213	0.213	0.211	0.211
SYNTH21	100	100,000	0.064	0.081	0.081	0.081	0.081
SYNTH22	100	100,000	0.055	0.070	0.070	0.070	0.070
SYNTH23	1,000	100,000	0.024	0.026	0.026	0.026	0.026
SYNTH24	1,000	100,000	0.021	0.022	0.022	0.022	0.022
SYNTH25	2	1,000,000	0.002	0.475	0.486	0.497	0.385
SYNTH26	2	1,000,000	0.002	0.377	0.422	0.400	0.329
SYNTH27	10	1,000,000	0.063	0.250	0.246	0.246	0.246
SYNTH28	10	1,000,000	0.060	0.210	0.206	0.218	0.214
SYNTH29	100	1,000,000	0.062	0.081	0.081	0.081	0.081
SYNTH30	100	1,000,000	0.053	0.070	0.070	0.070	0.070

This table shows the mean feature distance for the AURA algorithm with OBCC input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.3 gives the results for the AURA Nearest Neighbours algorithm using the OBCC input tokens and 2–bit output tokens. There is no statistically significant difference between the observed feature distances (One–Way ANOVA, $f = 0.006$, $p = 0.999 > 0.05$) despite the huge increase in the number of bins. The real world datasets also did not display any statistically significant differences (One–Way ANOVA, $f = 0.188$, $p = 0.905 > 0.05$) as the number of bins increased. As a result it appears that the OBCC tokens do not exhibit the expected degradation in accuracy as the number of bins increases.

Table 8.4: Mean Feature Distance for Number of Bins with 2–bit WOCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.062	0.120	0.243	0.379	0.435
SYNTH02	2	1,000	0.065	0.117	0.202	0.307	0.360
SYNTH03	10	1,000	0.142	0.249	0.247	0.246	0.246
SYNTH04	10	1,000	0.124	0.214	0.211	0.210	0.210
SYNTH05	100	1,000	0.071	0.081	0.081	0.081	0.081
SYNTH06	100	1,000	0.061	0.070	0.070	0.070	0.070
SYNTH07	1,000	1,000	0.025	0.026	0.026	0.026	0.026
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.203	0.134	0.160	0.317
SYNTH10	2	10,000	0.022	0.182	0.144	0.148	0.253
SYNTH11	10	10,000	0.107	0.246	0.248	0.247	0.248
SYNTH12	10	10,000	0.096	0.213	0.211	0.212	0.211
SYNTH13	100	10,000	0.067	0.081	0.081	0.081	0.081
SYNTH14	100	10,000	0.058	0.070	0.070	0.070	0.070
SYNTH15	1,000	10,000	0.024	0.026	0.026	0.026	0.026
SYNTH16	1,000	10,000	0.021	0.022	0.022	0.022	0.022
SYNTH17	2	100,000	0.006	0.464	0.392	0.239	0.150
SYNTH18	2	100,000	0.007	0.411	0.327	0.218	0.151
SYNTH19	10	100,000	0.082	0.243	0.243	0.244	0.249
SYNTH20	10	100,000	0.076	0.213	0.213	0.211	0.211
SYNTH21	100	100,000	0.064	0.081	0.081	0.081	0.081
SYNTH22	100	100,000	0.055	0.070	0.070	0.070	0.070
SYNTH23	1,000	100,000	0.024	0.026	0.026	0.026	0.026
SYNTH24	1,000	100,000	0.021	0.022	0.022	0.022	0.022
SYNTH25	2	1,000,000	0.002	0.475	0.486	0.497	0.385
SYNTH26	2	1,000,000	0.002	0.377	0.422	0.400	0.329
SYNTH27	10	1,000,000	0.063	0.250	0.246	0.246	0.246
SYNTH28	10	1,000,000	0.060	0.210	0.206	0.218	0.214
SYNTH29	100	1,000,000	0.062	0.081	0.081	0.081	0.081
SYNTH30	100	1,000,000	0.053	0.070	0.070	0.070	0.070

This table shows the mean feature distance for the AURA algorithm with WOCC input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.4 gives the results for the AURA Nearest Neighbours algorithm using the WOCC input tokens and 2–bit output tokens. As expected, the results are identical to those for OBCC tokens and there is no statistically significant difference between the observed feature distances (One–Way ANOVA, $f = 0.006$, $p = 0.999 > 0.05$) despite the huge increase in the number of bins. The real world datasets also did not display any statistically significant differences (One–Way ANOVA, $f = 0.188$, $p = 0.905 > 0.05$) as the number of bins increased. Since the WOCC tokens produce the same results as the OBCC tokens, they also do not exhibit the expected degradation in accuracy as the number of bins increases.

Query Performance

This Section investigates the effect on query time of increasing the number of bins for each of the three input token methods, PK, OBCC and WOCC using 2–bit output tokens. The results of these experiments are given below.

Table 8.5: Mean Query Time for Number of Bins with 2–Bit PK AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.007	0.103	0.084	0.085	0.085
SYNTH02	2	1,000	0.013	0.110	0.080	0.089	0.089
SYNTH03	10	1,000	0.029	0.096	0.097	0.098	0.107
SYNTH04	10	1,000	0.026	0.093	0.093	0.103	0.107
SYNTH05	100	1,000	0.147	0.172	0.263	0.247	0.311
SYNTH06	100	1,000	0.166	0.175	0.251	0.292	0.323
SYNTH07	1,000	1,000	1.167	0.837	1.934	1.786	2.438
SYNTH08	1,000	1,000	1.307	0.918	1.696	2.272	2.567
SYNTH09	2	10,000	0.044	0.155	0.117	0.119	0.130
SYNTH10	2	10,000	0.052	0.125	0.117	0.118	0.124
SYNTH11	10	10,000	0.212	0.157	0.156	0.230	0.274
SYNTH12	10	10,000	0.218	0.151	0.165	0.219	0.291
SYNTH13	100	10,000	1.447	0.250	0.415	1.038	1.659
SYNTH14	100	10,000	1.359	0.274	0.482	0.984	1.754
SYNTH15	1,000	10,000	12.863	1.342	3.045	9.421	16.307
SYNTH16	1,000	10,000	12.216	1.582	3.544	8.559	17.029
SYNTH17	2	100,000	0.319	0.260	0.153	0.134	0.176
SYNTH18	2	100,000	0.324	0.353	0.174	0.152	0.182
SYNTH19	10	100,000	1.898	0.289	0.341	0.413	0.661
SYNTH20	10	100,000	2.105	0.282	0.350	0.502	0.730
SYNTH21	100	100,000	13.699	0.653	0.952	1.650	4.096
SYNTH22	100	100,000	13.688	0.771	0.994	2.052	4.508
SYNTH23	1,000	100,000	119.184	3.905	7.203	14.369	39.194
SYNTH24	1,000	100,000	130.298	3.850	7.744	17.266	43.349
SYNTH25	2	1,000,000	2.684	4.237	0.843	0.233	0.193
SYNTH26	2	1,000,000	3.093	4.052	1.013	0.301	0.215
SYNTH27	10	1,000,000	20.675	0.892	0.951	1.196	1.533
SYNTH28	10	1,000,000	19.216	0.818	0.944	1.190	1.826
SYNTH29	100	1,000,000	148.958	1.840	2.799	4.925	9.074
SYNTH30	100	1,000,000	140.706	1.955	2.881	5.414	10.340

This table shows the mean query time in seconds for the AURA algorithm with PK input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.5 gives the results for the AURA Nearest Neighbours algorithm using PK input tokens and 2–bit output tokens. It appears that the results are as expected. Increasing the number of bins yields a significant increase in query time (One–Way ANOVA, $f = 33.6$, $p = 6.89 \times 10^{-21} < 0.05$). The mean query time when using 10 bins is only 19% of the mean query time using 80 bins.

Table 8.6: Mean Query Time for Number of Bins with 2–Bit OBCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.007	0.108	0.082	0.086	0.088
SYNTH02	2	1,000	0.013	0.103	0.085	0.083	0.085
SYNTH03	10	1,000	0.029	0.094	0.096	0.100	0.106
SYNTH04	10	1,000	0.026	0.100	0.093	0.098	0.100
SYNTH05	100	1,000	0.147	0.246	0.239	0.247	0.308
SYNTH06	100	1,000	0.166	0.282	0.241	0.259	0.310
SYNTH07	1,000	1,000	1.167	1.674	1.686	2.234	3.404
SYNTH08	1,000	1,000	1.307	1.768	1.731	2.351	3.597
SYNTH09	2	10,000	0.044	0.125	0.112	0.118	0.121
SYNTH10	2	10,000	0.052	0.127	0.121	0.123	0.121
SYNTH11	10	10,000	0.212	0.156	0.141	0.169	0.190
SYNTH12	10	10,000	0.218	0.218	0.152	0.172	0.185
SYNTH13	100	10,000	1.447	0.418	0.421	0.530	0.702
SYNTH14	100	10,000	1.359	0.461	0.423	0.513	0.655
SYNTH15	1,000	10,000	12.863	3.381	3.123	4.624	6.748
SYNTH16	1,000	10,000	12.216	3.502	3.204	4.412	6.407
SYNTH17	2	100,000	0.319	0.397	0.171	0.168	0.198
SYNTH18	2	100,000	0.324	0.334	0.210	0.171	0.198
SYNTH19	10	100,000	1.898	0.351	0.337	0.347	0.381
SYNTH20	10	100,000	2.105	0.333	0.335	0.363	0.395
SYNTH21	100	100,000	13.699	1.182	1.025	1.150	1.410
SYNTH22	100	100,000	13.688	1.153	1.077	1.185	1.388
SYNTH23	1,000	100,000	119.184	10.110	8.140	9.409	12.804
SYNTH24	1,000	100,000	130.298	9.282	8.162	9.590	12.147
SYNTH25	2	1,000,000	2.684	3.781	0.898	0.346	0.233
SYNTH26	2	1,000,000	3.093	4.428	1.055	0.456	0.251
SYNTH27	10	1,000,000	20.675	1.026	0.968	0.968	1.002
SYNTH28	10	1,000,000	19.216	1.245	0.925	1.025	1.003
SYNTH29	100	1,000,000	148.958	3.580	3.010	3.258	3.599
SYNTH30	100	1,000,000	140.706	3.363	3.028	3.298	3.620

This table shows the mean query time in seconds for the AURA algorithm with OBCC input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.6 gives the results for the AURA Nearest Neighbours algorithm using OBCC input tokens and 2–bit output tokens. As expected, the query time generally increases as the number of bins increases. This increase is significant (One–Way ANOVA, $f = 3.56$, $p = 0.01 < 0.05$) with 10 Bins being 13.6% faster.

Table 8.7: Mean Query Time for Number of Bins with 2–Bit WOCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.007	0.101	0.085	0.082	0.080
SYNTH02	2	1,000	0.013	0.149	0.083	0.084	0.084
SYNTH03	10	1,000	0.029	0.089	0.092	0.095	0.098
SYNTH04	10	1,000	0.026	0.100	0.094	0.096	0.103
SYNTH05	100	1,000	0.147	0.145	0.165	0.218	0.269
SYNTH06	100	1,000	0.166	0.192	0.198	0.226	0.262
SYNTH07	1,000	1,000	1.167	0.580	0.898	1.566	2.949
SYNTH08	1,000	1,000	1.307	0.690	1.030	1.636	3.008
SYNTH09	2	10,000	0.044	0.130	0.119	0.118	0.123
SYNTH10	2	10,000	0.052	0.151	0.122	0.118	0.121
SYNTH11	10	10,000	0.212	0.143	0.133	0.155	0.182
SYNTH12	10	10,000	0.218	0.151	0.149	0.161	0.172
SYNTH13	100	10,000	1.447	0.239	0.289	0.391	0.568
SYNTH14	100	10,000	1.359	0.261	0.315	0.412	0.559
SYNTH15	1,000	10,000	12.863	1.110	1.636	2.891	5.551
SYNTH16	1,000	10,000	12.216	1.309	1.743	3.070	5.589
SYNTH17	2	100,000	0.319	0.447	0.184	0.160	0.168
SYNTH18	2	100,000	0.324	0.346	0.226	0.172	0.184
SYNTH19	10	100,000	1.898	0.304	0.309	0.327	0.345
SYNTH20	10	100,000	2.105	0.346	0.294	0.351	0.367
SYNTH21	100	100,000	13.699	0.542	0.635	0.906	1.157
SYNTH22	100	100,000	13.688	0.559	0.669	0.962	1.196
SYNTH23	1,000	100,000	119.184	2.677	4.102	6.786	9.301
SYNTH24	1,000	100,000	130.298	2.851	4.154	7.252	10.224
SYNTH25	2	1,000,000	2.684	3.883	1.075	0.325	0.224
SYNTH26	2	1,000,000	3.093	3.402	0.998	0.435	0.265
SYNTH27	10	1,000,000	20.675	0.784	0.857	0.939	0.956
SYNTH28	10	1,000,000	19.216	0.818	0.875	0.938	0.965
SYNTH29	100	1,000,000	148.958	1.449	1.911	2.621	3.244
SYNTH30	100	1,000,000	140.706	1.601	1.924	2.730	3.213

This table shows the mean query time in seconds for the AURA algorithm with WOCC input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.7 gives the results for the AURA Nearest Neighbours algorithm using WOCC input tokens and 2–bit output tokens. As expected, the query time generally increases as the number of bins increases. This is significant (One–Way ANOVA $f = 16.2$, $p = 2.39 \times 10^{-10} < 0.05$) and the query time using 10 bins is 51% faster that with 80 bins.

Training Performance

This Section investigates the effect on the training time of increasing the number of bins for the PK, OBCC and WOCC input tokens. The results of these experiments are given below.

Table 8.8: Mean Training Time for Number of Bins with 2–Bit PK AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.000	0.002	0.001	0.003	0.003
SYNTH02	2	1,000	0.000	0.001	0.001	0.001	0.001
SYNTH03	10	1,000	0.001	0.004	0.002	0.003	0.002
SYNTH04	10	1,000	0.000	0.004	0.003	0.004	0.003
SYNTH05	100	1,000	0.005	0.010	0.011	0.013	0.019
SYNTH06	100	1,000	0.002	0.010	0.012	0.014	0.018
SYNTH07	1,000	1,000	0.023	0.088	0.113	0.170	0.211
SYNTH08	1,000	1,000	0.031	0.082	0.113	0.153	0.198
SYNTH09	2	10,000	0.005	0.006	0.006	0.009	0.016
SYNTH10	2	10,000	0.003	0.006	0.005	0.010	0.014
SYNTH11	10	10,000	0.007	0.037	0.026	0.028	0.031
SYNTH12	10	10,000	0.008	0.030	0.025	0.027	0.031
SYNTH13	100	10,000	0.058	0.098	0.110	0.132	0.166
SYNTH14	100	10,000	0.057	0.101	0.108	0.126	0.152
SYNTH15	1,000	10,000	0.534	0.867	1.048	1.363	2.010
SYNTH16	1,000	10,000	0.497	0.871	1.018	1.275	1.794
SYNTH17	2	100,000	0.057	0.039	0.047	0.054	0.073
SYNTH18	2	100,000	0.075	0.043	0.044	0.053	0.074
SYNTH19	10	100,000	0.172	0.294	0.310	0.321	0.324
SYNTH20	10	100,000	0.182	0.271	0.322	0.328	0.305
SYNTH21	100	100,000	1.183	1.163	1.182	1.379	1.544
SYNTH22	100	100,000	1.084	1.252	1.196	1.353	1.501
SYNTH23	1,000	100,000	13.359	10.061	11.607	14.371	18.269
SYNTH24	1,000	100,000	15.501	9.838	11.277	14.442	18.181
SYNTH25	2	1,000,000	2.442	0.521	0.465	0.544	0.640
SYNTH26	2	1,000,000	2.046	0.495	0.503	0.597	0.633
SYNTH27	10	1,000,000	3.165	3.297	3.278	3.315	3.359
SYNTH28	10	1,000,000	3.127	2.972	3.190	3.326	3.355
SYNTH29	100	1,000,000	24.611	11.616	12.415	13.874	15.476
SYNTH30	100	1,000,000	23.537	11.967	12.064	13.955	14.904

This table shows the mean training time in seconds for the AURA algorithm with PK input tokens, 2–bit output tokens and a varying numbers of bins.

Table 7.7 gives the results for the AURA Nearest Neighbours algorithm using PK input tokens and 2–bit output tokens. The training time generally appears to increase as the number of bins increases, but this is not statistically significant (One–Way ANOVA, $f = 2.385$, $p = 0.07 > 0.05$).

Table 8.9: Mean Training Time for Number of Bins with 2–Bit OBCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.000	0.001	0.002	0.005	0.007
SYNTH02	2	1,000	0.000	0.000	0.001	0.004	0.005
SYNTH03	10	1,000	0.001	0.007	0.007	0.010	0.011
SYNTH04	10	1,000	0.000	0.009	0.008	0.008	0.016
SYNTH05	100	1,000	0.005	0.071	0.065	0.098	0.159
SYNTH06	100	1,000	0.002	0.080	0.067	0.095	0.160
SYNTH07	1,000	1,000	0.023	1.070	1.144	1.592	2.234
SYNTH08	1,000	1,000	0.031	1.085	1.092	1.736	2.269
SYNTH09	2	10,000	0.005	0.004	0.006	0.010	0.026
SYNTH10	2	10,000	0.003	0.006	0.005	0.008	0.018
SYNTH11	10	10,000	0.007	0.079	0.075	0.099	0.116
SYNTH12	10	10,000	0.008	0.115	0.077	0.088	0.108
SYNTH13	100	10,000	0.058	0.728	0.669	0.902	1.406
SYNTH14	100	10,000	0.057	0.726	0.795	0.896	1.494
SYNTH15	1,000	10,000	0.534	10.067	11.007	16.559	21.774
SYNTH16	1,000	10,000	0.497	10.025	10.322	15.497	20.295
SYNTH17	2	100,000	0.057	0.045	0.045	0.060	0.085
SYNTH18	2	100,000	0.075	0.038	0.047	0.057	0.081
SYNTH19	10	100,000	0.172	0.858	0.845	1.053	1.233
SYNTH20	10	100,000	0.182	0.875	0.857	1.005	1.168
SYNTH21	100	100,000	1.183	8.633	8.067	11.063	16.201
SYNTH22	100	100,000	1.084	8.390	8.082	10.913	16.468
SYNTH23	1,000	100,000	13.359	114.587	118.270	178.087	260.882
SYNTH24	1,000	100,000	15.501	111.360	128.898	182.769	239.580
SYNTH25	2	1,000,000	2.442	0.437	0.465	0.559	0.628
SYNTH26	2	1,000,000	2.046	0.517	0.508	0.589	0.639
SYNTH27	10	1,000,000	3.165	10.344	8.935	10.705	12.222
SYNTH28	10	1,000,000	3.127	11.864	8.873	10.648	12.778
SYNTH29	100	1,000,000	24.611	95.253	87.570	122.108	174.761
SYNTH30	100	1,000,000	23.537	92.965	88.383	125.196	176.401

This table shows the mean training time in seconds for the AURA algorithm with OBCC input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.9 gives the results for the AURA Nearest Neighbours algorithm using OBCC input tokens and dual bit output tokens. As expected, the training time exhibits a statistically significant increase as the number of bins increases (One–Way ANOVA, $f = 6.506$, $p = 0.0002 < 0.05$) with 80 bins requiring a training time that is twice (2.06) as slow as 10 bins.

Table 8.10: Mean Training Time for Number of Bins with 2–Bit WOCC AURA (Synthetic)

ID	Features	Samples	Exact	10 Bins	20 Bins	40 Bins	80 Bins
SYNTH01	2	1,000	0.000	0.001	0.000	0.001	0.004
SYNTH02	2	1,000	0.000	0.001	0.000	0.002	0.002
SYNTH03	10	1,000	0.001	0.006	0.005	0.006	0.010
SYNTH04	10	1,000	0.000	0.002	0.006	0.006	0.010
SYNTH05	100	1,000	0.005	0.026	0.036	0.064	0.108
SYNTH06	100	1,000	0.002	0.038	0.044	0.064	0.116
SYNTH07	1,000	1,000	0.023	0.296	0.510	1.059	1.744
SYNTH08	1,000	1,000	0.031	0.336	0.572	1.079	1.771
SYNTH09	2	10,000	0.005	0.003	0.005	0.007	0.018
SYNTH10	2	10,000	0.003	0.005	0.009	0.010	0.016
SYNTH11	10	10,000	0.007	0.038	0.053	0.068	0.094
SYNTH12	10	10,000	0.008	0.042	0.058	0.076	0.091
SYNTH13	100	10,000	0.058	0.254	0.375	0.653	1.146
SYNTH14	100	10,000	0.057	0.294	0.421	0.646	1.085
SYNTH15	1,000	10,000	0.534	3.005	5.663	11.264	17.675
SYNTH16	1,000	10,000	0.497	3.616	6.710	11.945	17.986
SYNTH17	2	100,000	0.057	0.060	0.050	0.056	0.076
SYNTH18	2	100,000	0.075	0.047	0.048	0.058	0.078
SYNTH19	10	100,000	0.172	0.477	0.596	0.832	1.076
SYNTH20	10	100,000	0.182	0.545	0.626	0.891	1.049
SYNTH21	100	100,000	1.183	2.973	4.413	8.044	12.219
SYNTH22	100	100,000	1.084	3.225	4.635	8.064	13.052
SYNTH23	1,000	100,000	13.359	37.936	65.723	124.237	197.385
SYNTH24	1,000	100,000	15.501	38.851	74.417	141.724	202.160
SYNTH25	2	1,000,000	2.442	0.504	0.499	0.550	0.617
SYNTH26	2	1,000,000	2.046	0.454	0.472	0.550	0.653
SYNTH27	10	1,000,000	3.165	4.637	5.881	8.696	11.057
SYNTH28	10	1,000,000	3.127	5.001	6.193	8.691	11.191
SYNTH29	100	1,000,000	24.611	32.927	55.456	91.833	139.538
SYNTH30	100	1,000,000	23.537	35.202	52.941	93.827	151.068

This table shows the mean training time in seconds for the AURA algorithm with WOCC input tokens, 2–bit output tokens and a varying numbers of bins.

Table 8.10 gives the results for the AURA Nearest Neighbours algorithm using WOCC input tokens and dual bit output tokens. As expected, the training time exhibits a statistically significant increase as the number of bins increases (One–Way ANOVA, $f = 17.5$, $p = 3.858 \times 10^{-11} < 0.05$). The mean training time with 10 bin tokens is 4.6 times faster than with 80 bins.

Evaluation

In this Section, the effect of varying the number of bins used when generating the input tokens for the CMM has been examined when 2–bit output tokens are used.

The PK tokens exhibit the expected behaviour with respect to accuracy and query time. As the number of bins are increased the query takes longer and the accuracy is not significantly improved. This matches the results from the experiments with unary tokens. However the expected increase in training time for PK is not statistically significant. In general the expected result can be observed, in particular the large datasets that take multiple seconds to train show a clear trend, in which more bins leads to longer training times. However the smaller datasets that require only tens of milliseconds to train often

do not exhibit the expected increase. Considering the relatively small training times it is possible that issues such as process scheduling, cache misses and in some cases the precision of the timing code could be responsible. Additional repetitions of these experiments would be required to identify the cause, however because this anomalous result does not affect the overall decision to consider the 10 bins configuration in the remaining experiments, performing the additional repetitions is not necessary.

