An adaptive framework for classification of concept drift with limited supervision

Piero Conca

PhD

The University of York
Department of Electronics

October 2012

Abstract

This thesis deals with the problem of classification of data affected by concept drift. In particular, it investigates the area of unsupervised model updating in which a classification model is updated without using information about the changing distributions of the classes.

An adaptive framework that contains an ensemble of classifiers is developed. These can be mature or naïve. In particular, only mature classifiers generate decisions, through majority voting, while naïve classifiers are candidate to become mature.

The first novelty of the proposed framework is a technique of feedback that combines concepts from ensemble-learning with concepts from self-training. In particular, naïve classifiers are trained using unlabelled data and labels generated by mature classifiers over that data, by means of voting. This technique allows updates of the model of the framework in absence of supervision, namely, without using the true classes of the data. The second novelty is a technique that infers the presence of concept drift by measuring the similarity between the decisions of mature classifiers and the decisions of naïve classifiers. When concept drift is inferred, a naïve classifier is selected to become mature, and a mature classifier is deleted.

A series of experiments are performed. They show that the framework can classify data with Gaussian distribution, and that this capability regards different classification techniques. The experiments also reveal that the framework cannot deal with the concept drift of a uniformly distributed dataset. Moreover, further experiments show that the inference of drift combines quick adapation with low false detections, thus leading to higher classification performance than comparative methods. However, this technique is not able to detect concept drift if the classes are separable.

Contents

Contents **2**

List of Tables **7**

List of Figures **9**

List of Algorithms **17**

Acknowledgements **18**

Declaration **19**

1 Introduction **20**

1.1 Scope 20

1.1.1 Classification 20

1.1.2 Concept drift 21

1.2 Contributions 22

1.3 Novelties 24

1.4 Thesis structure 24

2 Literature review **26**

2.1 Introduction 26

2.2 Machine learning 27

2.2.1 Bayesian decision theory 29

2.2.2 Supervised learning: classification 32

2.2.2.1 Design cycle for classification systems 34

2.2.2.2 Classification techniques 36

Support vector machines. 39

AISEC. 41

Naïve Bayes. . . . 44
Multilayer perceptron. . . . 45
C4.5. . . . 46
2.2.3 Ensemble classification . . . 46
Bagging. . . . 48
Boosting. . . . 48
2.2.4 Semi-supervised classification . . . 49
2.3 Concept drift . . . 51
2.3.1 Categories . . . 51
2.3.2 Datasets . . . 53
2.3.2.1 Examples of Datasets . . . 54
2.4 Classification of data with concept drift . . . 57
2.4.1 Supervised model updating . . . 57
2.4.1.1 Assumptions about the future distribution . . . 58
2.4.1.2 Adaptivity . . . 58
2.4.1.3 Strategies for concept drift . . . 60
2.4.1.4 Evolving techniques . . . 61
Ensemble-based. . . . 61
Instance weighting and instance selection. . . . 63
Feature space. . . . 65
Density-adaptive forgetting. . . . 66
2.4.1.5 Triggering methods . . . 67
Change detection. . . . 67
Training windows. . . . 68
2.4.1.6 Other approaches . . . 69
2.4.2 Semi-supervised model updating . . . 71
2.4.3 Unsupervised model updating . . . 73
2.5 Clustering of data with concept drift . . . 73
2.5.1 Introduction: clustering approaches . . . 73
2.5.2 Online clustering . . . 75
2.5.2.1 Prototype-based online clustering: CluStream . . . 75
2.5.2.2 Distance-based online clustering: DenStream . . . 76
2.5.3 Semi-supervised clustering . . . 77
2.6 Unsupervised drift detection . . . 78

2.7 Adaptive frameworks for classification 80
2.8 Summary 83

3 Adaptive framework **85**
3.1 Introduction 85
3.2 Motivation 86
3.3 Framework implementation for classification 89
3.3.1 Ensemble classifier 93
3.3.2 Decision feedback 95
3.3.3 Inference of drift 98
3.4 Summary 100