The OBCC and WOCC tokens appear to behave differently with 2–bit output tokens compared to unary output tokens. With unary output tokens, the input tokens exhibit a loss in accuracy as the number of bins increased. However with the 2–bit output tokens, there is no statistically significant change in the observed feature distance for both the synthetic and real world datasets as the number of bins increases. A possible explanation for this is due to the reduced accuracy observed with 2–bit output tokens, since the baseline accuracy is worse than with unary output tokens there is less degradation possible before the accuracy is equivalent to a random guess. Therefore increasing the number of bins does not have such a significant effect on the observed accuracy.

With respect to the query and training time, OBCC and WOCC tokens perform as expected when the number of bins increases. Specifically, the training and query times increase as the number of bins increase. This is simply due to the need for larger CMMs to store the pairs of samples.

Summary

As a results of these experiments, it is clear that the findings with respect to the number of bins to use for these datasets with unary output tokens still hold when using 2–bit output tokens. Increasing the number of bins has been shown to have no significant effect on the accuracy of either the real world datasets or synthetic datasets. In addition, increasing the number of bins appears to have a detrimental effect on the query and training times of all three input token methods.

It is therefore reasonable to only consider the experiment configurations that make use of 10 bins for the remaining evaluation of 2–bit output tokens in this Section.

8.2 Input Tokens

The purpose of this experiment is to determine which of the three input token methods, PK, OBCC or WOCC performs best with 2–bit output tokens. These will only consider using 10 bins for each of the input token methods. The experiments with unary output tokens showed that the PK method was superior. However it is expected that this method will suffer the most when applied to 2–bit output tokens since the variable weights applied to each row combined with multiple samples being superimposed over a single column will result in the PK tokens essentially no longer having a fixed weight.

Accuracy

In this Section, the accuracy of the PK, OBCC and WOCC input tokens while using 2–bit output tokens is compared. The results of this experiment are given in Table 8.11.

Table 8.11: Mean Feature Distance for AURA Input Tokens (Synthetic)

ID	Features	Samples	Exact	Parabolic	OBCC	WOCC
SYNTH01	2	1,000	0.062	0.130	0.120	0.120
SYNTH02	2	1,000	0.065	0.152	0.117	0.117
SYNTH03	10	1,000	0.142	0.248	0.249	0.249
SYNTH04	10	1,000	0.124	0.211	0.214	0.214
SYNTH05	100	1,000	0.071	0.081	0.081	0.081
SYNTH06	100	1,000	0.061	0.070	0.070	0.070
SYNTH07	1,000	1,000	0.025	0.026	0.026	0.026
SYNTH08	1,000	1,000	0.021	0.022	0.022	0.022
SYNTH09	2	10,000	0.019	0.188	0.203	0.203
SYNTH10	2	10,000	0.022	0.184	0.182	0.182
SYNTH11	10	10,000	0.107	0.246	0.246	0.246
SYNTH12	10	10,000	0.096	0.215	0.213	0.213
SYNTH13	100	10,000	0.067	0.081	0.081	0.081
SYNTH14	100	10,000	0.058	0.070	0.070	0.070
SYNTH15	1,000	10,000	0.024	0.026	0.026	0.026
SYNTH16	1,000	10,000	0.021	0.022	0.022	0.022
SYNTH17	2	100,000	0.006	0.299	0.464	0.464
SYNTH18	2	100,000	0.007	0.329	0.411	0.411
SYNTH19	10	100,000	0.082	0.243	0.243	0.243
SYNTH20	10	100,000	0.076	0.213	0.213	0.213
SYNTH21	100	100,000	0.064	0.081	0.081	0.081
SYNTH22	100	100,000	0.055	0.070	0.070	0.070
SYNTH23	1,000	100,000	0.024	0.026	0.026	0.026
SYNTH24	1,000	100,000	0.021	0.022	0.022	0.022
SYNTH25	2	1,000,000	0.002	0.353	0.475	0.475
SYNTH26	2	1,000,000	0.002	0.380	0.377	0.377
SYNTH27	10	1,000,000	0.063	0.250	0.250	0.250
SYNTH28	10	1,000,000	0.060	0.210	0.210	0.210
SYNTH29	100	1,000,000	0.062	0.081	0.081	0.081
SYNTH30	100	1,000,000	0.053	0.070	0.070	0.070

This table shows the mean feature distance for the AURA algorithm with all three input tokens, 2–bit output tokens and 10 bins.

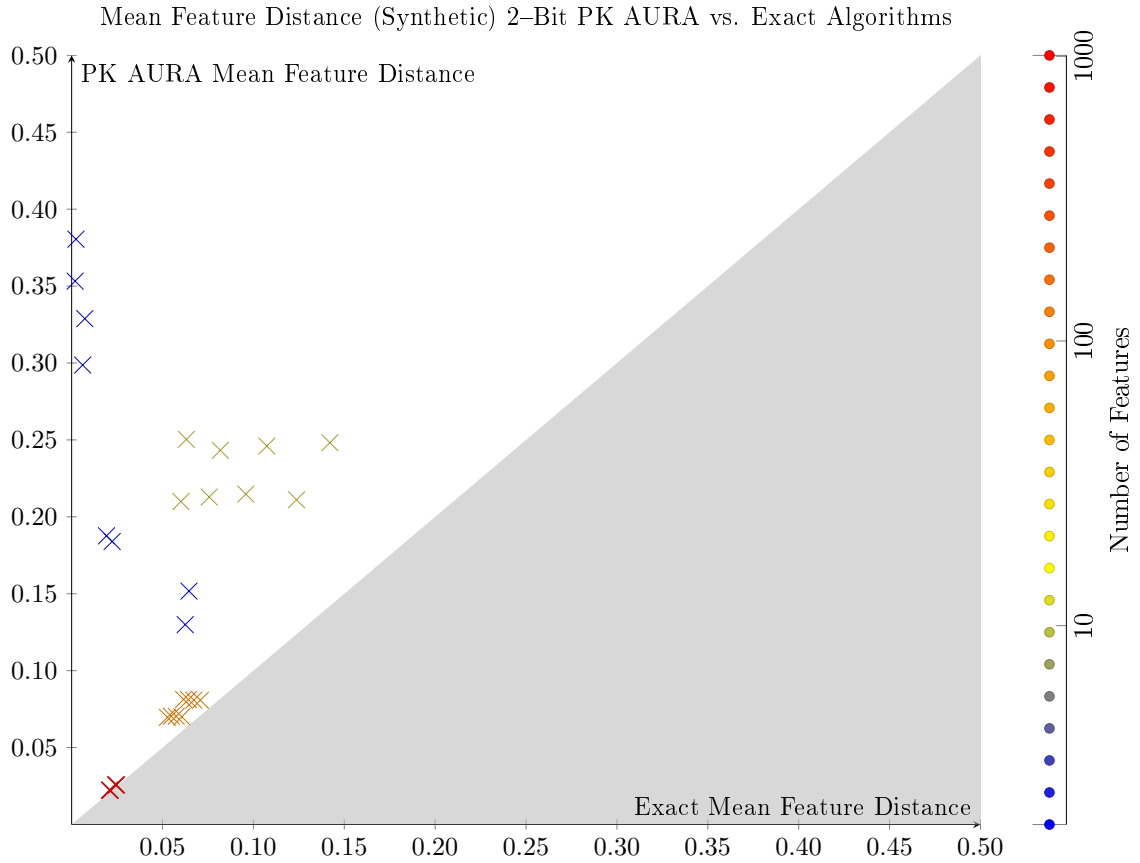


Figure 8.1: A comparison of the mean feature distance for each of the synthetic datasets. The PK input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A good result for PK AURA is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that PK AURA returns the same results as the exact algorithms.

Figure 8.1 shows how the mean feature distance of PK input tokens compares to that of the exact algorithms. It is clear that the datasets with more features are able to be queried more accurately. The 1000 feature datasets are the most accurate, the mean feature distance for these datasets is only 5.6% larger than the exact results. The 10 and 2 feature datasets are incredibly inaccurate in comparison, for 10 features, the mean feature distance is 2.65 times larger than the exact results and for 2 features, the mean feature distance is 67.7 times larger. The poor performance of the datasets with few features also appears to be made worse by increasing the number of samples stored in the CMM. With only 1000 2–feature samples stored, the mean feature distance is only twice as large as the exact result, with 1,000,000 samples stored, the mean feature distance is in the worst case 185 times larger. In contrast, for the 100–feature datasets, mean feature distance increases from a mean of 15% worse than the exact result to only 32% worse when the number of samples increases from 1,000 to 1,000,000.

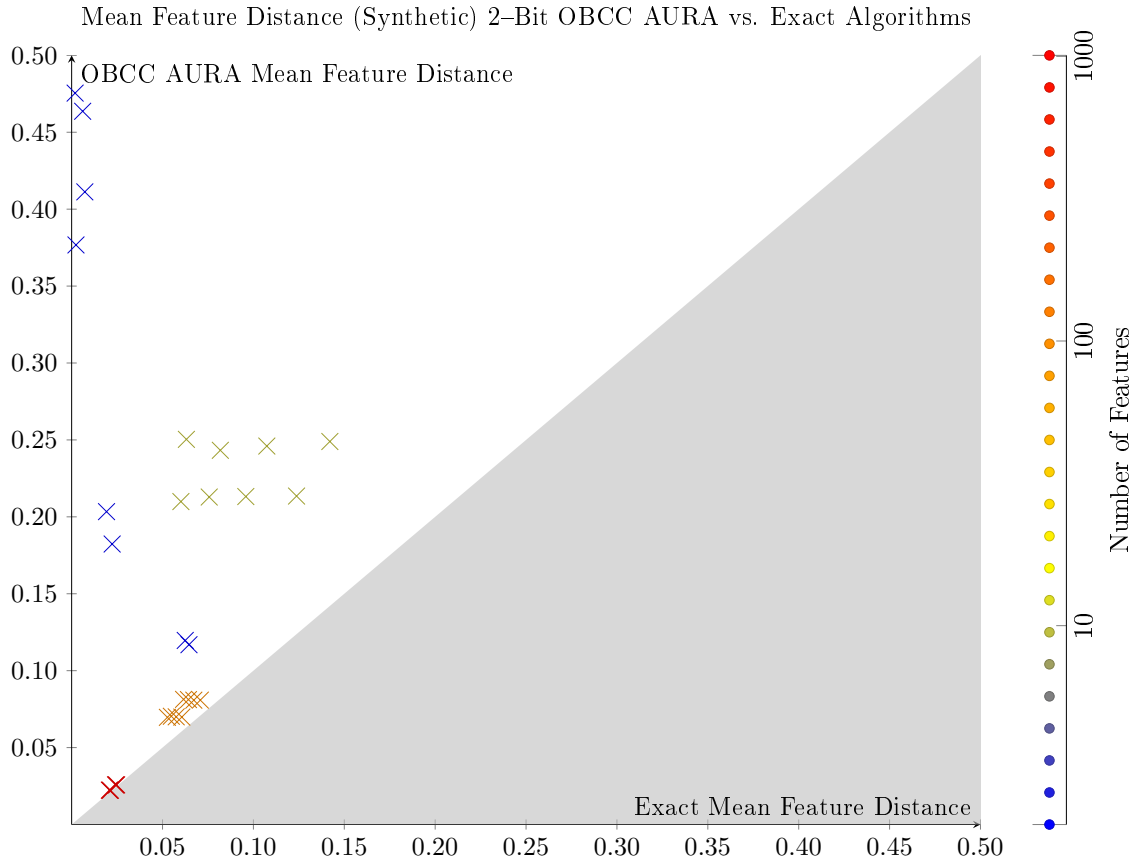


Figure 8.2: A comparison of the mean feature distance for each of the synthetic datasets. The OBCC input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A good result for OBCC AURA is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that OBCC AURA returns the same results as the exact algorithms.

Figure 8.2 shows how the mean feature distance of OBCC input tokens compared to that of the exact algorithms. As with the PK input tokens, it appears that the observed mean feature distance on the 2–feature and 10–feature datasets is very poor. Indeed the accuracy on these datasets is typically slightly worse with the mean feature distance of the 2–feature datasets ranging from 1.9 times the exact result for the smallest datasets to 249 times the exact result with the largest datasets. However the mean feature distance of the 1000 feature datasets is only 5.7% worse than the exact results.

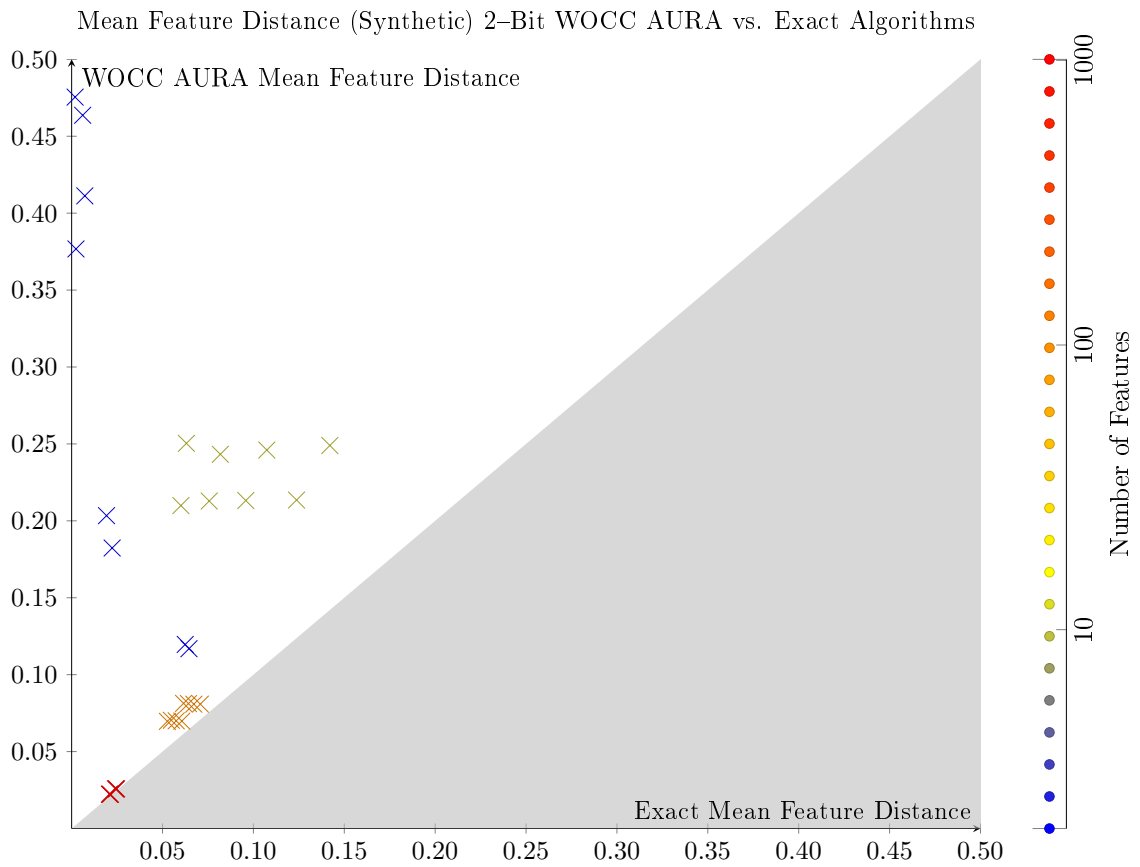


Figure 8.3: A comparison of the mean feature distance for each of the synthetic datasets. The WOCC input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A good result for WOCC AURA is indicated by how close the marker is to the boundary with the shaded area. A marker on the boundary indicates that WOCC AURA returns the same results as the exact algorithms.

Figure 8.3 shows how the mean feature distance of WOCC input tokens compared to that of the exact algorithms. The characteristics of the mean feature distance are, as expected, exactly the same as the OBCC tokens.

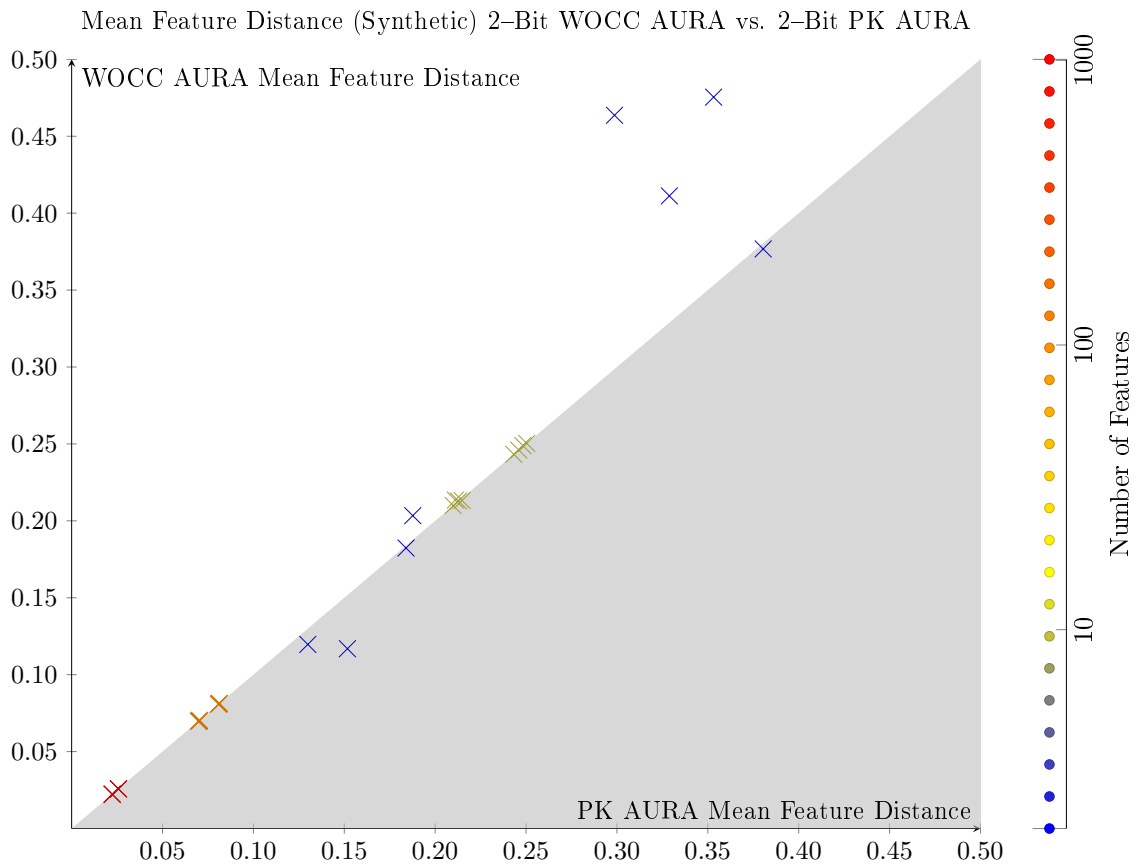


Figure 8.4: A comparison of the mean feature distance for each of the synthetic datasets. Both the WOCC input tokens and the PK input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. Markers in the shaded area indicate datasets where WOCC AURA has a smaller feature distance than PK AURA.

Figure 8.4 compares the mean feature distance of the OBCC and WOCC input tokens with that of the PK input tokens. It is clear that, with the exception of the 2–feature datasets, the accuracy of all the input token methods is very similar when used in conjunction with 2–bit output tokens. The difference in mean feature distance, excluding the 2 feature datasets, between AURA knn using PK and OBCC or WOCC tokens is only 0.005%.

Query Performance

In this Section the performance of the PK, OBCC and WOCC input tokens is considered with respect to query time. The results of this experiment are presented in Table 8.12.

As with the unary token experiments, the Dual KD–Tree algorithm was chosen to form the comparison from the exact algorithms because it was the fastest of the exact algorithms on the large datasets, particularly those with many features.

Table 8.12: Mean Query Time for AURA Input Tokens with 2–Bit Output Tokens (Synthetic)

ID	Features	Samples	Dual KD–Tree	Parabolic	OBCC	WOCC
SYNTH01	2	1,000	0.007	0.103	0.108	0.101
SYNTH02	2	1,000	0.013	0.110	0.103	0.149
SYNTH03	10	1,000	0.029	0.096	0.094	0.089
SYNTH04	10	1,000	0.026	0.093	0.100	0.100
SYNTH05	100	1,000	0.147	0.172	0.246	0.145
SYNTH06	100	1,000	0.166	0.175	0.282	0.192
SYNTH07	1,000	1,000	1.167	0.837	1.674	0.580
SYNTH08	1,000	1,000	1.307	0.918	1.768	0.690
SYNTH09	2	10,000	0.044	0.155	0.125	0.130
SYNTH10	2	10,000	0.052	0.125	0.127	0.151
SYNTH11	10	10,000	0.212	0.157	0.156	0.143
SYNTH12	10	10,000	0.218	0.151	0.218	0.151
SYNTH13	100	10,000	1.447	0.250	0.418	0.239
SYNTH14	100	10,000	1.359	0.274	0.461	0.261
SYNTH15	1,000	10,000	12.863	1.342	3.381	1.110
SYNTH16	1,000	10,000	12.216	1.582	3.502	1.309
SYNTH17	2	100,000	0.319	0.260	0.397	0.447
SYNTH18	2	100,000	0.324	0.353	0.334	0.346
SYNTH19	10	100,000	1.898	0.289	0.351	0.304
SYNTH20	10	100,000	2.105	0.282	0.333	0.346
SYNTH21	100	100,000	13.699	0.653	1.182	0.542
SYNTH22	100	100,000	13.688	0.771	1.153	0.559
SYNTH23	1,000	100,000	119.184	3.905	10.110	2.677
SYNTH24	1,000	100,000	130.298	3.850	9.282	2.851
SYNTH25	2	1,000,000	2.684	4.237	3.781	3.883
SYNTH26	2	1,000,000	3.093	4.052	4.428	3.402
SYNTH27	10	1,000,000	20.675	0.892	1.026	0.784
SYNTH28	10	1,000,000	19.216	0.818	1.245	0.818
SYNTH29	100	1,000,000	148.958	1.840	3.580	1.449
SYNTH30	100	1,000,000	140.706	1.955	3.363	1.601

This table shows the mean query time in seconds for the AURA algorithm with all three input tokens, 2–bit output tokens and 10 bins.

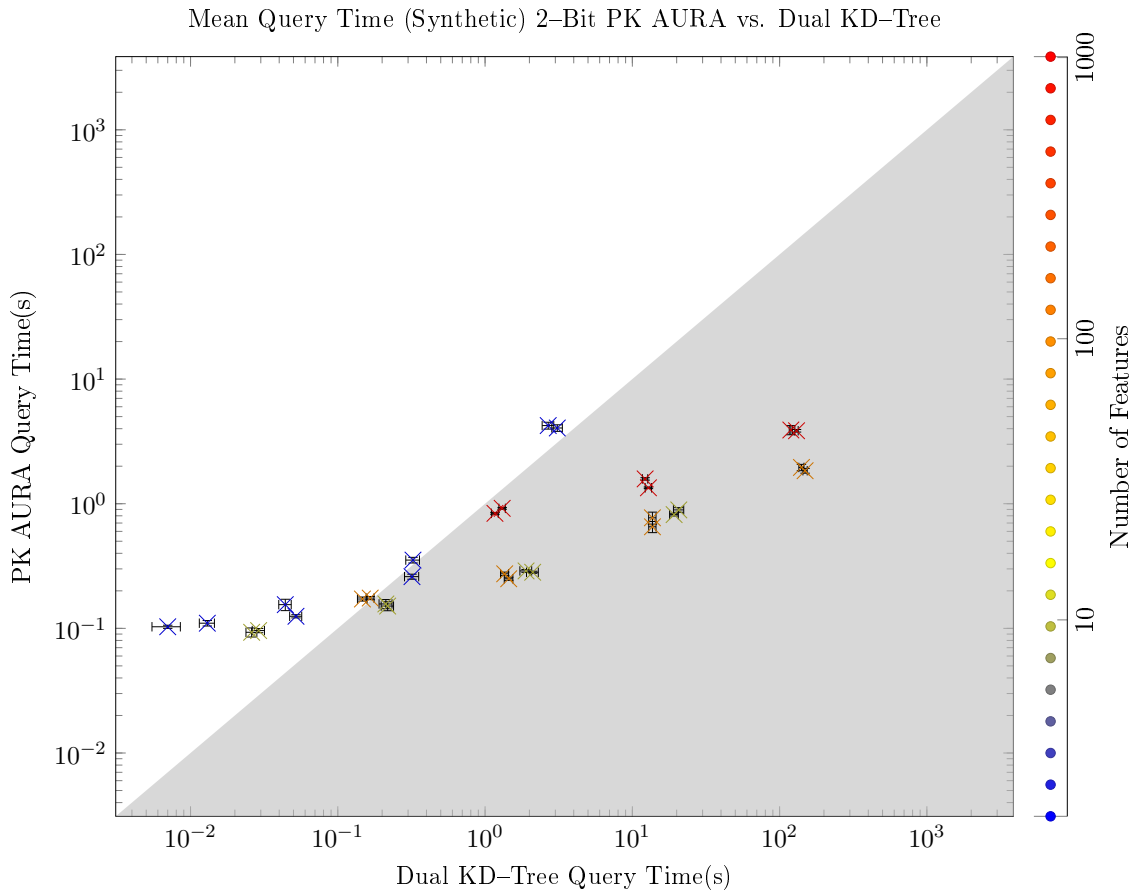


Figure 8.5: A comparison of the mean query time for each of the synthetic datasets. The PK input tokens for this experiment consist of 10 bins. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the PK AURA mean query time is faster.

Figure 8.5 shows a comparison between the mean query times of the datasets using AURA with PK input tokens and the Dual KD–Tree algorithm. The Dual KD–Tree is faster than the PK approach for both the small datasets that have only 1,000 samples and the datasets that consist of only 2–features. Otherwise the PK AURA approach is significantly faster than the Dual KD–Tree method. This is particularly true with the large datasets where the AURA approach is a mean of 76.5 times faster than the Dual KD–Tree method when querying 1,000,000 samples.

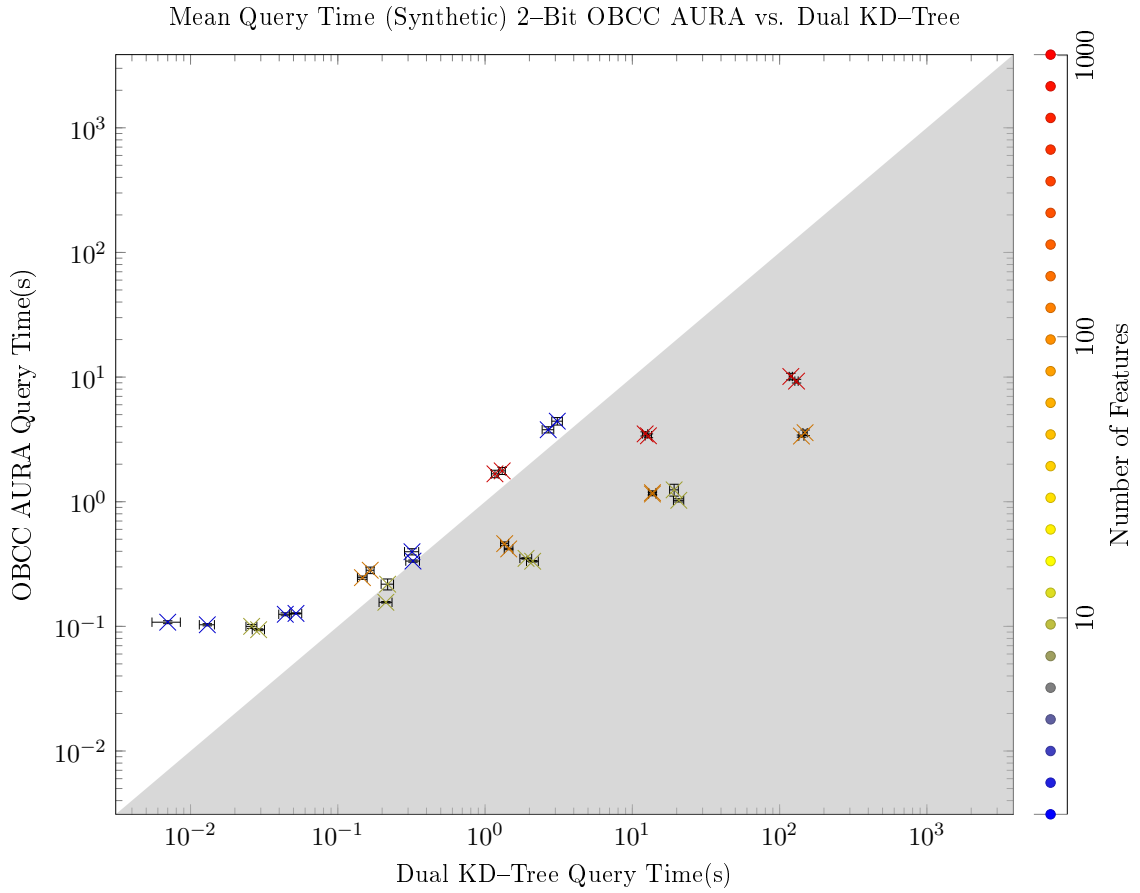


Figure 8.6: A comparison of the mean query time for each of the synthetic datasets. The OBCC input tokens for this experiment consist of 10 bins. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the OBCC AURA mean query time is faster.

Figure 8.6 shows a comparison between the mean query times of the datasets using AURA with OBCC input tokens and the Dual KD–Tree algorithm. As observed with the PK tokens, the Dual KD–Tree is faster than the OBCC AURA approach for both the small datasets that have only 1,000 samples and the datasets that consist of only 2–features while the AURA approach is faster for the large datasets. The OBCC tokens do however appear to be considerably slower than the PK tokens. With the large datasets, OBCC tokens are only 41.7 times faster than the Dual KD–Tree method where the PK tokens were 76.5 time faster.

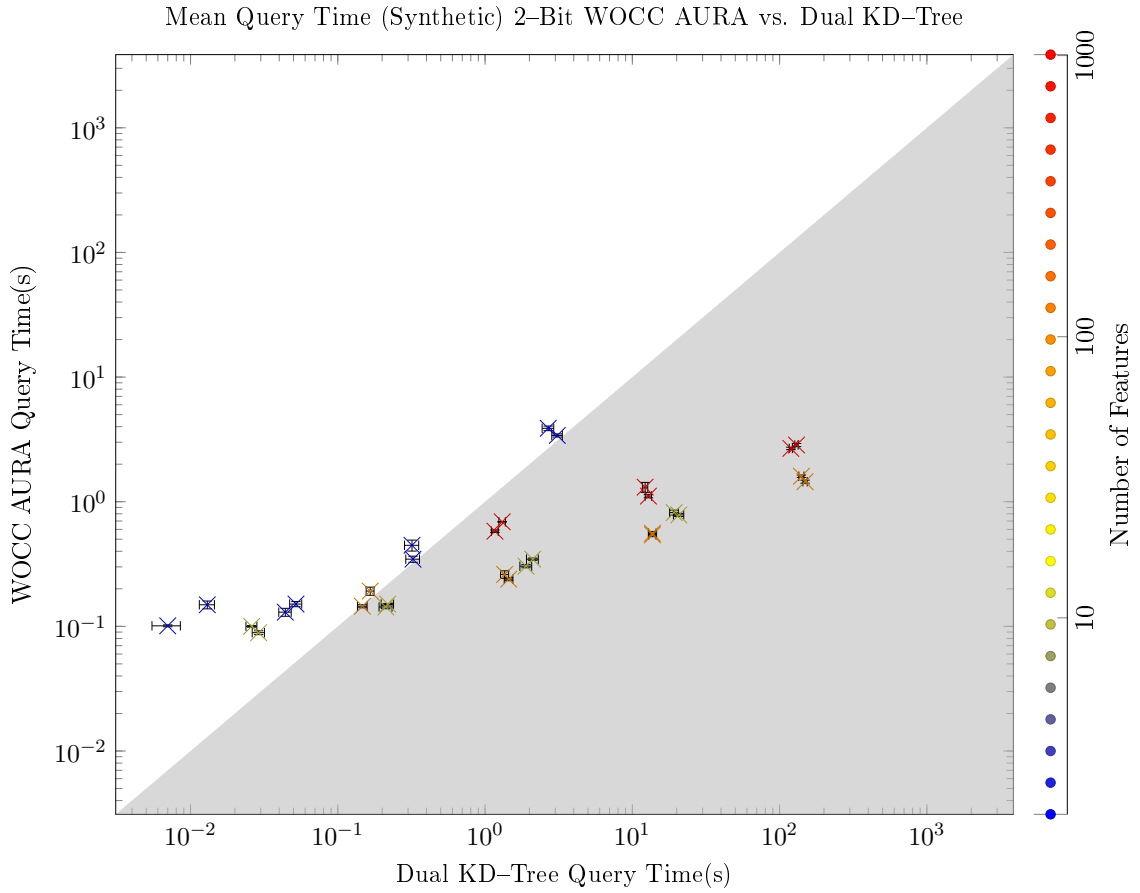


Figure 8.7: A comparison of the mean query time for each of the synthetic datasets. The WOCC input tokens for this experiment consist of 10 bins. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the WOCC AURA mean query time is faster.

Figure 8.7 shows a comparison between the mean query times of the datasets using AURA with WOCC input tokens and the Dual KD–Tree algorithm. The results are inline with the result of the other two input token methods. Dual KD–Tree is faster for the small datasets and slower for the large datasets. WOCC AURA is a mean of 95.3 times faster than Dual KD–Tree for the 1,000,000 sample datasets.

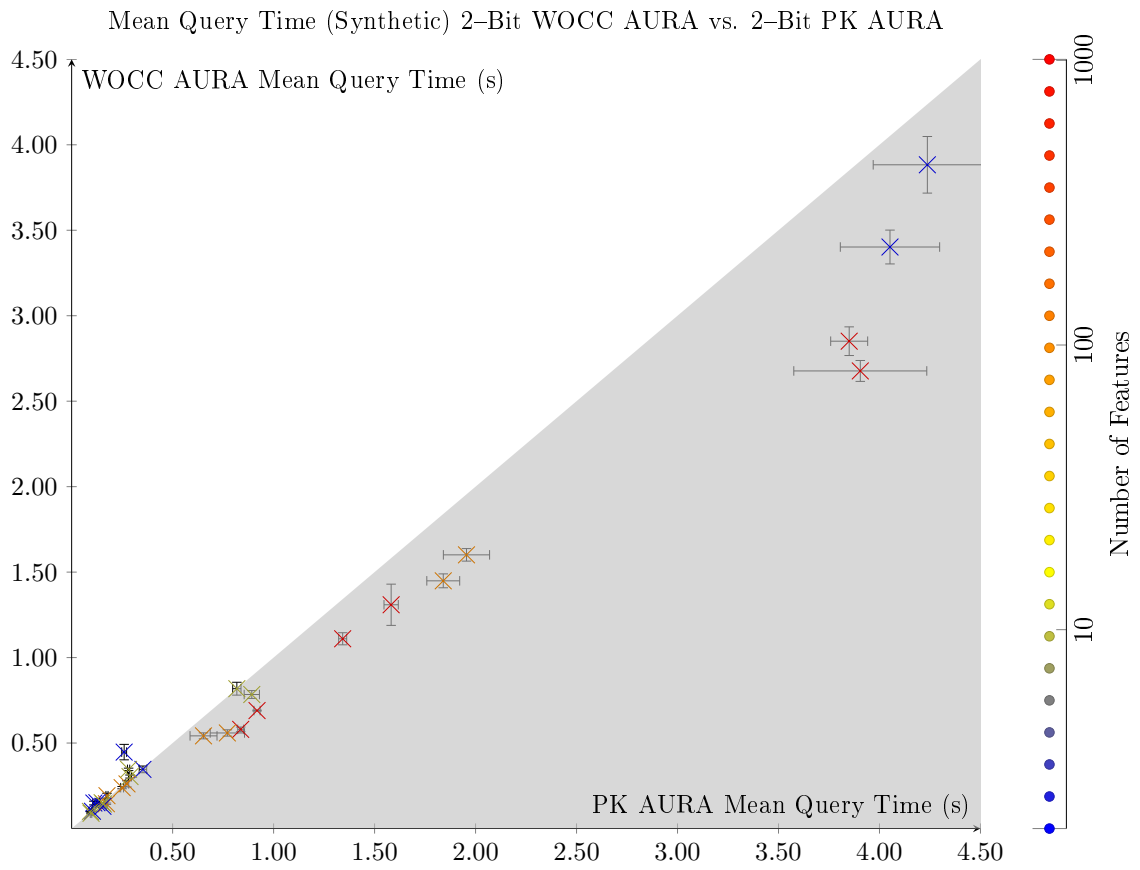


Figure 8.8: A comparison of the mean query time for each of the synthetic datasets. Both the WOCC input tokens and the PK input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. Markers in the shaded area indicate datasets where WOCC AURA is faster than PK AURA.

Figure 8.8 shows a comparison between the mean query times of the dataset with AURA using PK input tokens and AURA using WOCC input tokens with 2–bit output tokens. WOCC tokens appear to be generally faster than PK tokens. The mean query time of WOCC tokens is 5.2% faster. The only datasets where PK is faster than WOCC is on the small datasets where the query requires less than 0.5 seconds to execute.

Training Performance

In this Section the performance of the PK, OBCC and WOCC input tokens is considered with respect to training time. The results of this experiment are presented in Table 8.13.

As with the previous experiments, the Dual KD–Tree algorithm was chosen to form the comparison from the exact algorithms because it was the fastest of the exact algorithms on the large datasets, particularly those with many features.

Table 8.13: Mean Training Time for AURA Input Tokens with 2–Bit Output Tokens (Synthetic)

ID	Features	Samples	Dual KD–Tree	Parabolic	OBCC	WOCC
SYNTH01	2	1,000	0.000	0.002	0.001	0.001
SYNTH02	2	1,000	0.000	0.001	0.000	0.001
SYNTH03	10	1,000	0.001	0.004	0.007	0.006
SYNTH04	10	1,000	0.000	0.004	0.009	0.002
SYNTH05	100	1,000	0.005	0.010	0.071	0.026
SYNTH06	100	1,000	0.002	0.010	0.080	0.038
SYNTH07	1,000	1,000	0.023	0.088	1.070	0.296
SYNTH08	1,000	1,000	0.031	0.082	1.085	0.336
SYNTH09	2	10,000	0.005	0.006	0.004	0.003
SYNTH10	2	10,000	0.003	0.006	0.006	0.005
SYNTH11	10	10,000	0.007	0.037	0.079	0.038
SYNTH12	10	10,000	0.008	0.030	0.115	0.042
SYNTH13	100	10,000	0.058	0.098	0.728	0.254
SYNTH14	100	10,000	0.057	0.101	0.726	0.294
SYNTH15	1,000	10,000	0.534	0.867	10.067	3.005
SYNTH16	1,000	10,000	0.497	0.871	10.025	3.616
SYNTH17	2	100,000	0.057	0.039	0.045	0.060
SYNTH18	2	100,000	0.075	0.043	0.038	0.047
SYNTH19	10	100,000	0.172	0.294	0.858	0.477
SYNTH20	10	100,000	0.182	0.271	0.875	0.545
SYNTH21	100	100,000	1.183	1.163	8.633	2.973
SYNTH22	100	100,000	1.084	1.252	8.390	3.225
SYNTH23	1,000	100,000	13.359	10.061	114.587	37.936
SYNTH24	1,000	100,000	15.501	9.838	111.360	38.851
SYNTH25	2	1,000,000	2.442	0.521	0.437	0.504
SYNTH26	2	1,000,000	2.046	0.495	0.517	0.454
SYNTH27	10	1,000,000	3.165	3.297	10.344	4.637
SYNTH28	10	1,000,000	3.127	2.972	11.864	5.001
SYNTH29	100	1,000,000	24.611	11.616	95.253	32.927
SYNTH30	100	1,000,000	23.537	11.967	92.965	35.202

This table shows the mean training time in seconds for the AURA algorithm with all three input tokens, 2–bit output tokens and 10 bins.

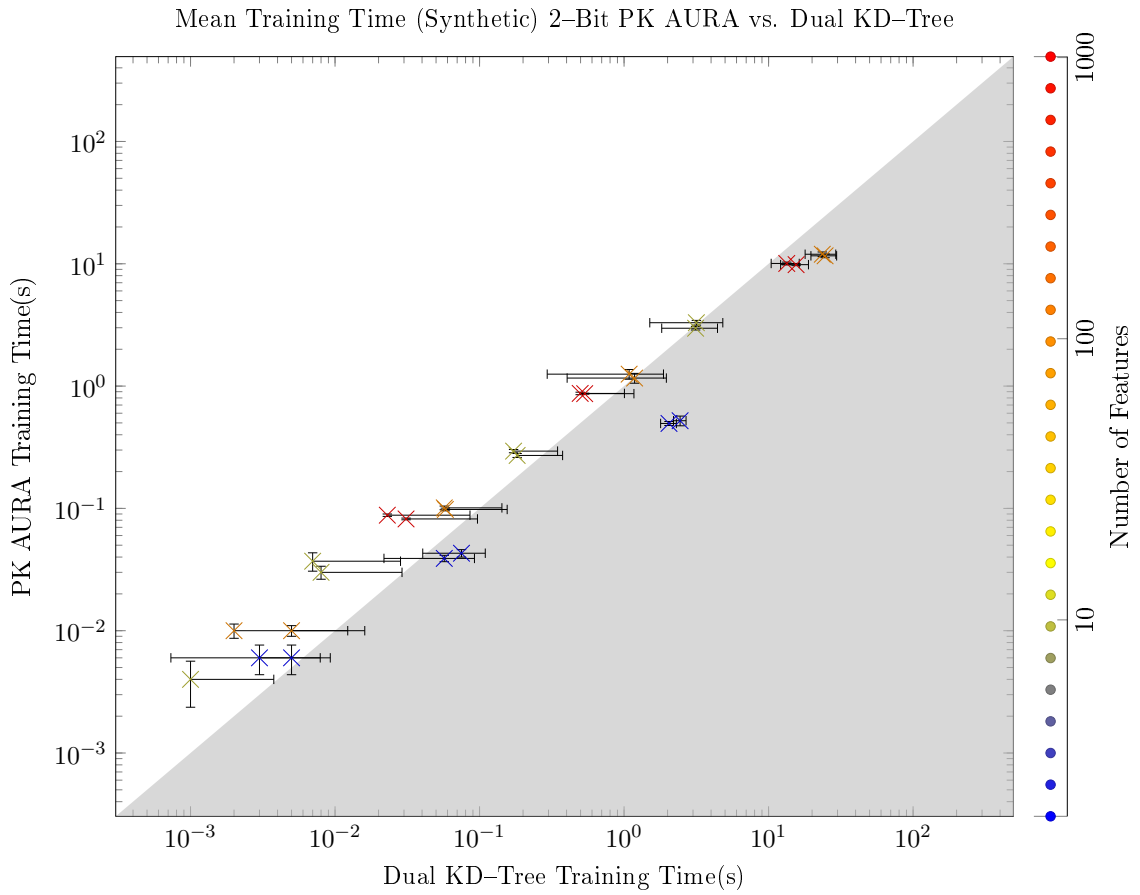


Figure 8.9: A comparison of the mean training time for each of the synthetic datasets. The PK input tokens for this experiment consist of 10 bins. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the PK AURA mean training time is faster.

Figure 8.9 shows a comparison of the mean training time between AURA with PK input tokens and the Dual KD–Tree algorithm. PK is slower than Dual KD–Tree to train the small datasets. However the datasets consisting of 10,000 or more samples are generally faster with PK.

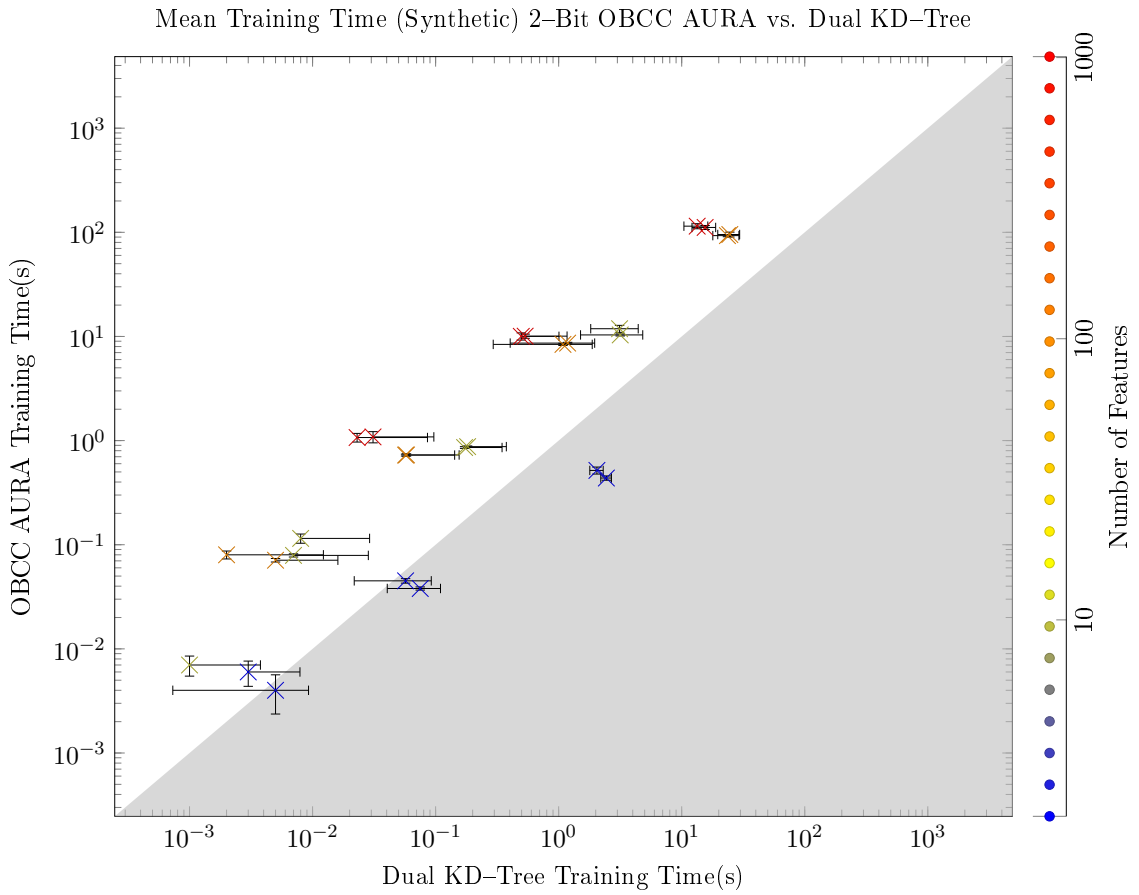


Figure 8.10: A comparison of the mean training time for each of the synthetic datasets. The OBCC input tokens for this experiment consist of 10 bins. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the OBCC AURA mean training time is faster.