4 Experiments **102**
4.1 Introduction 102
4.2 Investigation of data requirements for unsupervised model updating . 103
4.2.1 Classification of Gaussian data with the SVM 103
4.2.1.1 Experimental setup 103
Gaussian data distribution. 104
Parametric settings. 105
Hypothesis. 107
Determining the number of runs. 110
4.2.1.2 Sensitivity Analysis 111
Correlation table and plots. 112
Interpretation of the sensitivity analysis. 115
4.2.1.3 Comparison with supervised techniques 118
4.2.1.4 Interpretation of the results 121
Training with supervised data. 121
Model updating using labels from the voting. 122
Effect of the replacement of the different classifiers of the ensemble. 122
Effect of the gradient on the training of new classifiers. 126
Summary. 127
4.2.2 Uniformly distributed data 127
4.2.2.1 Sensitivity Analysis 130
4.2.2.2 Comparison with supervised techniques 131

4.2.3 Conclusions . . . 134
4.3 Different classification techniques . . . 134
4.3.1 Classification of Gaussian data with AISEC . . . 134
4.3.1.1 Sensitivity analysis . . . 138
4.3.1.2 Interpretation of the sensitivity analysis . . . 144
4.3.1.3 Interpretation of the results . . . 147
4.3.2 Classification of Gaussian data with naïve Bayes . . . 148
4.3.2.1 Sensitivity analysis . . . 150
4.3.3 Classification of Gaussian data with the multilayer perceptron algorithm . . . 151
4.3.4 Sensitivity analysis . . . 154
4.3.5 Classification of Gaussian data with C4.5 . . . 156
4.3.6 Sensitivity analysis . . . 157
4.4 Comparison of the framework with unsupervised model updating techniques . . . 159
4.5 Data with multiple Gaussians with changing positions and characteristics . . . 166
4.5.1 Comparison with unsupervised techniques . . . 169
4.5.2 Preliminary investigation on the numbers of instances that are maintained in memory . . . 173
4.6 Analysis of the inference of drift . . . 174
4.6.1 Dataset with Gaussian data . . . 175
4.6.2 Comparison of the mechanism of drift inference with alternative drift detection techniques . . . 176
4.6.3 Dataset with separable classes and uniform distributions . . . 182
4.6.4 Comparison of the mechanism of drift inference with the unsupervised drift detection techniques . . . 184
4.7 Summary . . . 194

5 Conclusions **196**
5.1 Introduction . . . 196
5.2 Review of the research problem and the solution . . . 196
5.3 Evaluation of the work . . . 198
5.4 Future directions . . . 200

5.4.1 Inference of drift 200
5.4.2 Parametric adaptivity 201
5.4.3 Structural adaptivity 202
5.4.4 Different forms of feedback 203
5.4.5 Multiple techniques 203
5.5 Final comments 204

Appendices **205**

A Interpretation of the sensitivity analysis of the framework with SVM: additional plots **206**
A.1 Plots of the instance of the framework with the first parametric setting 207
A.2 Plots of an instance of the framework associated with the second parametric setting 211
A.3 Conclusions 216

B Interpretation of the sensitivity analysis of the framework with AISEC: additional plots **217**
B.1 Plots of the instances of the framework of the first cluster of samples 217
B.2 Plots of the instances of the framework of the second cluster of samples 221

C Sensitivity analyses: additional plots **226**
C.1 Sensitivity Analysis plots of the framework with SVM and Gaussian data 226
C.2 Sensitivity Analysis plots of the framework with AISEC and Gaussian data 231
C.3 Sensitivity Analysis plots of the framework with SVM and uniform data 234