Figure 8.10 shows a comparison of the mean training time between AURA with OBCC input tokens and the Dual KD–Tree algorithm. With the exception of the 2–feature datasets, OBCC is considerably slower to train than Dual KD–Tree. It is particularly poor with datasets containing 1000–features, in the worst case taking 47 times longer to train dataset SYNTH07. However as the number of samples being stored increases, the performance of OBCC appears to improve relative to Dual KD–Tree.

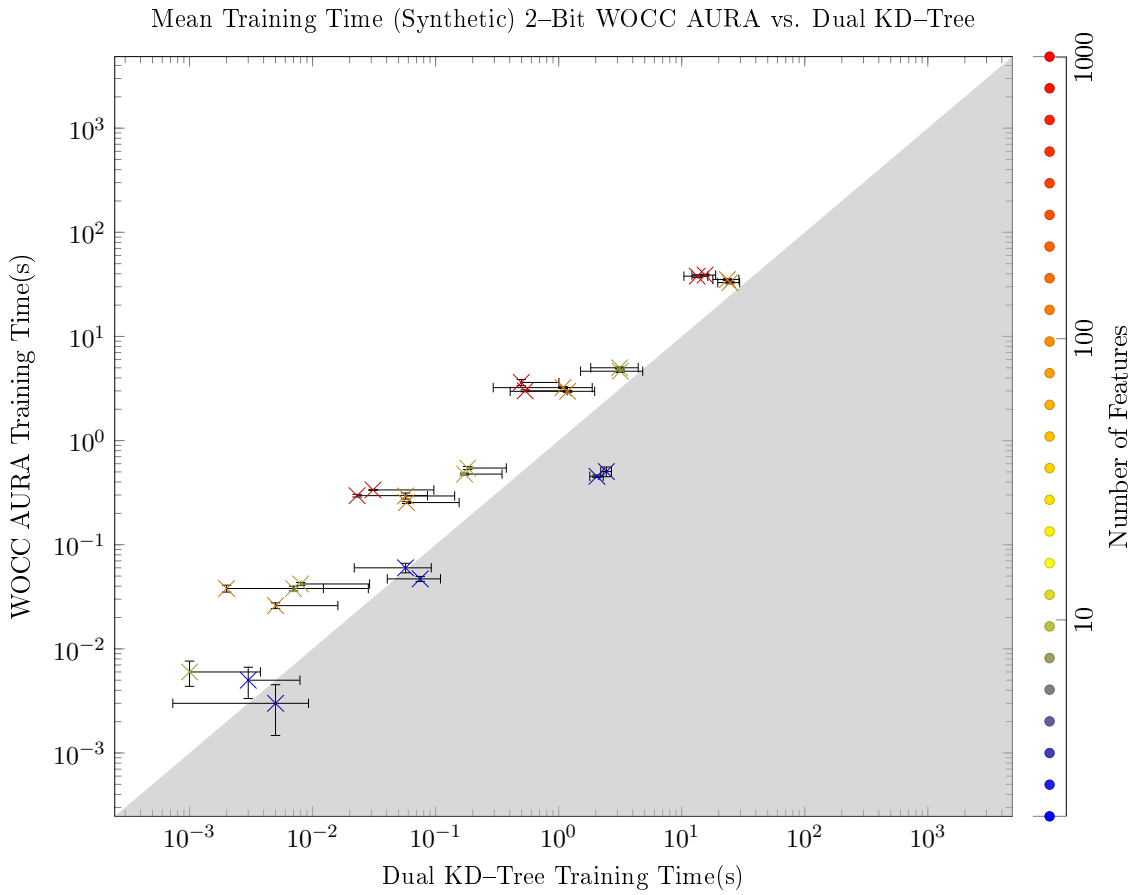


Figure 8.11: A comparison of the mean training time for each of the synthetic datasets. The WOCC input tokens for this experiment consist of 10 bins. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the WOCC AURA mean training time is faster.

Figure 8.11 shows a comparison of the mean training time between AURA with WOCC input tokens and the Dual KD–Tree algorithm. The performance characteristics are similar to those of OBCC in that, with the exception of the 2–feature datasets, WOCC is considerably slower to train than Dual KD–Tree and that the relative performance appears to improve as the size of the datasets increase. However WOCC is generally considerably faster than the OBCC tokens and in the worst case is only 13 times slower to train than Dual KD–Tree.

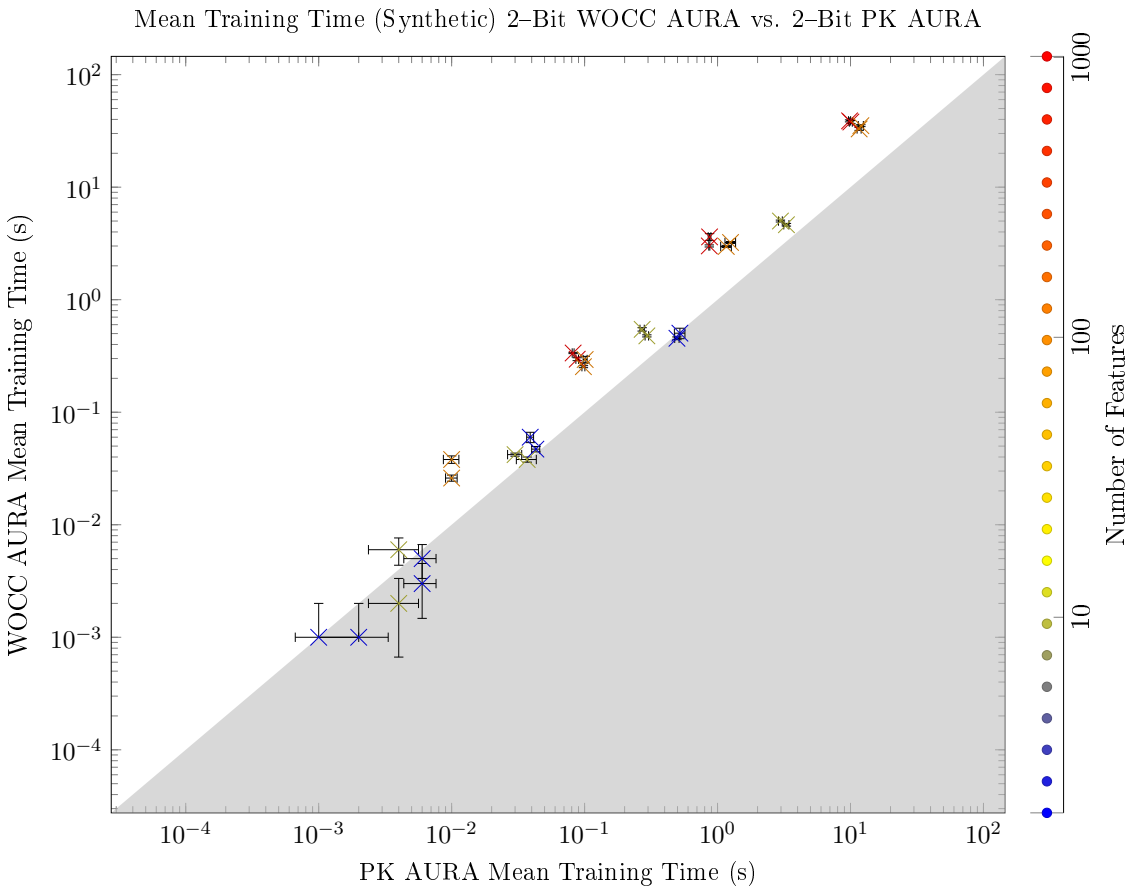


Figure 8.12: A comparison of the mean training time for each of the synthetic datasets. Both the WOCC input tokens and the PK input tokens for this experiment consist of 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. Markers in the shaded area indicate datasets where WOCC AURA is faster than PK AURA.

Figure 8.12 shows a comparison of the mean training time between AURA with WOCC input tokens and the Dual KD–Tree algorithm. It is clear that PK tokens are considerably faster to train than WOCC tokens. The only exceptions to this occur when the total training times are very short. On the mean training time of PK tokens is 2.14 times faster than WOCC tokens.

Evaluation

The accuracy of all three input token methods is relatively similar. The accuracy on the two feature datasets is very poor with all three input token methods and the observed accuracy increases as the number of features in the datasets increases.

This is probably due to the amount of interference caused by superimposing 2 samples in each column of the CMM. If the interference causes an error in the recall of a single feature then this has a disproportionate effect on the datasets with only a small number of features. For example the 1000 feature datasets are more robust to interference because if the contribution of a single feature is corrupted by the interference then there are many other features contributing to the match score that can compensate. For contrast, in a 2 feature dataset, corrupting the contribution of a single feature means that half the retrieved match score will be incorrect.

Another issue with using the 2–bit output tokens is that the observed accuracy decreases as the number of samples stored within the CMM increases, this observation is also consistent across all three input token methods. It seems likely that this is due to the storage capacity of the CMM being exceeded. As the number of samples stored in a CMM approaches the limit of that particular CMM, the probability that an error will occur during a query increases. As a result there is a greater likelihood of an incorrect state being retrieved when more samples are stored within the CMM. This would match the observed increase in mean feature distance, regardless of input token method, as the number of samples increases.

As expected, the accuracy of the OBCC and WOCC tokens is identical. In addition the accuracy of PK tokens is very close to that of the OBCC and WOCC tokens. Even including exception of the 2–feature datasets, for which the accuracy is universally poor, there is no statistically significant difference in accuracy between the input tokens (Dependent T–Test, $t = -1.53$, $p = 0.138 > 0.05$).

With respect to the query time there are two clear observations that can be made from the results of these experiments. The first observation is that the performance characteristics of all three input token methods in comparison to the Dual KD–Tree algorithm is generally very similar. For datasets that are very small or that contain relatively few features, Dual KD–Tree is faster than the AURA based algorithm regardless of the input tokens used. However for the larger datasets, particularly those with 1000 features, the AURA methods are consistently faster by 2 orders of magnitude than the Dual KD–Tree algorithm.

The second observation is that, in contrast to the unary output token experiment in Section 7.2, the WOCC input tokens clearly lead to faster query times than PK input tokens. The OBCC input tokens remain the slowest method.

The reason that WOCC tokens are fastest is likely twofold. Firstly, WOCC tokens in general will require fewer rows of the CMM to be evaluated than the PK tokens. However this was not sufficient to make them faster with unary outputs because the overhead involved with having to decode a denser CMM meant that expensively querying fewer rows was not faster than cheaply querying more rows. The second component to the relative speedup is that with the use of 2–bit output tokens, both the PK and WOCC CMMs will be relatively dense and as a result querying a row from either input token will be more expensive. The relative penalty incurred by WOCC tokens from having more bits set in the CMM is therefore reduced.

The training times for each of the three input token methods are inline with expectations. Storing more bits in the CMM results in slower training times. As a result the PK tokens are the fastest to train, followed by WOCC tokens and OBCC tokens. The use of 2–bit output tokens has not affected the relative performance of the input token methods for training a CMM.

Summary

Using 2–bit output tokens results in poor accuracy for datasets consisting of only a few features. However the accuracy improves with all input token methods as the number of features in a dataset increases. This is offset by a loss of accuracy that is incurred

by increasing the number of samples to be trained. Despite this there is no significant difference in the observed accuracy across all the datasets regardless of whether PK, OBCC or WOCC input tokens are used.

In terms of performance, WOCC tokens consistently provide the fastest query times, however the PK tokens are faster to train. Overall it appears that 2–bit output tokens perform best when combined with WOCC input tokens.

8.3 Conclusion

The characteristics of the CMM based kNN algorithm using 2–bit output tokens have been examined in this Chapter.

In reference to the questions posed in Section 6.2.1 the following observations can be made about the AURA kNN algorithm with 2–bit output tokens.

The number of bins used does not have a statistically significant effect on the accuracy of the algorithm on both the synthetic and real world datasets using either the PK, OBCC and WOCC tokens as input tokens. However increasing the number of bins used had a detrimental effect on the observed query and training times. As a result the optimal strategy for the number of bins remains the same as for unary output tokens, use as few bins as possible without overfilling the discrete states with samples. Unfortunately this means that the ideal number of bins will be dataset dependent.

The results show that the best approach for generating the input tokens for pairing with 2–bit output tokens is the WOCC method because it produces CMMs that can be queried fastest and with no significant loss in accuracy in comparison to the other input token methods. However it should be noted that PK tokens are still faster to train than WOCC tokens.

Overall the CMM based kNN algorithm with 2–bit output tokens has been shown to be substantially faster than the exact Dual KD–Tree algorithm for large datasets, however this speed comes at the cost of accuracy. As a result, provided that the level of accuracy is acceptable, the pairing of WOCC input tokens and 2–bit output tokens can potentially provide a CMM based kNN algorithm that is suitable for AURA Alert.

Chapter 9

AURA 2–bit Comparative Evaluation

9.1 Introduction

9.2 AURA Comparison

The purpose of this Section is to compare the best configurations for the AURA based approximate kNN algorithm. In Chapter 7 it was determined that PK input tokens were best paired with unary output tokens. This pairing currently forms the basis of AURA Alert and will be referred to as the unary configuration. However in Chapter 8 WOCC input token were identified as performing best when paired with 2–bit output tokens, this pairing will be referred to as the 2–bit configuration. Therefore in this Section a comparison between the unary and the 2–bit configuration will be performed in order to identify the situations to which each configuration is best suited.

9.2.1 Accuracy

In this Section a comparison of the accuracy between PK input tokens paired with unary output tokens, the unary configuration, and WOCC input token paired with 2–bit output tokens, the 2–bit configuration, is presented. The results of this experiment are given in Table 9.1.

Table 9.1: Mean Feature Distance AURA Comparison(Synthetic)

ID	Features	Samples	Exact	Unary	2–Bit
SYNTH01	2	1,000	0.062	0.062	0.120
SYNTH02	2	1,000	0.065	0.065	0.117
SYNTH03	10	1,000	0.142	0.149	0.249
SYNTH04	10	1,000	0.124	0.128	0.214
SYNTH05	100	1,000	0.071	0.072	0.081
SYNTH06	100	1,000	0.061	0.062	0.070
SYNTH07	1,000	1,000	0.025	0.025	0.026
SYNTH08	1,000	1,000	0.021	0.022	0.022
SYNTH09	2	10,000	0.019	0.020	0.203
SYNTH10	2	10,000	0.022	0.023	0.182
SYNTH11	10	10,000	0.107	0.110	0.246
SYNTH12	10	10,000	0.096	0.099	0.213
SYNTH13	100	10,000	0.067	0.069	0.081
SYNTH14	100	10,000	0.058	0.059	0.070
SYNTH15	1,000	10,000	0.024	0.025	0.026
SYNTH16	1,000	10,000	0.021	0.021	0.022
SYNTH17	2	100,000	0.006	0.006	0.464
SYNTH18	2	100,000	0.007	0.008	0.411
SYNTH19	10	100,000	0.082	0.084	0.243
SYNTH20	10	100,000	0.076	0.080	0.213
SYNTH21	100	100,000	0.064	0.066	0.081
SYNTH22	100	100,000	0.055	0.057	0.070
SYNTH23	1,000	100,000	0.024	0.024	0.026
SYNTH24	1,000	100,000	0.021	0.021	0.022
SYNTH25	2	1,000,000	0.002	0.002	0.475
SYNTH26	2	1,000,000	0.002	0.002	0.377
SYNTH27	10	1,000,000	0.063	0.066	0.250
SYNTH28	10	1,000,000	0.060	0.065	0.210
SYNTH29	100	1,000,000	0.062	0.064	0.081
SYNTH30	100	1,000,000	0.053	0.055	0.070

This table shows the mean feature distance for the AURA algorithm with two different configurations. The Unary configuration consists of PK input tokens and unary output tokens and the 2–bit configuration consists of WOCC input tokens with 2–bit output tokens. Both configurations use 10 bins.

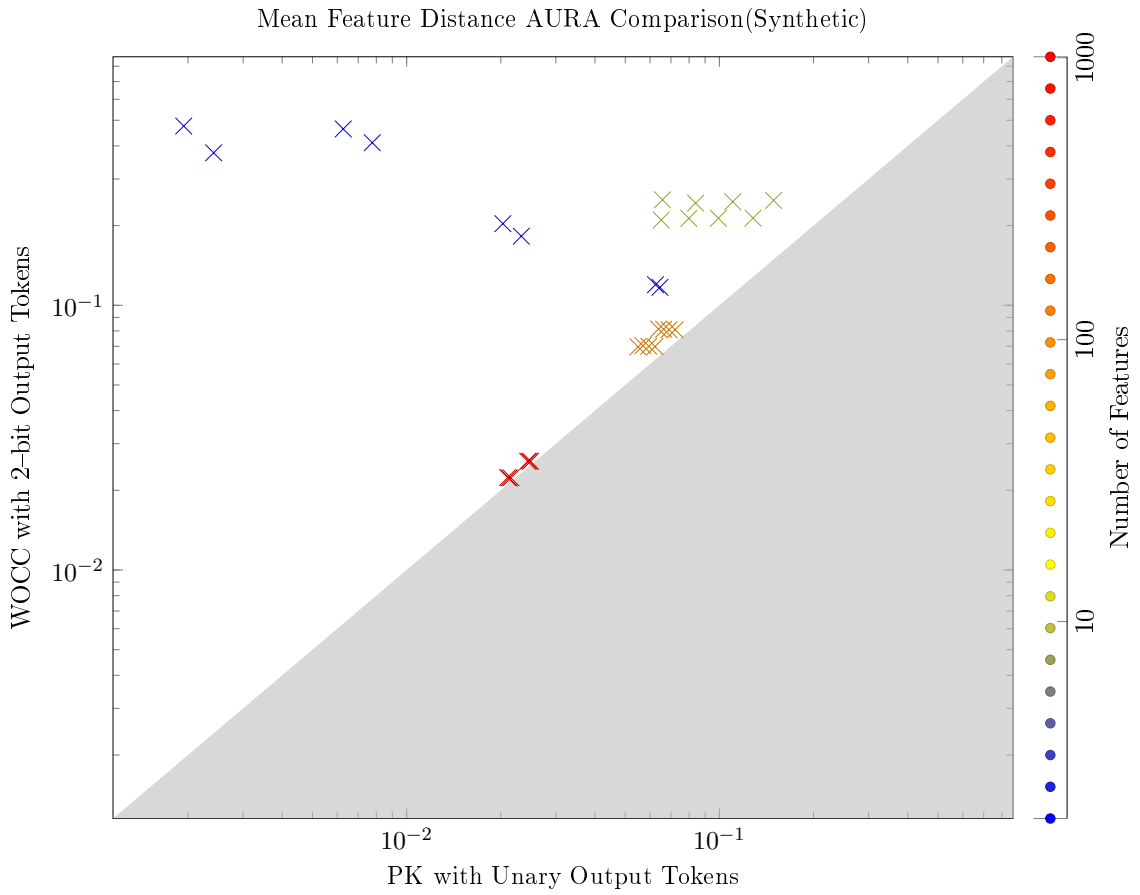


Figure 9.1: A comparison of the mean feature distance using AURA with two different configurations for each of the synthetic datasets. The Unary configuration consists of PK input tokens and unary output tokens and the 2–bit configuration consists of WOCC input tokens with 2–bit output tokens. Both configurations use 10 bins. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the 2–bit configuration is more accurate.

Figure 9.1 shows a comparison of the mean feature distance between the unary configuration and the 2–bit configuration. As expected, it is clear that the unary configuration is considerably more accurate than the 2–bit configurations. Across all the synthetic datasets, the mean feature distance of the 2–bit configuration is 19.5 times larger than than the unary configuration. However the 2–bit configuration performs comparably with the unary configuration on the 1000 feature datasets. Here the difference in mean feature distance is only 4.6% in favour of the unary configuration.

9.2.2 Query Performance

In this Section a comparison of the query time between PK input tokens paired with unary output tokens, the unary configuration, and WOCC input token paired with 2–bit output tokens, the 2–bit configuration, is presented. The results of this experiment are given in Table 9.2.

Table 9.2: Mean Query Time AURA Comparison(Synthetic)

ID	Features	Samples	Exact	Unary	2–Bit
SYNTH01	2	1,000	0.123	0.096	0.101
SYNTH02	2	1,000	0.139	0.097	0.149
SYNTH03	10	1,000	0.154	0.127	0.089
SYNTH04	10	1,000	0.160	0.112	0.100
SYNTH05	100	1,000	0.438	0.210	0.145
SYNTH06	100	1,000	0.466	0.216	0.192
SYNTH07	1,000	1,000	3.585	1.142	0.580
SYNTH08	1,000	1,000	3.519	1.314	0.690
SYNTH09	2	10,000	1.154	0.135	0.130
SYNTH10	2	10,000	1.236	0.118	0.151
SYNTH11	10	10,000	1.411	0.656	0.143
SYNTH12	10	10,000	1.445	0.621	0.151
SYNTH13	100	10,000	4.524	1.459	0.239
SYNTH14	100	10,000	4.624	1.561	0.261
SYNTH15	1,000	10,000	34.654	12.455	1.110
SYNTH16	1,000	10,000	34.189	12.900	1.309
SYNTH17	2	100,000	11.931	0.390	0.447
SYNTH18	2	100,000	11.862	0.465	0.346
SYNTH19	10	100,000	14.250	6.636	0.304
SYNTH20	10	100,000	14.834	6.071	0.346
SYNTH21	100	100,000	44.913	17.344	0.542
SYNTH22	100	100,000	44.026	18.543	0.559
SYNTH23	1,000	100,000	345.050	119.093	2.677
SYNTH24	1,000	100,000	352.514	119.534	2.851
SYNTH25	2	1,000,000	116.661	4.385	3.883
SYNTH26	2	1,000,000	116.297	6.132	3.402
SYNTH27	10	1,000,000	143.728	65.392	0.784
SYNTH28	10	1,000,000	144.761	66.379	0.818
SYNTH29	100	1,000,000	450.798	178.781	1.449
SYNTH30	100	1,000,000	433.448	182.917	1.601

This table shows the mean query time in seconds for the AURA algorithm with two different configurations. Unary consists of PK input tokens and unary output tokens and 2–Bit consists of WOCC input tokens with 2–bit output tokens. Both configurations use 10 bins.

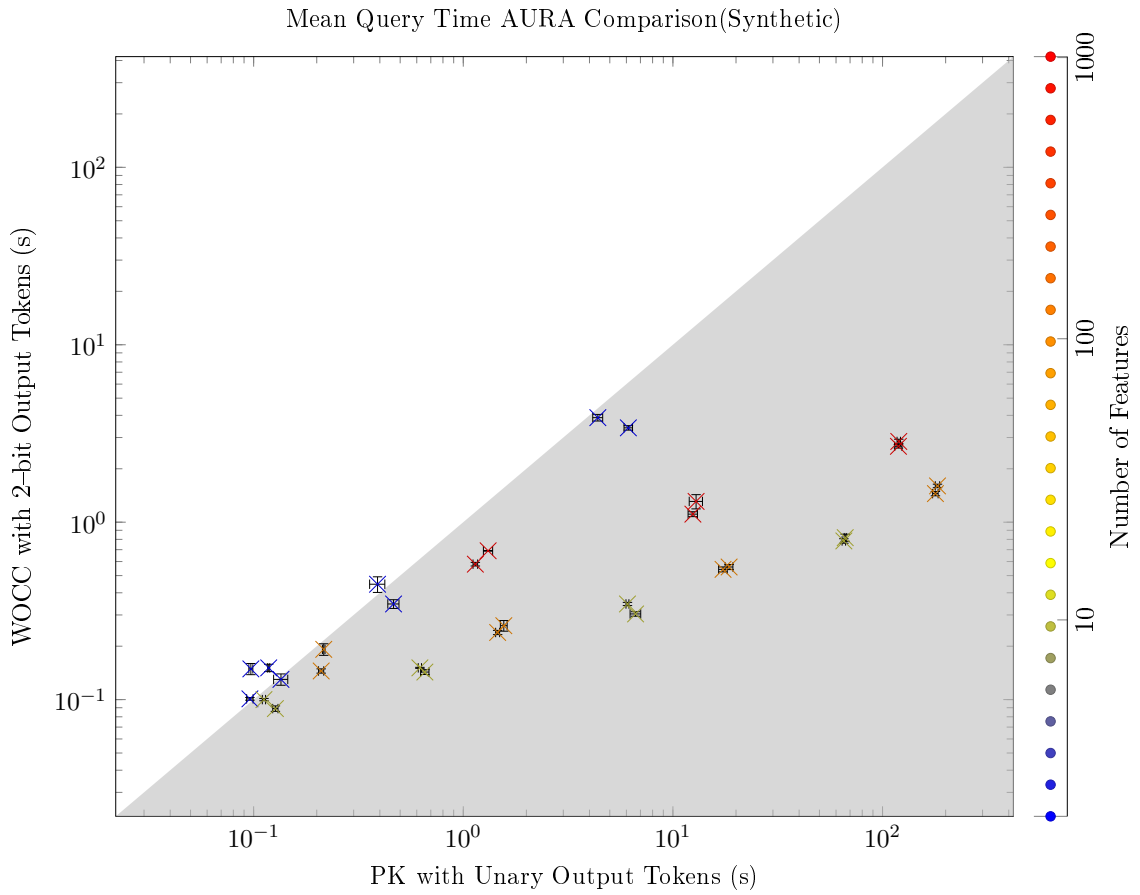


Figure 9.2: A comparison of the mean query time using AURA with two different configurations for each of the synthetic datasets. The Unary configuration consists of PK input tokens and unary output tokens and the 2–bit configuration consists of WOCC input tokens with 2–bit output tokens. Both configurations use 10 bins. Each marker represents the mean query time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the 2–bit configuration is faster.

Figure 9.2 shows a comparison of the mean query time between the unary configuration and the 2–bit configuration. As expected, it is clear that the 2–bit configuration is considerably faster than the unary configuration. In addition the relative performance advantage of the 2–bit configuration increases as the size of the dataset increases. The 2–bit configuration was a mean of 67.5 times faster than the unary configuration for the largest datasets of 1,000,000 sample datasets. This advantage is even greater when considering only the largest datasets with 1000 features where it is 118 times faster.

9.2.3 Training Performance

In this Section a comparison of the training time between PK input tokens paired with unary output tokens, the unary configuration, and WOCC input token paired with 2–bit output tokens, the 2–bit configuration, is presented. The results of this experiment are given in Table 9.3.

Table 9.3: Mean Training Time AURA Comparison(Synthetic)

ID	Features	Samples	Exact	Unary	2–Bit
SYNTH01	2	1,000	0.000	0.000	0.001
SYNTH02	2	1,000	0.000	0.000	0.001
SYNTH03	10	1,000	0.000	0.004	0.006
SYNTH04	10	1,000	0.000	0.003	0.002
SYNTH05	100	1,000	0.000	0.012	0.026
SYNTH06	100	1,000	0.000	0.011	0.038
SYNTH07	1,000	1,000	0.000	0.119	0.296
SYNTH08	1,000	1,000	0.000	0.128	0.336
SYNTH09	2	10,000	0.000	0.004	0.003
SYNTH10	2	10,000	0.000	0.007	0.005
SYNTH11	10	10,000	0.000	0.024	0.038
SYNTH12	10	10,000	0.000	0.024	0.042
SYNTH13	100	10,000	0.000	0.129	0.254
SYNTH14	100	10,000	0.000	0.123	0.294
SYNTH15	1,000	10,000	0.000	1.330	3.005
SYNTH16	1,000	10,000	0.000	1.249	3.616
SYNTH17	2	100,000	0.000	0.042	0.060
SYNTH18	2	100,000	0.000	0.043	0.047
SYNTH19	10	100,000	0.000	0.295	0.477
SYNTH20	10	100,000	0.000	0.255	0.545
SYNTH21	100	100,000	0.000	1.353	2.973
SYNTH22	100	100,000	0.000	1.327	3.225
SYNTH23	1,000	100,000	0.000	12.703	37.936
SYNTH24	1,000	100,000	0.000	11.536	38.851
SYNTH25	2	1,000,000	0.000	0.455	0.504
SYNTH26	2	1,000,000	0.000	0.529	0.454
SYNTH27	10	1,000,000	0.000	2.668	4.637
SYNTH28	10	1,000,000	0.000	2.728	5.001
SYNTH29	100	1,000,000	0.000	15.489	32.927
SYNTH30	100	1,000,000	0.000	13.033	35.202

This table shows the mean train time in seconds for the AURA algorithm with two different configurations. Unary consists of PK input tokens and unary output tokens and 2–Bit consists of WOCC input tokens with 2–bit output tokens. Both configurations use 10 bins.

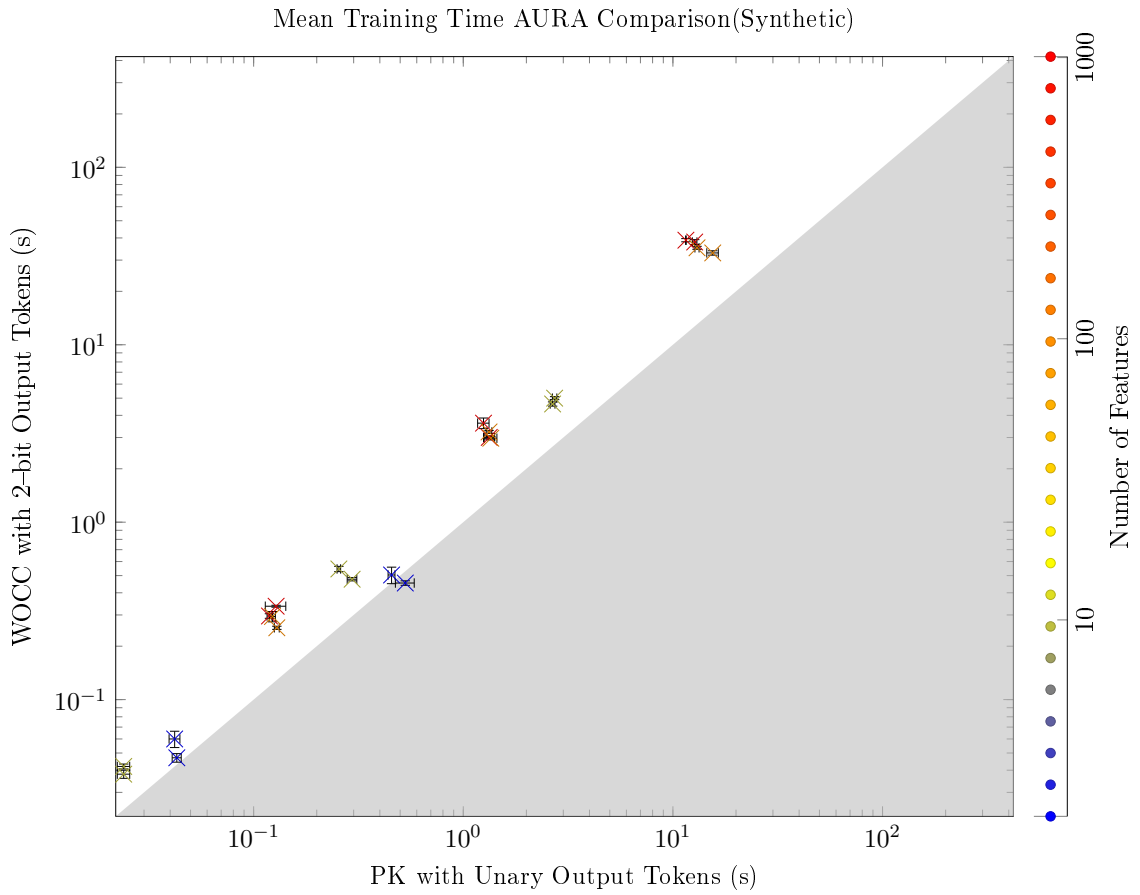


Figure 9.3: A comparison of the mean training time using AURA with two different configurations for each of the synthetic datasets. The Unary configuration consists of PK input tokens and unary output tokens and the 2–bit configuration consists of WOCC input tokens with 2–bit output tokens. Both configurations use 10 bins. Each marker represents the mean training time for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where the 2–bit configuration is faster.

Figure 9.3 shows a comparison of the mean feature distance between the unary configuration and the 2–bit configuration. The unary configuration is clearly much faster to train, across all the datasets it is a mean of 1.95 times faster to train the dataset with the unary configuration than the 2–bit configuration.

9.2.4 Evaluation

Overall the accuracy of the unary configuration is superior to that of the 2–bit configuration. There are two reasons that this could be to case, first the PK tokens are more accurate than WOCC tokens when paired with unary input tokens. As a result it is expected that the unary configuration which uses PK input tokens would be more accurate. The second reason is that the use of 2–bit output tokens results in multiple samples being superimposed on a every column of the CMM. The interference cause by this superimposition will potentially introduce errors in the CMM recall and therefore result in a reduction in the overall accuracy of a query. It is because the PK tokens suffer more greatly from this interference that the WOCC tokens provide an equivalent level of accuracy when paired with 2–bit output tokens. As a result, the superior accuracy of the unary configuration falls in line with expectations. However it is important that the relative accuracy of the

datasets with 1000 features is very close because this suggests that 2–bit configuration may be a superior choice in these situations, especially when the query performance is considered.

The 2–bit configuration has significantly faster query times than the unary configuration. This is primarily due to the size of the CMM that has to be queried. Since the 2–bit configuration enables multiple samples to be superimposed within the CMM, the CMM can be constructed to store the same number of samples with considerably fewer columns than with the unary configuration. In addition the rate at which the CMM is required to grow in order to handle larger datasets is much smaller with the 2–bit configuration. This is because the unary configuration requires an additional column for each new state that is to be stored whereas the 2–bit configuration can store multiple states for each additional column, the number of new states that can be stored in the 2–bit configuration depends on the length of the Baum section used and these sections increase in length as the total number of columns in the CMM increases, this was examined in Section 5.4. The end result is that the 2–bit configuration is faster than the unary configuration and that the relative performance advantage increases with larger datasets.

However with respect to the training time, the 2–bit configuration is roughly twice as slow as the unary configuration. This is simply due to the need to set more bits within the CMM with the 2–bit configuration. Both the input tokens and the output tokens from the 2–bit configuration require more bits and so combining them provides the huge number of extra bits that have to be set in the CMM and therefore the performance penalty.

Overall, when making a decision between whether to use the unary or 2–bit configuration it is first necessary to determine whether the level of accuracy provided by the 2–bit configuration is sufficient. If the accuracy is considered to be sufficient for the expected dataset then the major advantage in the query time makes the 2–bit configuration the clear choice.

9.2.5 Summary

The unary configuration, PK input tokens paired with Unary output tokens, is both faster to train than the 2–bit configuration, WOCC input tokens paired with 2–bit output tokens, and more accurate than the 2–bit configuration.

However the 2–bit configuration has a significantly faster query time as a result of requiring fewer columns in the CMM. The time required to query a dataset as the size of the dataset increases also grows much more slowly in the 2–bit configuration and as a result the 2–bit configuration is 118 times faster to query the largest synthetic datasets.

9.3 LSH Comparison

The purpose of this Section is to compare the 2–bit configuration, WOCC input tokens paired with 2–bit output tokens, for the AURA based approximate kNN algorithm with the LSH algorithm.

In Chapter 7 it was determined that the best results when using unary input tokens could be obtained with PK input tokens. However this configuration of tokens was shown to have a slower query time than the exact Dual KD–Tree algorithm and as a result it

was not necessary to perform a comparison between the LSH algorithm and the AURA method because LSH has been shown in Section 6.4 to be substantially faster than the Dual KD–Tree method in most instances.

In Section 9.2 an alternative pairing of WOCC input tokens and 2–bit output tokens was shown to have substantially faster query times than could be observed using AURA with unary output tokens.

As a result, the purpose of this Section is compare the performance of the modified AURA kNN algorithm using the 2–bit configuration with the LSH algorithm, currently the fastest approximate kNN algorithm.

9.3.1 Accuracy

In this Section a comparison of the accuracy between the AURA kNN implementation using WOCC input tokens paired with 2–bit output tokens and the LSH algorithm is presented. The results of this experiment are given in Table 9.4.

Table 9.4: Mean Feature Distance LSH AURA Comparison(Synthetic)

ID	Features	Samples	LSH	2–Bit AURA
SYNTH01	2	1,000	0.063	0.120
SYNTH02	2	1,000	0.065	0.117
SYNTH03	10	1,000	0.144	0.249
SYNTH04	10	1,000	0.125	0.214
SYNTH05	100	1,000	0.073	0.081
SYNTH06	100	1,000	0.062	0.070
SYNTH07	1,000	1,000	0.025	0.026
SYNTH08	1,000	1,000	0.021	0.022
SYNTH09	2	10,000	0.019	0.203
SYNTH10	2	10,000	0.022	0.182
SYNTH11	10	10,000	0.111	0.246
SYNTH12	10	10,000	0.099	0.213
SYNTH13	100	10,000	0.071	0.081
SYNTH14	100	10,000	0.061	0.070
SYNTH15	1,000	10,000	0.025	0.026
SYNTH16	1,000	10,000	0.021	0.022
SYNTH17	2	100,000	0.006	0.464
SYNTH18	2	100,000	0.007	0.411
SYNTH19	10	100,000	0.086	0.243
SYNTH20	10	100,000	0.080	0.213
SYNTH21	100	100,000	0.070	0.081
SYNTH22	100	100,000	0.060	0.070
SYNTH23	1,000	100,000	0.025	0.026
SYNTH24	1,000	100,000	0.021	0.022
SYNTH25	2	1,000,000	0.002	0.475
SYNTH26	2	1,000,000	0.002	0.377
SYNTH27	10	1,000,000	0.067	0.250
SYNTH28	10	1,000,000	0.065	0.210
SYNTH29	100	1,000,000	0.069	0.081
SYNTH30	100	1,000,000	0.060	0.070

This table shows the mean feature distance for the AURA algorithm with WOCC input tokens and 2–bit input tokens compared with the LSH algorithm.

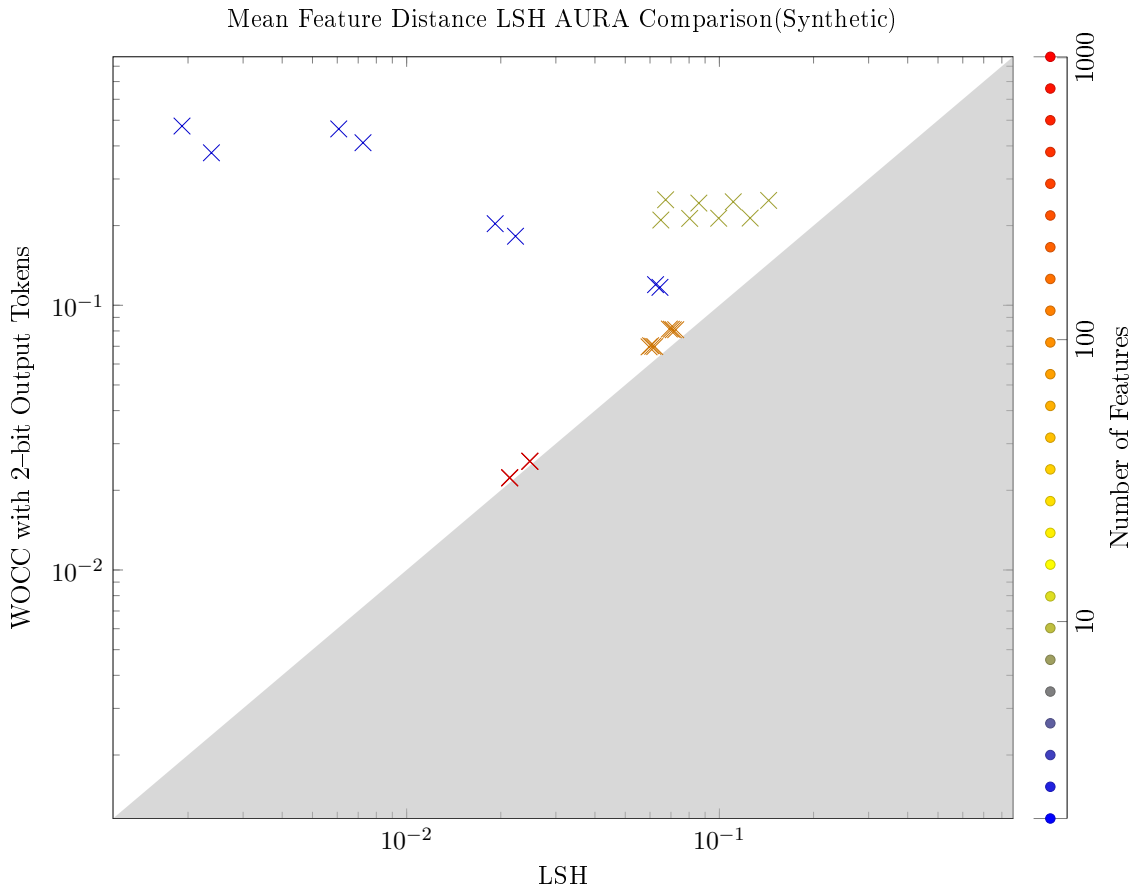


Figure 9.4: A comparison of the mean feature distance between the AURA algorithm with WOCC input tokens and 2–bit output tokens and the LSH algorithm. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where AURA is more accurate.

Figure 9.4 shows a comparison of the mean feature distance between AURA kNN using WOCC input tokens and 2–bit output tokens and LSH. Given the poor accuracy of this AURA configuration when applied to datasets with a small number of features, as observed in Chapter 8, and the generally high accuracy of LSH across all datasets, as observed in Section 6.4, the results of this comparison are unsurprising. The LSH algorithm is significantly more accurate on the datasets with few features however the gap between the accuracy of both methods narrows as the number of features in the datasets increases. The difference between mean feature distances observed with the 1000 feature datasets is only 4%. Regardless LSH is strictly more accurate across all the synthetic datasets. With regards to the real world datasets, the LSH algorithm is also more accurate. The mean feature distance across all the real world datasets is 28.4% lower with LSH than AURA kNN.

9.3.2 Query Performance

In this Section a comparison of the query time between the AURA kNN implementation using WOCC input tokens paired with 2–bit output tokens and the LSH algorithm is presented. The relative query performance of these two algorithms is unknown. The results of this experiment are given in Table 9.5.

Table 9.5: Mean Query Time LSH AURA Comparison(Synthetic)

ID	Features	Samples	LSH	2–Bit AURA
SYNTH01	2	1,000	0.068	0.101
SYNTH02	2	1,000	0.075	0.149
SYNTH03	10	1,000	0.076	0.089
SYNTH04	10	1,000	0.073	0.100
SYNTH05	100	1,000	0.143	0.145
SYNTH06	100	1,000	0.137	0.192
SYNTH07	1,000	1,000	1.818	0.580
SYNTH08	1,000	1,000	1.973	0.690
SYNTH09	2	10,000	0.089	0.130
SYNTH10	2	10,000	0.086	0.151
SYNTH11	10	10,000	0.116	0.143
SYNTH12	10	10,000	0.124	0.151
SYNTH13	100	10,000	0.233	0.239
SYNTH14	100	10,000	0.241	0.261
SYNTH15	1,000	10,000	1.749	1.110
SYNTH16	1,000	10,000	1.889	1.309
SYNTH17	2	100,000	0.095	0.447
SYNTH18	2	100,000	0.108	0.346
SYNTH19	10	100,000	0.145	0.304
SYNTH20	10	100,000	0.165	0.346
SYNTH21	100	100,000	0.294	0.542
SYNTH22	100	100,000	0.290	0.559
SYNTH23	1,000	100,000	1.919	2.677
SYNTH24	1,000	100,000	1.857	2.851
SYNTH25	2	1,000,000	0.110	3.883
SYNTH26	2	1,000,000	0.123	3.402
SYNTH27	10	1,000,000	0.201	0.784
SYNTH28	10	1,000,000	0.190	0.818
SYNTH29	100	1,000,000	0.308	1.449
SYNTH30	100	1,000,000	0.300	1.601

This table shows the mean query time in seconds for the AURA algorithm with WOCC input tokens and 2–bit input tokens compared with the LSH algorithm.

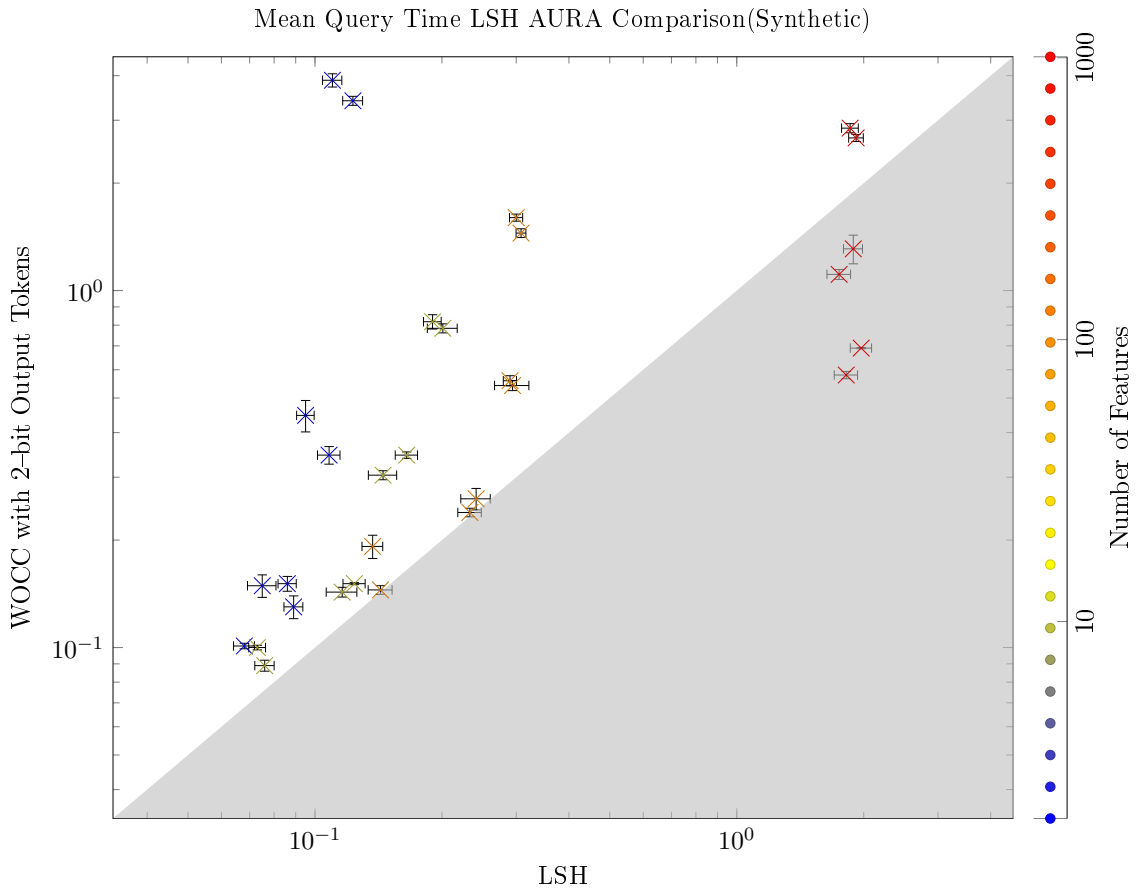


Figure 9.5: A comparison of the mean query time in seconds between the AURA algorithm with WOCC input tokens and 2–bit output tokens and the LSH algorithm. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where AURA is faster.