D Data plots **239**

Acronyms **241**

Bibliography **242**

List of Tables

1 Concept drift of the dataset presented in [24]. The positions (μ_x, μ_y) and the standard deviations (σ_x, σ_y) of four Gaussian distributions $(C1, C2, C3$ and $C4)$ data vary according to three patterns, described by the linear equations of, respectively, the top, the middle and the bottom sections of the table, in a temporal sequence. 55

2 Parametric configuration of the framework for the experiment involving the SVM and Gaussian data. 106

3 Intervals of the parameters for the sensitivity analysis. 112

4 Correlation coefficients between parameters and output measures. . . 112

5 Parametric configurations of DWM and bagging+ADWIN for the experiment involving SVM and Gaussian data. 119

6 Configuration of the parameters of AISEC. 135

7 p-values at different stages of completion of a run. 138

8 Intervals of the paramaters for the sensitivity analysis. 139

9 Correlation coefficients between parameters and output measures. . . 140

10 Intervals of the parameters for the sensitivity analysis of the Framework with naïve Bayes as a base learner. 150

11 Correlation coefficients between parameters and output measures. . . 151

12 Correlation coefficients between parameters and output measures. . . 155

13 Correlation coefficients between parameters and output measures. . . 158

14 Parametric configurations of the of the unsupervised comparisons of the framework. 160

15 p-values of the framework with naïve Bayes and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use naïve Bayes). The p-values that are responsible for the rejection of the hypothesis are highlighted in bold type. 161

16 p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use the SVM). The p-values that are responsible for the rejection of the hypothesis are highlighted in bold type. 162
17 Parametric configuration of the framework for the experiment involving the SVM and Gaussian data. 166
18 Parametric configurations of the unsupervised comparisons of the framework. 170
19 p-values of the framework with naïve Bayes and the comparative for the measures of accuracy, precision and recall (the single-classifier variant also uses naïve Bayes). 171
20 p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variant also uses the SVM). 171
21 Parameters of the configurations of the framework for the different mechanisms of drift detection. 176
22 p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use naïve Bayes). 177
23 p-values of the framework and the comparative with the SVM for the measures of accuracy, precision and recall. 178
24 Parameters of the configurations of the framework for the different mechanisms of drift detection. 182
25 p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use naïve Bayes). 185
26 p-values of the framework and the comparative with the SVM for the measures of accuracy, precision and recall. 185

List of Figures

1 Visual representation of the probabilities involved in the Bayes theorem. From the top-left corner in clockwise order. Prior probabilities of class ω_1 and class ω_2. Class-conditional pdfs, they reflect the distributions of the classes. Notice that these are both normal distributions with the same standard deviation but different mean values. Joint pdfs $p(x|\omega_i)$ (green and red curves) are the product between prior probabilities and class-conditional pdfs; their sum is the class-unconditional pdf $p(x)$ (blue line). Posterior probabilites $P(\omega_i|x)$, that reflect the probability of a value belonging to a class are depicted in the bottom-left plot. Posterior probabilites are used to determine the decision boundary β (grey line). 31

2 The areas highlighted in red represent the error of assigning class ω_1 to instances of class ω_2. Alternately, the areas highlighted in green reflect the error of assigning class ω_2 to instances of class ω_1. The boundary in the plot of case 1 of the figure maximises the posterior probabilites and, as a consequence, minimises the risk of assigning an instance to the wrong class. In fact, if a different decision boundary is chosen, this error increases, as shown for case 2. 33

3 Flowchart of the design of a classification system (adapted from [27, 54]). 34

4 The linear classifier *perceptron* (a) combines the inputs $x_1, \ldots, x_n$ linearly by means of a set of weights $w_1, \ldots, w_n$ and a bias factor b. The output y is generated by discretising the linear function through a step function. Artificial neural networks (b) combine multiple perceptrons to solve nonlinear problems. The function applied to the nodes of the network is normally a sigmoid function, with the exception of the output node. 37

5 Example of a decision tree. In order to classify fruits the features *color*, *shape*, *size*, *taste* are used. At each node, the feature associated with the node is questioned, its value determines what node should be visited next. In this example, the root of the node is associated with the feature *color* (image from [27]). 38