Figure 9.5 shows a comparison of the mean query time between AURA kNN using WOCC input tokens and 2–bit output tokens and LSH. In general the LSH algorithm is faster than AURA, particularly with the largest of the datasets. Overall the mean query time of the LSH algorithm is 22.9% smaller than the AURA mean query time. However with the smaller datasets consisting of 1000 features AURA is actually faster than LSH. In the best case dataset SYNTH07 can be queried 3.1 times faster with AURA than LSH. However as the number of samples in the dataset increases, LSH becomes faster than AURA.

9.3.3 Training Performance

In this Section a comparison of the training time between the AURA kNN implementation using WOCC input tokens paired with 2–bit output tokens and the LSH algorithm is presented. The relative query performance of these two algorithms is unknown. The results of this experiment are given in Table 9.6.

Table 9.6: Mean Training Time LSH AURA Comparison(Synthetic)

ID	Features	Samples	LSH	Dual AURA
SYNTH01	2	1,000	0.032	0.001
SYNTH02	2	1,000	0.034	0.001
SYNTH03	10	1,000	0.055	0.006
SYNTH04	10	1,000	0.048	0.002
SYNTH05	100	1,000	0.191	0.026
SYNTH06	100	1,000	0.209	0.038
SYNTH07	1,000	1,000	0.365	0.296
SYNTH08	1,000	1,000	0.405	0.336
SYNTH09	2	10,000	0.351	0.003
SYNTH10	2	10,000	0.331	0.005
SYNTH11	10	10,000	0.655	0.038
SYNTH12	10	10,000	0.734	0.042
SYNTH13	100	10,000	3.762	0.254
SYNTH14	100	10,000	3.932	0.294
SYNTH15	1,000	10,000	20.380	3.005
SYNTH16	1,000	10,000	23.184	3.616
SYNTH17	2	100,000	3.692	0.060
SYNTH18	2	100,000	4.200	0.047
SYNTH19	10	100,000	8.848	0.477
SYNTH20	10	100,000	10.319	0.545
SYNTH21	100	100,000	60.299	2.973
SYNTH22	100	100,000	66.571	3.225
SYNTH23	1,000	100,000	387.808	37.936
SYNTH24	1,000	100,000	398.625	38.851
SYNTH25	2	1,000,000	49.638	0.504
SYNTH26	2	1,000,000	57.752	0.454
SYNTH27	10	1,000,000	176.161	4.637
SYNTH28	10	1,000,000	188.207	5.001
SYNTH29	100	1,000,000	1,055.715	32.927
SYNTH30	100	1,000,000	1,042.559	35.202

This table shows the mean training time in seconds for the AURA algorithm with WOCC input tokens and 2–bit input tokens compared with the LSH algorithm.

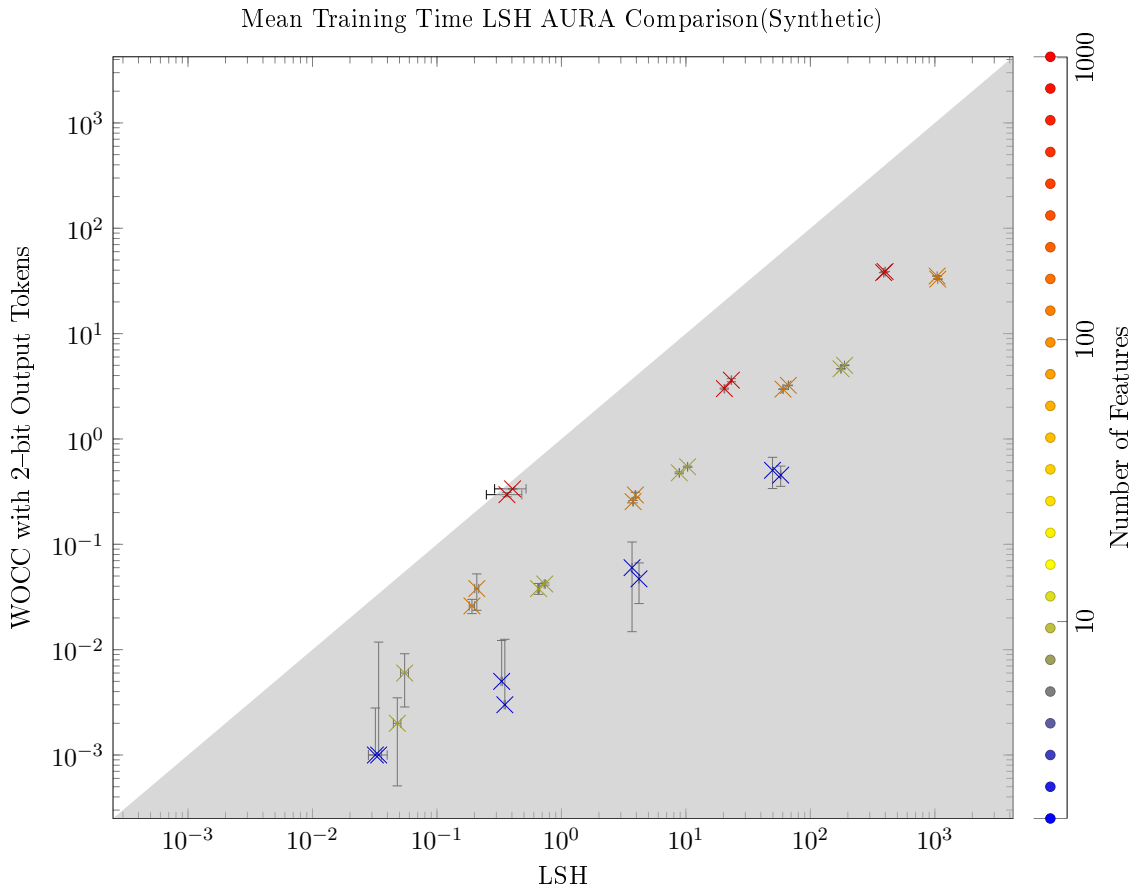


Figure 9.6: A comparison of the mean training time in seconds between the AURA algorithm with WOCC input tokens and 2–bit output tokens and the LSH algorithm. Each marker represents the mean feature distance for a single synthetic dataset, the markers are coloured according to the number of features in the dataset. A marker in the shaded area indicates a dataset where AURA is faster.

Figure 9.5 shows a comparison of the mean query time between AURA kNN using WOCC input tokens and 2–bit output tokens and LSH. In Section 6.4 the training time of the LSH algorithm was identified as its major weakness. This is illustrated once again as the AURA algorithm is significantly faster to train for nearly every dataset. In the case of the 1000 feature datasets when the accuracy and query times of AURA are comparable to LSH, AURA can be trained a mean of 6.02 times faster than LSH.

9.3.4 Evaluation

The accuracy of AURA using WOCC input tokens and 2–bit output tokens is very poor for datasets with only a small number of features. This was discussed in Section 8.2. In contrast the accuracy of the LSH algorithm is consistently good with respect to the exact algorithms as demonstrated in Section 6.4. As a result it is unsurprising that when comparing these two algorithms the results are very similar to those observed between this AURA configuration and the exact algorithms.

Additionally the comparison between training times for each algorithm showed that the AURA configuration was much faster to train than LSH. The AURA configuration used is not the fastest configuration for training times. However the training performance of LSH was identified as the major weakness in Section 6.4 and it is unsurprising that this AURA

configuration is able to train the datasets faster.

In contrast to the accuracy and training time comparisons, the results of the comparison between query times were unexpected. In the majority of the datasets LSH is faster than AURA, however for the 1000 feature datasets with 10,000 samples or less AURA is a mean of 3.1 times faster. These datasets are also the ones in which the accuracy of AURA is comparable to that of the LSH algorithms. As a result it appears that this AURA configuration is potentially a better choice than LSH for datasets with these characteristics. However the query times of the LSH algorithm grow much more slowly than those of AURA, so as a result LSH is faster for the 100,000 sample datasets with 1000 features.

Overall the LSH algorithm is more accurate than this configuration of AURA and in most cases has faster query times. As a result, from a technical perspective, the LSH algorithm would be a superior choice to this AURA configuration for the kNN component of AURA Alert.

9.3.5 Summary

In this Section a comparison between AURA using WOCC input tokens and 2–bit output tokens and the LSH algorithm was presented. In general LSH is superior to AURA with respect to both its query times and its accuracy. Across all the synthetic datasets LSH is consistently more accurate and is generally faster to query. However the main weakness of LSH is the training time which is consistently slower than AURA. Despite this the query time and accuracy benefits provided by LSH mean that, without considering business factors, LSH would be a better choice for the kNN implementation within AURA Alert.

The major exception to these findings is that for the datasets of 1000 features and 10,000 features or less AURA is both faster to train, faster to query and relatively accurate in comparison the LSH algorithm. If the datasets to be evaluated by AURA Alert fell within these parameters then the case can be made for using this AURA configuration despite the small accuracy loss. However, at the moment AURA Alert is not typically used for datasets with these properties.

9.4 Conclusion

In this chapter, the AURA approximate kNN algorithm has been evaluated with respect to the standard kNN algorithms reviewed in Section 3.11.

Initially the baseline experiments were performed using the exact Linear, KD Tree, and Dual KD Tree algorithms and the approximate LSH algorithm. The results of these experiments showed that for datasets with a very small number of features, KD Tree is the fastest algorithm. While, for very large datasets, the Dual KD Tree algorithm is the fastest exact algorithm, the approximate LSH algorithm was shown to be massively faster than any of the exact algorithms for all the synthetic datasets with more than 2 features. Despite the speed of LSH, it is also relatively accurate, with the penalty to the mean feature distance across all the synthetic datasets being only 3.54%.

The results of these baseline experiments match the expected results suggested by the relevant literature and therefore validates the experimental setup for use in the evaluation of AURA kNN.

The first stage of this evaluation was to compare the standard AURA kNN using Unary output tokens. Three methods for generating input tokens, Parabolic Kernel (PK), Overlapped Binary Code Construction (OBCC) and Weighted Overlapped Code Construction (WOCC) were compared with respect to the accuracy of the kNN query, and the execution times for querying and training the algorithm.

The main finding of these experiments was that PK tokens, the standard input tokens used by AURA kNN, were both faster and more accurate than the other input tokens when paired with unary output tokens. Despite this, the exact Dual KD Tree algorithm is still faster than AURA kNN with PK input tokens and Unary output tokens.

With regards to the OBCC and WOCC input tokens, it was shown that the WOCC tokens exhibit identical accuracy to the OBCC tokens and that WOCC tokens are generally faster when both training and querying a CMM.

The next set of experiments considered the use of 2–bit Baum coded output tokens to speed up the query time of the AURA kNN algorithm. Compared to the other input tokens, the accuracy of PK tokens suffers greatly with 2–bit output tokens. As a result, there is no significant difference in accuracy between the three input token methods.

When dataset only have a small number of features, the observed accuracy is very poor. For datasets with many features however, there is only a small penalty in the mean feature distance in comparison to the exact results. Another observation of the 2–bit output experiments is that accuracy appears to decrease as the size of the dataset increases.

However the 2–bit output tokens achieve their objective of improving the query time of the algorithm. Overall WOCC tokens were shown to provide the fastest query times when combined with 2–bit output tokens. This combination of tokens improves the query time by a factor of 67.5 for the largest dataset examined. However this comes at a cost of accuracy. In the best case, where the datasets have a large number of features, there is a 4.6% penalty to using 2–bit output tokens.

The 2–bit AURA kNN algorithm also fairs poorly in comparison to LSH. In both the synthetic and real world datasets, LSH is more accurate and, for the majority of datasets, faster than 2–bit AURA kNN. However crucially, for the 1000 feature datasets, where the accuracy of 2–bit AURA kNN is only a mean of 4% worse than LSH, AURA kNN is 3.1 times faster than LSH when there are 10,000 or fewer samples in the dataset.

From these results it is possible to drawn the following conclusions. The original AURA kNN algorithm is inferior to both the LSH and Dual KD Tree algorithms in both speed and accuracy. The modifications to AURA kNN that were proposed in Chapter 5, particularly WOCC input tokens and 2–bit Baum coded output tokens, have significantly improved the speed of the algorithm, but at the cost of accuracy. The specific situation, a dataset with less than 100,000 samples and 1,000 features, where the modified AURA kNN is faster and with only a minor accuracy penalty in comparison to LSH is not commonly encountered by AURA Alert. Therefore the LSH algorithm would be a superior choice for the AURA Alert kNN algorithm where the typical datasets consist of less than 1,000 features or more than 100,000 samples.

Chapter 10

Conclusions

10.1 Introduction

In Chapter 1, two research questions were posed; firstly, how does the AURA k–Nearest Neighbour (kNN) algorithm compare with the existing state of the art kNN algorithms?; and secondly, how can the AURA kNN algorithm be modified so that it is competitive with those state of the art kNN algorithms?

The answer to these questions is important because the AURA kNN algorithm is integral to the AURA Alert anomaly detection system described in Chapter 4. Anomaly detection, along with classification, are the key methods by which data driven models of can be used to support both prognostics and diagnostics of complex systems. These topics are discussed in detail in Chapter 3.

It is important for AURA Alert to be able to handle the increasingly large volumes of data that are collected as part of monitoring large and complex systems. For this, it is essential that the kNN algorithm used to identify similar historical system states is able to scale well so that it can provide a fast and accurate search across a large number of such historical system states.

Previously, the only comparative analysis of kNN algorithms to consider AURA kNN was performed by Hodge and Austin (2005). However this analysis only considers the basic Linear Scan algorithm in comparison to AURA kNN. In Chapter 3, several algorithms were detailed with the potential to be both more accurate and faster than AURA kNN.

With regards to the question of how AURA kNN could be improved, the work of Hobson (2011) described several techniques for maximising the information density of the Correlation Matrix Memories (CMM) used to implement AURA kNN. This presented the possibility of being able to use a smaller CMM to implement a particular search and therefore enable AURA kNN to scale better with larger datasets.

10.2 Findings

Given the two research questions stated above, the findings of this research can be divided into two groups. Those belonging to research into how modify AURA kNN and those belonging to the comparative evaluation of AURA kNN.

The findings that relate to how to modify AURA kNN are discussed in detail in Chapter 5. However there are three novel contributions that are summarised below.

The first concerns optimisation of the Overlapped Binary Code Construction (OBCC) method. The work of Hobson (2011) described a method for generating OBCC tokens. But this work gives little considerable to how the method would scale to handle real world problems. In Section 5.2.4 it was shown that by using the MIN clique decomposition algorithm instead of the standard Bron–Kerbosch algorithm a substantial improvement in execution time can be achieved with only a very small increase in the size of the tokens that are generated. Encoding 400 unique values requires only 4.99 seconds using MIN in comparison to approximately 4 days when using Bron–Kerbosch. The cost of this speedup is that the weight of the resulting tokens is increased by only 12 bits.

The next contribution was the introduction of the Weighted Overlap Code Construction (WOCC) method for generating tokens. The OBCC method of Hobson (2011) generates pure binary CMM tokens. However AURA based CMMs are able to support weighted tokens during recall, but not during the training stage. After observing that multiple columns of the token matrix generated via OBCC were identical, the WOCC method was presented in Section 5.3 that allows these duplicate columns to be collapsed into a single column and associated weight. This method allows the theoretical advantages of OBCC to preserved while also generating smaller tokens.

The accuracy of AURA kNN with WOCC input tokens is robust to changes in parameters such as; max distance, max overlap and number of bins, that are used to generate the input tokens. This is due to the final filtering stage where a Linear scan of the candidate samples is used to find the k neighbours. However the CMM training and query times are very sensitive to changes in token length that are caused by increasing the above parameters. Therefore it is necessary to minimise these parameters, but also balance the reduced CMM query time with an increased Linear scan caused by more candidates being retrieved from the CMM when these parameters are small.

The final novel modification was to consider the use of Baum Coded output tokens for the CMM used in AURA kNN. Hobson (2011) noted that, up to a certain point, the information density of a CMM can be increased by increasing the weight of the CMM output tokens. This is because it allows more unique tokens to be encoded for a given length of token. Baum Codes (Baum, Moody, and Wilczek 1988) provide a fast method of generating sparse tokens of a specified weight. In Section 5.4 it was observed that by increasing the information density of the CMM, a smaller CMM can be used to store a dataset of a given size. It was shown that a dataset of 5,000 samples could be stored with 93% fewer CMM columns using 2–Bit Baum Coded output tokens rather than unary tokens. This use of non–unary output tokens had not previously been used with AURA kNN.

These changes to the method for generating input and output tokens presented several alternative configurations for a CMM to use as part of AURA kNN. In Chapters 6, 7, 8 and 9 these were examined with a view to comparing both the standard and modified AURA kNN with the spatial partitioning kNN algorithms described in Section 3.11.3 and the state of the art LSH algorithm described in Section 3.11.4. A summary of the findings is given below.

Neither OBCC or WOCC input tokens are as fast or accurate as Parabolic Kernel (Hodge and Austin 2005) (PK) input tokens. Regardless, the standard AURA kNN algo-

rithm, using PK input tokens and unary output tokens, was consistently slower than the Dual KD–Tree algorithm. This is an exact algorithm, therefore there is no advantage to be gained by using AURA kNN over Dual KD–Tree. The LSH algorithm was shown to be several orders of magnitude faster than the exact Dual KD–Tree algorithm with only a 0.32% accuracy penalty.

In order to allow AURA kNN to be competitive with LSH it was necessary to use 2–Bit Baum Coded output tokens. However a result of using these output tokens was that WOCC input tokens provided the fastest queries of a CMM. In addition, the accuracy of using PK tokens was reduced to the point that there was no significant difference in accuracy between WOCC tokens and PK tokens.

Comparing the modified AURA kNN, using WOCC input tokens and 2–Bit Baum Coded output tokens, with LSH showed that, for most datasets, LSH is still superior. The mean query time of LSH is 22% faster than the modified AURA kNN algorithm. In addition, LSH is considerably more accurate, particularly for datasets with a small number of features. LSH is a mean of 2.5 times more accurate on the 10 feature synthetic datasets. However with datasets that have 1,000 features and fewer than 10,000 samples, modified AURA kNN is faster than LSH and in the best case it is 3.1 times faster with a 4% accuracy penalty.

Overall, considering the number of feature and samples in the datasets that are typically processed by AURA Alert, LSH is clearly superior for these use cases.

10.3 Limitations

There are two caveats that must be considered with respect to the findings of this research.

The first is that the commercially sensitive nature of industrial condition monitoring data has meant that it has not been possible to acquire a large real world dataset with which to perform the experiments in Chapters 6, 7, 8 and 9. As a result majority of the conclusions have has to be drawn from synthetic datasets.

However, these results should still hold when applied to real world condition monitoring data. This is because the Gaussian random datasets were generated such that they would be similar to the expected condition monitoring datasets (A) and the findings with respect to accuracy are consistent with the real world datasets that were examined.

The second caveat is that these experiments were only performed with single threaded implementations of the algorithms. This is because the AURA library that formed the basis for much of this research currently does not currently support multi–threaded querying of the CMM.

Both AURA kNN and LSH are well suited to parallel query execution. However there is significant commercial value in these single threaded results and the best approach for implementing a parallel AURA kNN search would require significant further research.

10.4 Future Work

However the results do point to some further research that could potentially improve AURA kNN. Increasing the weight of the output tokens can further improve the speed of

the algorithm. However the accuracy penalty must be countered.

One way in which this could potentially be achieved is to use the WOCC method to encode the overlap function that defines PK tokens (Section 4.3.1). This way, the superior accuracy of the PK tokens would hopefully not be as affected by the crosstalk introduced by the increased weight of the output tokens.

Another method for improving the accuracy could be to improve the decoding of the output scores during the threshold stage of the CMM query (Section 5.4.2). Alternative methods for determining the ranking the decoded Baum Coded tokens, for example, by total weight, could potentially improve the overall accuracy of the kNN search.

The accuracy can be improved by increasing the number of candidate samples that are considered in the filtering stage of the query. This will increase the query time, but potentially by trading an increased filtering time for a reduced CMM query time, the overall time for the kNN query could be reduced.

Finally, it would be worthwhile to evaluate AURA Alert using the alternative kNN algorithms considered as part of this research. In particular an implementation that uses LSH would be worth investigating.

10.5 Final conclusion

The two questions that were posed as the basis for this research were: How does the AURA kNN algorithm compare with the existing state of the art kNN algorithms?; And how can the AURA kNN algorithm be modified so that it is competitive with those state of the art kNN algorithms?

It has been shown that the standard AURA kNN compares poorly to both the exact spatial partitioning kNN algorithms and the approximate kNN algorithm LSH. In addition, the modifications investigated as part of this research were insufficient to improve AURA kNN to the point where is competitive with LSH for the types of dataset that AURA kNN is typically used for. However the results of these modification do point towards potential further improvements, by combining WOCC input tokens with more bits set in the output tokens, that could help close the gap in performance.

Appendices

Appendix A

Normalised Timeseries data has Gaussian distribution

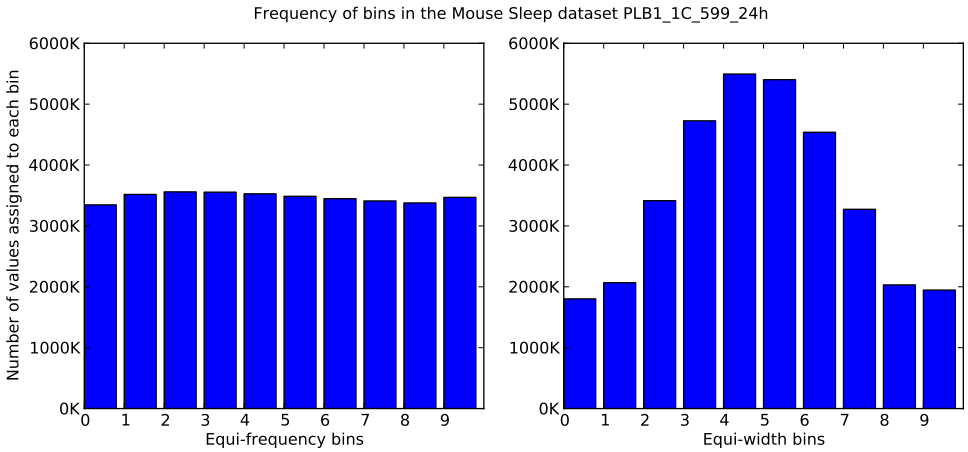
Lin et al. (2003) demonstrate that, after normalisation, the values in a time series can typically be assumed to fall approximately within a Gaussian distribution. I have verified that this assumption holds with some of the real world datasets that I have used as part of my research. In the following experiment I demonstrate that the assumption holds in one of these datasets.