6 Example of an ensemble classifier. Since the three base classifiers are linear, none of them is able to classify correctly the blue and the red instances. However their combination through majority voting (bold line) allows to discriminate with the highest accuracy the instances of the blue class from the instances of the red class. 48

7 Visual representation of the dataset proposed in [24]. 56

8 Model of the adaptive framework (left) and a possible instantiation (right) [51]. In the model of the framework (left), the input information vector IV_{input} is processed by a set of stovepipes. Their output is combined by the data fusion module to produce the ouput vector IV_{output}, which can also be fed-back to the input. The diagram on the left shows a possible instantiation of the framework. 82

9 UML class diagram of the framework. 91

10 Feedback of the decisions of the framework. 98

11 Example of the matrix of the distances M for an ensemble consisting of four mature and four naïve stovepipes. The matrix is square and symmetrical. The first four lines and columns are associated with mature stovepipes, while the remaining are associated with naïve stovepipes. The distances among mature stovepipes, naïve stovepipes, mature and naïve stovepipes are calculated by averaging the values in the areas highlighed using respectively the colors red, green and blue. 100

12 Representation of the Gaussian data. Distribution at the beginning of the experiment (a) and at the end (b). Notice that the position of the clusters moves along the x axis, but their relative position does not change. 105

13 Comparisons of the values of accuracy, precision and recall of the adaptive framework and the static framework. Notice that different scales are used. 109

14 A-test results. Despite the lowest scores are obtained for 200 runs, we considered that repeating the same experiment 50 times is sufficient, given that all the median accuracy values fall in a very narrow interval.111

15 Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall. 114

16 Representation of the model of the framework featuring the SVM at the beggining of a run(a), at the end of a successful run (b) and in case of failure (c). 117

17 Distributions of the values of accuracy, precision and recall of the framework, of bagging+ADWIN, and of DWM over the Gaussian data stream. Notice that different scales are used. 120

18 Data distribution of the supervised training phase. The green line and the red line represent respectively the distributions of the negative class and the positive class. Their sum, the class-unconditional distribution, is represented by the dotted line. 121

19 In the online phase, new naïve classifiers are trained using labels from the voting. Therefore, the distribution of the data and their labels can be described by $p'(x, \text{“}+1\text{”})$ and $p'(x, \text{“}-1\text{”})$”. 122

20 Probability density distributions of class “+1” (red line), “−1” (green line) in the online phase. The grey lines and the black line represent the boundaries of the three classifiers α, β, γ, while the blue line is the boundary of the naïve classifier ν. In particular, the boundary of β conincides with the boundary of the model of the framework. . . . 123

21 Effect of the replacement of the naïve classifier ν with the mature classifiers α (top), β (middle), γ (bottom) on the boundary of the model of the framework (black line). The dotted grey line indicates a classifier that has been deleted. 125

22 Probability of points to lie within a distance "d" from the boundary β is higher for points classified as "$+1$" (top). The diagram at the bottom shows that, when enough data has been classified, the same data is used to train a new naïve classifier ν', which is shifted to the left with respect to β. 126

23 Representation of the uniform distribution at the beginning of the experiment (a) and at the end (b). The position of the positive class has shifted along the x axis. 128

24 A-test results. Small scores are obtained for 100 runs. However precision and recall, which are more significant than accuracy in this concetxt, are sufficiently low for 50 runs. 129

25 Comparison of the values of accuracy of the adaptive implementation and the static implentation of the framework using the SVM on the dataset with uniform distribution. 130

26 Latin hypercube sensitivity analysis plot of $th_{training}$ against accuracy, precision and recall. 131

27 Distributions of the values of accuracy, precision and recall of the framework, of bagging+ADWIN, and of DWM over the data with uniform distribution. The values of recall and precision of the framework indicate that it is not able to deal with the concept drift of this dataset. Notice that different scales are used. 133