The Left Hippocampus EEG channel from dataset PLB1_1C_599_24h was selected randomly as the target time series. This is mouse EEG data recorded at 200Hz for a period of approximately 24 hours giving a total of 17,347,228 samples. This channel was divided into windows of length 200 with an overlap of 100 samples between adjacent windows. In total this provides 173,471 example windows of time series data. Each window was normalised to have a mean of 0 and a standard deviation of 1.

These windows were then binned using both the Equi–width and Equi–frequency binning strategies. For both strategies, the breakpoints were calculated for 10 bins and a normalised data range between -2 and 2.

Finally the number of occurrences of each bin across all the examples was counted. The results are shown in Figure A.1. Using the Equi–frequency binning strategy produces bins that have a roughly equal number of occurrences, this is the desired outcome for this strategy. However, the implementation of Equi–frequency binning is based on the assumption that the normalised time series data has an approximately Gaussian distribution in order to determine the location of the breakpoints. Since the placement of the breakpoints has lead to a roughly equal occurrence of each bin, this assumption appears to be true. Further evidence that the normalised time series data has a Gaussian distribution can be seen via the distinctive bell curve that can be observed with the Equi–width strategy where there are more occurrences of bins that are close to the middle of the range.

Figure A.1: Verification that normalised Mouse Sleep EEG data has a Gaussian distribution.



These histograms show the number of times each bin occurs when binning the normalised Left Hippocampus EEG channel. Each of the 10 bins are labelled $0 \ldots 9$ along the x–axis while their respective counts are given on the y–axis. The Equi–frequency binning strategy assumes that the normalised data has a Gaussian distribution, this assumption is backed up by the roughly equal occurrences of each bin using this strategy. The distinctive bell curve observed with the Equi–width binning strategy provides further evidence that this assumption holds.

Appendix B

Nearest Neighbour Experiment Tables of Results

Table B.1: Mean Feature Distance for Number of Bins with PK AURA(Real)

ID	Exact	10 Bins	20 Bins	40 Bins	80 Bins
50words	0.046	0.051	0.051	0.051	0.052
Adiac	0.005	0.006	0.006	0.006	0.006
Beef	0.002	0.002	0.002	0.002	0.002
CBF	0.093	0.094	0.094	0.094	0.094
ChlorineConcentration	0.011	0.012	0.012	0.012	0.012
CinC_ECG_torso	0.032	0.032	0.033	0.033	0.033
Coffee	0.208	0.208	0.208	0.208	0.208
Cricket_X	0.044	0.048	0.048	0.048	0.048
Cricket_Y	0.044	0.048	0.048	0.047	0.047
Cricket_Z	0.044	0.049	0.048	0.048	0.048
DiatomSizeReduction	0.010	0.010	0.010	0.010	0.010
ECG200	0.053	0.054	0.054	0.054	0.054
ECGFiveDays	0.081	0.081	0.081	0.081	0.081
FaceAll	0.072	0.077	0.077	0.077	0.077
FaceFour	0.063	0.063	0.063	0.063	0.063
FacesUCR	0.079	0.085	0.085	0.085	0.085
Gun_Point	0.034	0.034	0.034	0.034	0.034
Haptics	0.012	0.012	0.012	0.012	0.012
InlineSkate	0.013	0.014	0.014	0.014	0.014
ItalyPowerDemand	0.074	0.075	0.075	0.075	0.075
Lighting2	0.039	0.039	0.039	0.039	0.039
Lighting7	0.052	0.055	0.052	0.052	0.053
MALLAT	0.009	0.009	0.009	0.009	0.009
MedicalImages	0.045	0.054	0.054	0.054	0.054
MoteStrain	0.110	0.110	0.110	0.110	0.110
NonInvasiveFatalECG_Thorax1	0.004	0.005	0.005	0.005	0.005
NonInvasiveFatalECG_Thorax2	0.004	0.004	0.004	0.004	0.005
OSULeaf	0.044	0.045	0.045	0.045	0.046
OliveOil	0.000	0.000	0.000	0.000	0.000
SonyAIBORobot_Surface	0.089	0.089	0.089	0.089	0.089
SonyAIBORobot_SurfaceII	0.131	0.132	0.132	0.132	0.132
StarLightCurves	0.007	0.008	0.007	0.007	0.007
SwedishLeaf	0.029	0.030	0.030	0.030	0.030
Symbols	0.054	0.054	0.054	0.054	0.054
Trace	0.027	0.028	0.028	0.028	0.028
TwoLeadECG	0.055	0.055	0.055	0.055	0.055
Two_Patterns	0.083	0.087	0.087	0.088	0.088
WordsSynonyms	0.051	0.056	0.057	0.057	0.057
fish	0.009	0.009	0.009	0.009	0.009
synthetic_control	0.106	0.108	0.108	0.108	0.108
test	0.379	0.379	0.379	0.379	0.379
uWaveGestureLibrary_X	0.031	0.034	0.035	0.036	0.037
uWaveGestureLibrary_Y	0.028	0.032	0.032	0.032	0.032
uWaveGestureLibrary_Z	0.030	0.032	0.034	0.034	0.034
wafer	0.029	0.032	0.032	0.032	0.034
yoga	0.019	0.019	0.020	0.028	0.029

This table shows the mean feature distance for the AURA algorithm with Parabolic Kernel input tokens, single bit output tokens and a varying numbers of bins.

Table B.2: Mean Feature Distance for Number of Bins with OBCC AURA(Real)

ID	Exact	10 Bins	20 Bins	40 Bins	80 Bins
50words	0.046	0.053	0.053	0.053	0.053
Adiac	0.005	0.006	0.006	0.006	0.006
Beef	0.002	0.002	0.002	0.002	0.002
CBF	0.093	0.094	0.094	0.094	0.094
ChlorineConcentration	0.011	0.012	0.012	0.012	0.012
CinC_ECG_torso	0.032	0.033	0.033	0.033	0.033
Coffee	0.208	0.208	0.208	0.208	0.208
Cricket_X	0.044	0.052	0.052	0.052	0.052
Cricket_Y	0.044	0.052	0.053	0.053	0.051
Cricket_Z	0.044	0.052	0.053	0.053	0.051
DiatomSizeReduction	0.010	0.010	0.010	0.010	0.010
ECG200	0.053	0.055	0.055	0.055	0.055
ECGFiveDays	0.081	0.081	0.081	0.081	0.081
FaceAll	0.072	0.079	0.078	0.078	0.077
FaceFour	0.063	0.063	0.063	0.063	0.063
FacesUCR	0.079	0.086	0.085	0.084	0.084
Gun_Point	0.034	0.035	0.035	0.034	0.034
Haptics	0.012	0.012	0.012	0.012	0.012
InlineSkate	0.013	0.014	0.014	0.014	0.014
ItalyPowerDemand	0.074	0.076	0.076	0.077	0.077
Lighting2	0.039	0.039	0.039	0.039	0.039
Lighting7	0.052	0.055	0.052	0.052	0.053
MALLAT	0.009	0.009	0.009	0.009	0.009
MedicalImages	0.045	0.061	0.062	0.063	0.062
MoteStrain	0.110	0.110	0.110	0.110	0.110
NonInvasiveFatalECG_Thorax1	0.004	0.005	0.005	0.005	0.005
NonInvasiveFatalECG_Thorax2	0.004	0.005	0.005	0.005	0.006
OSULeaf	0.044	0.046	0.046	0.046	0.046
OliveOil	0.000	0.000	0.000	0.000	0.000
SonyAIBORobot_Surface	0.089	0.089	0.089	0.089	0.089
SonyAIBORobot_SurfaceII	0.131	0.132	0.132	0.132	0.132
StarLightCurves	0.007	0.008	0.008	0.008	0.008
SwedishLeaf	0.029	0.031	0.031	0.031	0.031
Symbols	0.054	0.054	0.054	0.054	0.054
Trace	0.027	0.028	0.028	0.028	0.028
TwoLeadECG	0.055	0.055	0.055	0.055	0.055
Two_Patterns	0.083	0.088	0.088	0.087	0.087
WordsSynonyms	0.051	0.058	0.058	0.058	0.057
fish	0.009	0.009	0.009	0.009	0.009
synthetic_control	0.106	0.110	0.110	0.110	0.110
test	0.379	0.360	0.341	0.335	0.334
uWaveGestureLibrary_X	0.031	0.035	0.036	0.036	0.036
uWaveGestureLibrary_Y	0.028	0.033	0.033	0.032	0.032
uWaveGestureLibrary_Z	0.030	0.033	0.035	0.033	0.034
wafer	0.029	0.034	0.034	0.036	0.034
yoga	0.019	0.020	0.021	0.027	0.028

This table shows the mean feature distance for the AURA algorithm with OBCC input tokens, single bit output tokens and a varying numbers of bins.

Table B.3: Mean Feature Distance for Number of Bins with WOCC AURA(Real)

ID	Exact	10 Bins	20 Bins	40 Bins	80 Bins
50words	0.046	0.053	0.053	0.053	0.053
Adiac	0.005	0.006	0.006	0.006	0.006
Beef	0.002	0.002	0.002	0.002	0.002
CBF	0.093	0.094	0.094	0.094	0.094
ChlorineConcentration	0.011	0.012	0.012	0.012	0.012
CinC_ECG_torso	0.032	0.033	0.033	0.033	0.033
Coffee	0.208	0.208	0.208	0.208	0.208
Cricket_X	0.044	0.052	0.052	0.052	0.052
Cricket_Y	0.044	0.052	0.053	0.053	0.051
Cricket_Z	0.044	0.052	0.053	0.053	0.051
DiatomSizeReduction	0.010	0.010	0.010	0.010	0.010
ECG200	0.053	0.055	0.055	0.055	0.055
ECGFiveDays	0.081	0.081	0.081	0.081	0.081
FaceAll	0.072	0.079	0.078	0.078	0.077
FaceFour	0.063	0.063	0.063	0.063	0.063
FacesUCR	0.079	0.086	0.085	0.084	0.084
Gun_Point	0.034	0.035	0.035	0.034	0.034
Haptics	0.012	0.012	0.012	0.012	0.012
InlineSkate	0.013	0.014	0.014	0.014	0.014
ItalyPowerDemand	0.074	0.076	0.076	0.077	0.077
Lighting2	0.039	0.039	0.039	0.039	0.039
Lighting7	0.052	0.055	0.052	0.052	0.053
MALLAT	0.009	0.009	0.009	0.009	0.009
MedicalImages	0.045	0.061	0.062	0.063	0.062
MoteStrain	0.110	0.110	0.110	0.110	0.110
NonInvasiveFatalECG_Thorax1	0.004	0.005	0.005	0.005	0.005
NonInvasiveFatalECG_Thorax2	0.004	0.005	0.005	0.005	0.006
OSULeaf	0.044	0.046	0.046	0.046	0.046
OliveOil	0.000	0.000	0.000	0.000	0.000
SonyAIBORobot_Surface	0.089	0.089	0.089	0.089	0.089
SonyAIBORobot_SurfaceII	0.131	0.132	0.132	0.132	0.132
StarLightCurves	0.007	0.008	0.008	0.008	0.008
SwedishLeaf	0.029	0.031	0.031	0.031	0.031
Symbols	0.054	0.054	0.054	0.054	0.054
Trace	0.027	0.028	0.028	0.028	0.028
TwoLeadECG	0.055	0.055	0.055	0.055	0.055
Two_Patterns	0.083	0.088	0.088	0.087	0.087
WordsSynonyms	0.051	0.058	0.058	0.058	0.057
fish	0.009	0.009	0.009	0.009	0.009
synthetic_control	0.106	0.110	0.110	0.110	0.110
test	0.379	0.360	0.341	0.335	0.334
uWaveGestureLibrary_X	0.031	0.035	0.036	0.036	0.036
uWaveGestureLibrary_Y	0.028	0.033	0.033	0.032	0.032
uWaveGestureLibrary_Z	0.030	0.033	0.035	0.033	0.034
wafer	0.029	0.034	0.034	0.036	0.034
yoga	0.019	0.020	0.021	0.027	0.028

This table shows the mean feature distance for the AURA algorithm with WOCC input tokens, single bit output tokens and a varying numbers of bins.

Table B.4: Mean Feature Distance for Number of Bins with 2–Bit PK AURA(Real)

ID	Exact	10 Bins	20 Bins	40 Bins	80 Bins
50words	0.046	0.064	0.065	0.064	0.064
Adiac	0.005	0.009	0.010	0.008	0.008
Beef	0.002	0.002	0.002	0.002	0.002
CBF	0.093	0.097	0.097	0.096	0.096
ChlorineConcentration	0.011	0.026	0.027	0.025	0.025
CinC_ECG_torso	0.032	0.033	0.033	0.033	0.034
Coffee	0.208	0.208	0.208	0.208	0.208
Cricket_X	0.044	0.061	0.059	0.059	0.059
Cricket_Y	0.044	0.059	0.059	0.057	0.057
Cricket_Z	0.044	0.060	0.058	0.059	0.058
DiatomSizeReduction	0.010	0.010	0.010	0.010	0.010
ECG200	0.053	0.070	0.068	0.068	0.067
ECGFiveDays	0.081	0.081	0.081	0.081	0.081
FaceAll	0.072	0.093	0.093	0.093	0.093
FaceFour	0.063	0.063	0.063	0.063	0.063
FacesUCR	0.079	0.097	0.097	0.097	0.097
Gun_Point	0.034	0.043	0.037	0.037	0.034
Haptics	0.012	0.013	0.014	0.015	0.015
InlineSkate	0.013	0.018	0.017	0.017	0.017
ItalyPowerDemand	0.074	0.105	0.096	0.098	0.099
Lighting2	0.039	0.039	0.039	0.039	0.039
Lighting7	0.052	0.057	0.052	0.052	0.053
MALLAT	0.009	0.010	0.009	0.009	0.010
MedicalImages	0.045	0.073	0.073	0.070	0.071
MoteStrain	0.110	0.110	0.110	0.110	0.110
NonInvasiveFatalECG_Thorax1	0.004	0.009	0.008	0.009	0.008
NonInvasiveFatalECG_Thorax2	0.004	0.009	0.008	0.009	0.009
OSULeaf	0.044	0.053	0.053	0.052	0.053
OliveOil	0.000	0.000	0.000	0.000	0.000
SonyAIBORobot_Surface	0.089	0.089	0.089	0.089	0.089
SonyAIBORobot_SurfaceII	0.131	0.132	0.132	0.133	0.132
StarLightCurves	0.007	0.015	0.014	0.014	0.013
SwedishLeaf	0.029	0.044	0.043	0.043	0.042
Symbols	0.054	0.054	0.054	0.054	0.054
Trace	0.027	0.031	0.032	0.031	0.032
TwoLeadECG	0.055	0.055	0.055	0.055	0.055
Two_Patterns	0.083	0.104	0.104	0.105	0.104
WordsSynonyms	0.051	0.067	0.066	0.067	0.067
fish	0.009	0.011	0.009	0.009	0.009
synthetic_control	0.106	0.139	0.139	0.139	0.139
test	0.379	0.379	0.379	0.379	0.379
uWaveGestureLibrary_X	0.031	0.051	0.048	0.045	0.044
uWaveGestureLibrary_Y	0.028	0.044	0.044	0.038	0.039
uWaveGestureLibrary_Z	0.030	0.048	0.046	0.039	0.040
wafer	0.029	0.062	0.055	0.048	0.062
yoga	0.019	0.035	0.027	0.032	0.039

This table shows the mean feature distance for the AURA algorithm with Parabolic Kernel input tokens, single bit output tokens and a varying numbers of bins.

Table B.5: Mean Feature Distance for Number of Bins with 2–Bit OBCC AURA(Real)

ID	Exact	10 Bins	20 Bins	40 Bins	80 Bins
50words	0.046	0.065	0.065	0.065	0.065
Adiac	0.005	0.008	0.008	0.008	0.007
Beef	0.002	0.002	0.002	0.002	0.002
CBF	0.093	0.097	0.097	0.096	0.096
ChlorineConcentration	0.011	0.025	0.025	0.024	0.023
CinC_ECG_torso	0.032	0.033	0.033	0.033	0.034
Coffee	0.208	0.208	0.208	0.208	0.208
Cricket_X	0.044	0.062	0.062	0.060	0.061
Cricket_Y	0.044	0.061	0.061	0.060	0.059
Cricket_Z	0.044	0.062	0.061	0.061	0.060
DiatomSizeReduction	0.010	0.010	0.010	0.010	0.010
ECG200	0.053	0.068	0.068	0.068	0.068
ECGFiveDays	0.081	0.081	0.081	0.081	0.081
FaceAll	0.072	0.094	0.094	0.093	0.093
FaceFour	0.063	0.063	0.063	0.063	0.063
FacesUCR	0.079	0.097	0.097	0.096	0.096
Gun_Point	0.034	0.042	0.037	0.036	0.034
Haptics	0.012	0.013	0.015	0.015	0.015
InlineSkate	0.013	0.017	0.018	0.018	0.018
ItalyPowerDemand	0.074	0.099	0.099	0.099	0.099
Lighting2	0.039	0.039	0.039	0.039	0.039
Lighting7	0.052	0.057	0.052	0.052	0.053
MALLAT	0.009	0.010	0.009	0.009	0.009
MedicalImages	0.045	0.075	0.075	0.075	0.075
MoteStrain	0.110	0.110	0.110	0.110	0.110
NonInvasiveFatalECG_Thorax1	0.004	0.008	0.007	0.008	0.008
NonInvasiveFatalECG_Thorax2	0.004	0.009	0.008	0.009	0.008
OSULeaf	0.044	0.053	0.053	0.052	0.052
OliveOil	0.000	0.000	0.000	0.000	0.000
SonyAIBORobot_Surface	0.089	0.089	0.089	0.089	0.089
SonyAIBORobot_SurfaceII	0.131	0.133	0.133	0.133	0.133
StarLightCurves	0.007	0.013	0.014	0.014	0.014
SwedishLeaf	0.029	0.043	0.042	0.042	0.042
Symbols	0.054	0.054	0.054	0.054	0.054
Trace	0.027	0.032	0.031	0.032	0.031
TwoLeadECG	0.055	0.055	0.055	0.055	0.055
Two_Patterns	0.083	0.105	0.105	0.104	0.104
WordsSynonyms	0.051	0.067	0.067	0.067	0.067
fish	0.009	0.011	0.009	0.009	0.009
synthetic_control	0.106	0.139	0.140	0.140	0.140
test	0.379	0.370	0.360	0.357	0.357
uWaveGestureLibrary_X	0.031	0.051	0.046	0.044	0.043
uWaveGestureLibrary_Y	0.028	0.047	0.043	0.038	0.038
uWaveGestureLibrary_Z	0.030	0.050	0.043	0.039	0.040
wafer	0.029	0.060	0.054	0.049	0.054
yoga	0.019	0.034	0.028	0.032	0.038

This table shows the mean feature distance for the AURA algorithm with OBCC input tokens, single bit output tokens and a varying numbers of bins.

Table B.6: Mean Feature Distance for Number of Bins with 2–Bit WOCC AURA(Real)

ID	Exact	10 Bins	20 Bins	40 Bins	80 Bins
50words	0.046	0.065	0.065	0.065	0.065
Adiac	0.005	0.008	0.008	0.008	0.007
Beef	0.002	0.002	0.002	0.002	0.002
CBF	0.093	0.097	0.097	0.096	0.096
ChlorineConcentration	0.011	0.025	0.025	0.024	0.023
CinC_ECG_torso	0.032	0.033	0.033	0.033	0.034
Coffee	0.208	0.208	0.208	0.208	0.208
Cricket_X	0.044	0.062	0.062	0.060	0.061
Cricket_Y	0.044	0.061	0.061	0.060	0.059
Cricket_Z	0.044	0.062	0.061	0.061	0.060
DiatomSizeReduction	0.010	0.010	0.010	0.010	0.010
ECG200	0.053	0.068	0.068	0.068	0.068
ECGFiveDays	0.081	0.081	0.081	0.081	0.081
FaceAll	0.072	0.094	0.094	0.093	0.093
FaceFour	0.063	0.063	0.063	0.063	0.063
FacesUCR	0.079	0.097	0.097	0.096	0.096
Gun_Point	0.034	0.042	0.037	0.036	0.034
Haptics	0.012	0.013	0.015	0.015	0.015
InlineSkate	0.013	0.017	0.018	0.018	0.018
ItalyPowerDemand	0.074	0.099	0.099	0.099	0.099
Lighting2	0.039	0.039	0.039	0.039	0.039
Lighting7	0.052	0.057	0.052	0.052	0.053
MALLAT	0.009	0.010	0.009	0.009	0.009
MedicalImages	0.045	0.075	0.075	0.075	0.075
MoteStrain	0.110	0.110	0.110	0.110	0.110
NonInvasiveFatalECG_Thorax1	0.004	0.008	0.007	0.008	0.008
NonInvasiveFatalECG_Thorax2	0.004	0.009	0.008	0.009	0.008
OSULeaf	0.044	0.053	0.053	0.052	0.052
OliveOil	0.000	0.000	0.000	0.000	0.000
SonyAIBORobot_Surface	0.089	0.089	0.089	0.089	0.089
SonyAIBORobot_SurfaceII	0.131	0.133	0.133	0.133	0.133
StarLightCurves	0.007	0.013	0.014	0.014	0.014
SwedishLeaf	0.029	0.043	0.042	0.042	0.042
Symbols	0.054	0.054	0.054	0.054	0.054
Trace	0.027	0.032	0.031	0.032	0.031
TwoLeadECG	0.055	0.055	0.055	0.055	0.055
Two_Patterns	0.083	0.105	0.105	0.104	0.104
WordsSynonyms	0.051	0.067	0.067	0.067	0.067
fish	0.009	0.011	0.009	0.009	0.009
synthetic_control	0.106	0.139	0.140	0.140	0.140
test	0.379	0.370	0.360	0.357	0.357
uWaveGestureLibrary_X	0.031	0.051	0.046	0.044	0.043
uWaveGestureLibrary_Y	0.028	0.047	0.043	0.038	0.038
uWaveGestureLibrary_Z	0.030	0.050	0.043	0.039	0.040
wafer	0.029	0.060	0.054	0.049	0.054
yoga	0.019	0.034	0.028	0.032	0.038

This table shows the mean feature distance for the AURA algorithm with WOCC input tokens, single bit output tokens and a varying numbers of bins.

Glossary

CMM Correlation Matrix Memory – A binary associative neural network.

Token The inputs or outputs to a CMM. A single input token is associated with a single output token for each training iteration.

Token Length The total number of values, both zero and non–zero, in a token.

Token Weight The sum of the non–zero values in a token.

AURA A library for training and querying CMMs.

kNN k–Nearest Neighbour – An algorithm that selects the k most similar samples to a target sample.

PK Parabolic Kernel – A method for generating weighted CMM input tokens.

OBCC Overlapped Binary Code Construction – A method for generating binary CMM input tokens.

WOCC Weighted Overlapped Code Construction – A method for generating weighted CMM input tokens.

LSH Locality Sensitive Hashing – An approximate kNN algorithm.

Training Associating pairs of input and output tokens within a CMM.

Query Retrieving the output token from a CMM that was associated with a given input token.

Bibliography

[1] R. Agrawal, C. Faloutsos, and A. Swami. "Efficient similarity search in sequence databases". In: *Foundations of Data Organization and Algorithms* (1993), pp. 69–84.

[2] R. Agrawal, K. Lin, H. Sawhney, and K. Shim. "Fast similarity search in the presence of noise, scaling, and translation in time-series databases". In: (1995).

[3] L. Atlas, G. Bloor, T. Brotherton, L. Howard, L. Jaw, G. Kacprzynski, G. Karsai, R. Mackey, J. Mesick, R. Reuter, et al. "An evolvable tri-reasoner ivhm system". In: *Aerospace Conference, 2001, IEEE Proceedings.* Vol. 6. IEEE. 2001, pp. 3023–3037.

[4] J. Austin. "Distributed associative memories for high-speed symbolic reasoning". In: *Fuzzy Sets and Systems* 82.2 (1996), pp. 223–233.

[5] J. Austin. "Grey–scale n–tuple processing". In: *Lecture Notes in Computer Science* 301 (1988), pp. 110–119. ISSN: 0302-9743.

[6] J. Austin. "Rapid learning with a hybrid neural network". In: *Neural Network World* 5.93 (1993), pp. 531–550.

[7] J. Austin, G. Brewer, T. Jackson, and V. Hodge. "AURA-Alert: The use of binary associative memories for condition monitoring applications." In: *Proceedings of 7th International Conference on Condition Monitoring and Machinery Failure Prevention Technologies.* 2010.

[8] J. Austin. "ADAM: A distributed associative memory for scene analysis". In: *Proceedings of First International Conference on Neural Networks.* Vol. 4. 1987, p. 285.

[9] J. Austin, J. V. Kennedy, and K. Lees. "The Advanced Uncertain Reasoning Architecture". In: *Proceedings of the Weightless Neural Network Workshop* (1995).

[10] J. Austin and T. J. Stonham. "Distributed associative memory for use in scene analysis". In: *Image and Vision Computing* 5.4 (1987), pp. 251–260.

[11] J. Austin, C. Bailey, A. Moulds, G. Hollier, M. Freeman, G. Riedel, A. Fargus, T. Lampert, and B. Platt. "A Miniaturized 4-Channel, 2KSa/sec Biosignal Data Recorder With 3-Axis Accelerometer and Infra-red Timestamp Function". In: *SENSORCOMM 2013, The Seventh International Conference on Sensor Technologies and Applications.* 2013, pp. 213–219.

[12] O. Balci. "Verification validation and accreditation of simulation models". In: *Proceedings of the 29th conference on Winter simulation.* IEEE Computer Society. 1997, pp. 135–141.

[13] E. B. Baum, J. Moody, and F. Wilczek. "Internal representations for associative memory". In: *Biological Cybernetics* 59.4-5 (1988), pp. 217–228.

[14] M. Bawa, T. Condie, and P. Ganesan. "LSH forest: self-tuning indexes for similarity search". In: *Proceedings of the 14th international conference on World Wide Web*. ACM. 2005, pp. 651–660.

[15] R. Bellman. *Adaptive control processes: a guided tour*. Rand Corporation Research studies. Princeton University Press, 1961.

[16] J. L. Bentley. "Multidimensional binary search trees used for associative searching". In: *Communications of the ACM* 18.9 (1975), pp. 509–517.

[17] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and software verification: model-checking techniques and tools*. Springer Publishing Company, Incorporated, 2010.

[18] D. Berndt and J. Clifford. "Using dynamic time warping to find patterns in time series". In: *AAAI94 workshop on knowledge discovery in databases*. 1994, pp. 359–370.

[19] A. Beygelzimer, S. Kakade, and J. Langford. "Cover trees for nearest neighbor". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 97–104.

[20] C. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[21] W. Bledsoe and I. Browning. *Pattern recognition and reading by machine*. PGEC, 1959, pp. 225–232.

[22] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. "The maximum clique problem". In: *Handbook of combinatorial optimization*. Springer, 1999, pp. 1–74.

[23] C. Bron and J. Kerbosch. "Algorithm 457: finding all cliques of an undirected graph". In: *Communications of the ACM* 16.9 (1973), pp. 575–577.

[24] C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". In: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167.

[25] Y. Caballero, R. Bello, A. Taboada, A. Nowe, M. M. García, and G. Casas. "A new measure based in the rough set theory to estimate the training set quality". In: *Symbolic and Numeric Algorithms for Scientific Computing, 2006. SYNASC'06. Eighth International Symposium on*. IEEE. 2006, pp. 133–140.

[26] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. "iSAX 2.0: Indexing and mining one billion time series". In: *ICDM, September* (2010).

[27] D. Casasent and B. Telfer. "High capacity pattern recognition associative processors". In: *Neural Networks* 5.4 (1992), pp. 687–698.

[28] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. "Locally adaptive dimensionality reduction for indexing large time series databases". In: *ACM Transactions on Database Systems (TODS)* 27.2 (2002), pp. 188–228.

[29] K. Chan and A. Fu. "Efficient time series matching by wavelets". In: *Data Engineering, 1999. Proceedings., 15th International Conference on*. 1999, pp. 126 –133.

[30] R. Chinnam and P. Baruah. "A neuro-fuzzy approach for estimating mean residual life in condition-based maintenance systems". In: *International Journal of Materials and Product Technology* 20.1 (2004), pp. 166–179.

[31] R. Collobert and S. Bengio. "Links between perceptrons, MLPs and SVMs". In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 23.

[32] R. Commission. *Investigation of the Challenger Accident*. Tech. rep. U.S. House Committee on Science and Technology., 1986.

[33] V. Cutello, G. Nicosia, M. Pavone, and J. Timmis. "An immune algorithm for protein structure prediction on lattice models". In: *Evolutionary Computation, IEEE Transactions on* 11.1 (2007), pp. 101–117.

[34] D. Dasgupta and S. Forrest. "Novelty detection in time series data using ideas from immunology". In: *Proceedings of the International Conference on Intelligent Systems*. Citeseer. 1996, pp. 82–87.

[35] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the twentieth annual symposium on Computational geometry*. ACM. 2004, pp. 253–262.

[36] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1977), pp. 1–38.

[37] P. D'haeseleer, S. Forrest, and P. Helman. "An immunological approach to change detection: Algorithms, analysis and implications". In: *sp* (1996), p. 0110.

[38] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. "Querying and mining of time series data: experimental comparison of representations and distance measures". In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1542–1552.

[39] J. Dougherty, R. Kohavi, and M. Sahami. "Supervised and unsupervised discretization of continuous features". In: *ICML*. 1995, pp. 194–202.

[40] J. Edgerton and G. Kochis. *Hard drive protection system and method*. US Patent 5,835,298. 1998.

[41] T. Edison. *FUSE-BLOCK*. US Patent 438,305. 1890.

[42] V. Estivill-Castro and D. Wood. "A survey of adaptive sorting algorithms". In: *ACM Computing Surveys (CSUR)* 24.4 (1992), pp. 441–476.

[43] European Centre for Medium Range Weather Forcasting. *European Centre for Medium Range Weather Forcasting, Annual Report 2013*. 2013.

[44] A. Fargus. *Classifying Mouse Sleep States*. 2012. URL: `https://vimeo.com/56685898`.

[45] C. Ferri, P. Flach, and J. Hernandez-Orallo. "Learning Decision Trees Using the Area Under the ROC Curve". In: *Proceedings of the Nineteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc. 2002, pp. 139–146.

[46] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri. "Self-nonself discrimination in a computer". In: *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*. IEEE. 1994, pp. 202–212.

[47] J. H. Friedman, J. L. Bentley, and R. A. Finkel. "An algorithm for finding best matches in logarithmic expected time". In: *ACM Transactions on Mathematical Software (TOMS)* 3.3 (1977), pp. 209–226.

[48] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990. ISBN: 0122698517.

[49] A. Gardner, A. Krieger, G. Vachtsevanos, and B. Litt. "One-class novelty detection for seizure analysis from intracranial EEG". In: *The Journal of Machine Learning Research* 7 (2006), pp. 1025–1044.

[50] P. Geurts. "Pattern Extraction for Time Series Classification". In: *Principles of Data Mining and Knowledge Discovery* (2001), pp. 115–127.

[51] F. Glover. "Tabu search-part I". In: *ORSA Journal on computing* 1.3 (1989), pp. 190–206.

[52] D. Goldin and P. Kanellakis. "On similarity queries for time-series data: Constraint specification and implementation". In: *Principles and Practice of Constraint ProgrammingâĂŤCP'95*. Springer. 1995, pp. 137–153.

[53] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*. Vol. 57. Elsevier, 2004.

[54] A. Graps. "An introduction to wavelets". In: *Computational Science & Engineering, IEEE* 2.2 (1995), pp. 50–61.

[55] A. G. Gray and A. W. Moore. "N-Body'problems in statistical learning". In: *NIPS*. Vol. 4. Citeseer. 2000, pp. 521–527.

[56] M. Guo, L. Xie, S. Wang, and J. Zhang. "Research on an integrated ICA-SVM based framework for fault diagnosis". In: *Systems, Man and Cybernetics, 2003. IEEE International Conference on*. Vol. 3. 2003, 2710 –2715 vol.3.

[57] T. Hancock, T. Jiang, M. Li, and J. Tromp. "Lower Bounds on Learning Decision Lists and Trees* 1". In: *Information and Computation* 126.2 (1996), pp. 114–122.

[58] J. Harant, Z. Ryjacek, and I. Schiermeyer. "Forbidden subgraphs implying the MIN-algorithm gives a maximum independent set". In: *Discrete mathematics* 256.1 (2002), pp. 193–201.

[59] T. Hastie, R. Tibshirani, J. Friedman, T Hastie, J Friedman, and R Tibshirani. *The elements of statistical learning*. Vol. 2. 1. Springer, 2009.

[60] S. Hobson. *Correlation Matrix Memories: Improving Performance for Capacity and Generalisation*. 2011.

[61] V. J. Hodge and J. Austin. *Discretisation of Data in a Binary Neural k-Nearest Neighbour Algorithm*. Technical Report YCS-2012-473. 2012.

[62] V. Hodge and J. Austin. "A binary neural k-nearest neighbour technique". In: *Knowledge and Information Systems* 8.3 (2005), pp. 276–291.

[63] R. Hopkins and K. Jenkins. *Eating the IT elephant: Moving from greenfield development to brownfield*. IBM Press, 2008.

[64] K. Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2 (1991), pp. 251–257.

[65] I. Howard, S. Jia, and J. Wang. "The Dynamic Modelling Of a Spur Gear In Mesh Including Friction and a Crack". In: *Mechanical Systems and Signal Processing* 15.5 (2001), pp. 831 –853. ISSN: 0888-3270.

[66] P. Indyk and R. Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM. 1998, pp. 604–613.

[67] A. Jardine, D. Lin, and D. Banjevic. "A review on machinery diagnostics and prognostics implementing condition-based maintenance". In: *Mechanical Systems and Signal Processing* 20.7 (2006), pp. 1483 –1510. ISSN: 0888-3270.

[68] L. Jiang, Z. Cai, D. Wang, and S. Jiang. "Survey of improving k-nearest-neighbor for classification". In: *Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on*. Vol. 1. IEEE. 2007, pp. 679–683.

[69] I. Jolliffe. "Principal component analysis". In: *Encyclopedia of Statistics in Behavioral Science* (2002).

[70] R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[71] E. Keogh and S. Kasetty. "On the need for time series data mining benchmarks: a survey and empirical demonstration". In: *Data Mining and Knowledge Discovery* 7.4 (2003), pp. 349–371.

[72] E. Keogh, S. Chu, D. Hart, and M. Pazzani. "An online algorithm for segmenting time series". In: *icdm*. Published by the IEEE Computer Society. 2001, p. 289.

[73] E Keogh, X Xi, L. Wei, and C. Ratanamahatana. "UCR time series classification/-clustering page". In: *Training and testing data sets: Available online: http://www. cs. ucr. edu/eamonn/time series data/(accessed on 15 January 2014)* (2011).

[74] E. Keogh and M. Pazzani. "Scaling up dynamic time warping for datamining applications". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '00. Boston, Massachusetts, United States: ACM, 2000, pp. 285–289. ISBN: 1-58113-233-6.

[75] R. Krishnan, V. Hodge, J. Austin, and J. Polak. "A computationally efficient method for online identification of traffic control intervention measures". In: *42nd Annual UTSG Conference, University of Plymouth, UK: Jan*. 2010, pp. 5–7.

[76] B. Lee. "Application of the discrete wavelet transform to the monitoring of tool failure in end milling using the spindle motor current". In: *The International Journal of Advanced Manufacturing Technology* 15.4 (1999), pp. 238–243.

[77] D. Li, K. Wong, Y. Hu, and A. Sayeed. "Detection, classification and tracking of targets in distributed sensor networks". In: *IEEE signal processing magazine* 19.2 (2002), pp. 17–29.

[78] B. Liang and J. Austin. "A neural network for mining large volumes of time series data". In: *Industrial Technology, 2005. ICIT 2005. IEEE International Conference on*. IEEE. 2005, pp. 688–693.

[79] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms". In: *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. ACM. 2003, p. 11.

[80] R. Little. "Regression With Missing X's: A Review". English. In: *Journal of the American Statistical Association* 87.420 (1992), pp. 1227–1237. ISSN: 01621459.

[81] J. Ma and S. Perkins. "Time-series novelty detection using one-class support vector machines". In: *Neural Networks, 2003. Proceedings of the International Joint Conference on*. Vol. 3. Ieee. 2003, pp. 1741–1745.

[82] O. Maimon and L. Rokach. "Data Mining and Knowledge Discovery Handbook". In: (2010).

[83] J. I. Maletic and A. Marcus. "Data cleansing: A prelude to knowledge discovery". In: *Data Mining and Knowledge Discovery Handbook*. Springer, 2010, pp. 19–32.

[84] W. B. March. "Multi-tree algorithms for computational statistics and phyiscs". In: (2013).

[85] M. Markou and S. Singh. "Novelty detection: a review–part 1: statistical approaches". In: *Signal Processing* 83.12 (2003), pp. 2481 –2497. ISSN: 0165-1684.

[86] M. Markou and S. Singh. "Novelty detection: a review–part 2:: neural network based approaches". In: *Signal Processing* 83.12 (2003), pp. 2499 –2521. ISSN: 0165-1684.

[87] P. Marteau. "Time warp edit distance with stiffness adjustment for time series matching". In: *IEEE transactions on pattern analysis and machine intelligence* (2008), pp. 306–318.

[88] A. R. Masegosa and S. Moral. "An interactive approach for cleaning noisy observations in Bayesian networks with the help of an expert". In: *6th European Workshop on Probabilistic Graphical Models (PGM 2012)*. 2012, pp. 243–250.

[89] M. Mezzanzanica, R. Boselli, M. Cesarini, and F. Mercorio. "Automatic synthesis of data cleansing activities". In: *The 2nd International Conference on Data Management Technologies and Applications (DATA)*. Vol. 2. Markus Helfert and Chiara Francalanci. 2013, pp. 138–149.

[90] H. Min. "A Global Discretization and Attribute Reduction Algorithm Based on K-Means Clustering and Rough Sets Theory". In: *Proceedings of the 2009 Second International Symposium on Knowledge Acquisition and Modeling - Volume 02*. KAM '09. IEEE Computer Society, 2009, pp. 92–95. ISBN: 978-0-7695-3888-4.

[91] J. W. Moon and L. Moser. "On cliques in graphs". In: *Israel journal of Mathematics* 3.1 (1965), pp. 23–28.

[92] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[93] C. O'Halloran and C. Pygott. "Formalising C and C++ for Use in High Integrity Systems". In: *The Safety of Systems* (2007), pp. 243–260.

[94] S. M. Omohundro. "Five balltree construction algorithms". In: (1989).

[95] Oxford Dictionaries. ""system"". In: (2010). URL: `http://oxforddictionaries.com/definition/system`.

[96] G. Palm, F. Schwenker, and F. T. Sommer. "Associative memory networks and sparse similarity preserving codes". In: *From Statistics to Neural Networks*. Springer, 1994, pp. 282–302.

[97] S. Park, D. Lee, and W. Chu. "Fast retrieval of similar subsequences in long sequence databases". In: *Knowledge and Data Engineering Exchange, 1999.(KDEX'99) Proceedings. 1999 Workshop on*. IEEE. 1999, pp. 60–67.

[98] A. Patcha and J. Park. "An overview of anomaly detection techniques: Existing solutions and latest technological trends". In: *Computer Networks* 51.12 (2007), pp. 3448–3470.

[99] T. Pavlidis and S. Horowitz. "Segmentation of plane curves". In: *Computers, IEEE Transactions on* 100.8 (1974), pp. 860–870.

[100] M. Pecht. *Prognostics and Health Management of Electronics*. New York: Wiley, 2008. ISBN: 9780470278024.

[101] A. Penzias. "The origin of the elements". In: *Reviews of Modern Physics* 51.3 (1979), pp. 425–431.

[102] V. Pope, S. Brown, R. Clark, M. Collins, W. Collins, C. Dearden, J. Gunson, G. Harris, C. Jones, A. Keen, et al. "The Met Office Hadley Centre climate modelling capability: the competing requirements for improved resolution, complexity and dealing with uncertainty". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365.1860 (2007), p. 2635.

[103] S. Poyhonen, P. Jover, and H. Hyotyniemi. "Signal processing of vibrations for condition monitoring of an induction motor". In: *Control, Communications and Signal Processing, 2004. First International Symposium on*. 2004, pp. 499 –502.

[104] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. "Searching and mining trillions of time series subsequences under dynamic time warping". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 262–270.

[105] P. Ram, D. Lee, W. March, and A. G. Gray. "Linear-time algorithms for pairwise statistical problems". In: *Advances in Neural Information Processing Systems*. 2009, pp. 1527–1535.

[106] S. J. Raudys and A. K. Jain. "Small sample size effects in statistical pattern recognition: Recommendations for practitioners". In: *IEEE Transactions on pattern analysis and machine intelligence* 13.3 (1991), pp. 252–264.

[107] C. Robert, C. Guilpin, and A. Limoge. "Automated sleep staging systems in rats". In: *Journal of neuroscience methods* 88.2 (1999), pp. 111–122.

[108] J. Ryan, M. Lin, and R. Miikkulainen. "Intrusion detection with neural networks". In: *Advances in neural information processing systems*. Morgan Kaufmann Publishers. 1998, pp. 943–949.

[109] W. Van der Schalie, H. Gardner Jr, J. Bantle, C. De Rosa, R. Finch, J. Reif, R. Reuter, L. Backer, J. Burger, L. Folmar, et al. "Animals as sentinels of human health hazards of environmental chemicals." In: *Environmental health perspectives* 107.4 (1999), p. 309.

[110] G. Schmidberger and E. Frank. "Unsupervised discretization using tree-based density estimation". In: *Knowledge Discovery in Databases: PKDD 2005*. Springer, 2005, pp. 240–251.

[111] M. Schwabacher. "A survey of data-driven prognostics". In: *Proceedings of the AIAA Infotech@ Aerospace Conference*. 2005.

[112] G. Shakhnarovich, P. Indyk, and T. Darrell. *Nearest-neighbor methods in learning and vision: theory and practice*. 2006.

[113] B. Silverman. "Density Estimation for Statistics and Data Analysis". In: *Monographs on Statistics and Applied Probability* (1986).

[114] P. St-Louis, B Gendron, and J Ferland. "A penalty-evaporation heuristic in a decomposition method for the maximum clique problem". In: *Optimization Days* (2004).

[115] R. Stephenson, A. Caron, D. Cassel, and J. Kostela. "Automated analysis of sleep-wake state in rats". In: *Journal of Neuroscience Methods* 184.2 (2009), pp. 263 –274. ISSN: 0165-0270.

[116] J. V. Stone. *Bayes' Rule: A Tutorial Introduction to Bayesian Analysis*. JV Stone, 2013.

[117] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey. "Streaming similarity search over one billion tweets using parallel locality-sensitive hashing". In: *Proceedings of the VLDB Endowment* 6.14 (2013), pp. 1930–1941.

[118] D. Tax and R. Duin. "Data domain description using support vectors". In: *Proceedings of the European Symposium on Artificial Neural Networks*. Vol. 256. Citeseer. 1999.

[119] E. Tomita, A. Tanaka, and H. Takahashi. "The worst-case time complexity for generating all maximal cliques and computational experiments". In: *Theoretical Computer Science* 363.1 (2006), pp. 28–42.

[120] J. Travis. "Chernobyl explosion. Inside look confirms more radiation". In: *Science* 263.5148 (1994), p. 750.

[121] V. Tuzlukov. *Signal processing noise*. Vol. 8. CRC Press, 2002.

[122] G. Vasconcelos, M. Fairhurst, and D. Bisset. "A bootstrap-like rejection mechanism for multilayer perceptron networks". In: *II Simposio Brasileiro de Redes Neurais, São Carlos-SP, Brazil*. Citeseer. 1995, pp. 167–172.

[123] N. Vichare and M. Pecht. "Prognostics and health management of electronics". In: *Components and Packaging Technologies, IEEE Transactions on* 29.1 (2006), pp. 222 –229. ISSN: 1521-3331.

[124] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. "Indexing multidimensional time-series". In: *The VLDB Journal* 15.1 (2006), pp. 1–20.

[125] R. Weber, H.-J. Schek, and S. Blott. "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces". In: *VLDB*. Vol. 98. 1998, pp. 194–205.

[126] M. Weeks, V. Hodge, S. O'Keefe, J. Austin, and K. Lees. "Improved AURA k-nearest neighbour approach". In: *Artificial Neural Nets Problem Solving Methods* (2003), pp. 1043–1043.

[127] L. Wehenkel. *Automatic Learning Techniques in Power Systems*. 429. Kluwer Academic Publishers, 1998.

[128] K. Weinberger and L. Saul. "Distance metric learning for large margin nearest neighbor classification". In: *The Journal of Machine Learning Research* 10 (2009), pp. 207–244.

[129] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. "Non-holographic associative memory." In: *Nature* (1969).

[130] R. A. Wilson, F. Keil, and M. Fahle. "The MIT Encyclopedia of the Cognitive Sciences". In: *Nature* 402.6761 (1999), p. 460.

[131] B. Yi and C. Faloutsos. "Fast time sequence indexing for arbitrary Lp norms". In: *Proceedings of the 26th international conference on very large data bases*. Citeseer. 2000, pp. 385–394.

[132] D. Yu, X. Yu, Q. Hu, J. Liu, and A. Wu. "Dynamic time warping constraint learning for large margin nearest neighbor classification". In: *Information Sciences* (2011).

[133] D. Yu, G. Sheikholeslami, and A. Zhang. "Findout: finding outliers in very large datasets". In: *Knowledge and Information Systems* 4.4 (2002), pp. 387–412.

[134] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity search: the metric space approach*. Springer, 2006.