28 A-test scores for the AISEC algorithm. The graph shows that for 30 runs, the A-test score is sufficiently small. 136

29 Values of accuracy in function of the percentage of completion of a run, for the dynamic implementation (top) and the static one (bottom). In correspondence of 50%, the distributions are very similar. . . 137

30 Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall. 141

31 Latin hypercube sensitivity analysis plot of $th_{training}$ against accuracy, precision and recall. 142

32 Latin hypercube sensitivity analysis plot of th_{ID} against accuracy, precision and recall. 144

33 Representation of the model of the framework featuring AISEC at the beggining of an experiment (a), at the end of a successful experiment (b) and in case of failure (c). 146
34 A-test results for the framework with naïve Bayes. The test determines that low difference between two different sets of output values is reached when 200 runs are performed. 148
35 Values of accuracy, precision and recall of the adaptive framework with naïve Bayes. 150
36 Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall. 152
37 A-test results for the framework with MLP. The test determines performing 200 runs is sufficient. 153
38 The overlap between the values of accuracy (top plot) and precision (bottom plot) of the adaptive of the static and the adaptive framework is limited. However the different sets of values lie is small intervals of the spaces of the parameters. 154
39 Plot of the sensitivity analysis of th_{online} against accuracy, precision and recall when the MLP is used. 156
40 A-test results for the framework with the algorithm C4.5. The test determines performing 200 runs is sufficient. 157
41 Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall. 158
42 Box and whisker plots of the values of accuracy of the techniques for unsupervised model updating. 163
43 Box and whisker plots of the values of precision of the techniques for unsupervised model updating. 164
44 Box and whisker plots of the values of recall of the techniques for unsupervised model updating. 165
45 A-test results of the framework using naïve Bayes. 167
46 Distributions of the values of accuracy of the static instance and the dynamic instance of the framework using naïve Bayes. 168
47 Values of accuracy of the static instance and the dynamic instance of the framework using the SVM. 169

48 Distributions of the values of accuracy, precision and recall of the unsupervised techniques. 172

49 Distributions of the values of accuracy of several instances of the single-classifier technique (with different window sizes) and the framework. 174

50 A-test results of the framework using naïve Bayes. 177

51 Distributions of the values of sensitivity, specificity and delay of detection of the unsupervised techniques. 179

52 Distributions of the values of accuracy, precision and recall of the unsupervised techniques. 181

53 Concept drift of the dataset with uniformly distributed classes. Red is used to indicate the distribution of the class of positive instances, while blue is used for the negative class. The data distribution does not change for the first quarter of the number of instances, as shown in the diagram (a). After that, the center of the negative distribution moves to position (4,1), as shown in diagram (b). The center of negative instances changes direction in the second half of the concept drift (diagram (c)), and reaches the final position depicted by diagram (d). 183

54 A-test results of the framework using naïve Bayes. 184

55 Distributions of the values of sensitivity, specificity and delay of detection of the unsupervised techniques. 186

56 Distributions of the values of accuracy, precision and recall of the unsupervised techniques. 188

57 Plots of the accuracy and plots of the detection of drift of the version of the framework with the mechanism of drift inference (top), of the framework with the test by Hido et al. (center) and of the framework with the Friedman-Rafsky test (bottom). 191

58 Plots of actual data and boundaries of the classifiers. The red points represent the positive instances, blue is used for the negative instance. The red lines represents the boundaries of the mature classifiers and the green lines represent the naïve classifiers. 192

59 First detection of drift. The two plots at the top are enlargements of the plots at the top of Figure 57. The plot at the bottom shows the mean distances between the mature classifiers d_M, between the na"ive classifiers d_M, and those between the mature and the naïve classifiers. The arrow highlights a spike of d_{MN} that is higher than d_M and d_N. That triggers the updating of the classification model, and the subsequent increment of the accuracy. 193

60 TP and TN in function of the percentage of completion of a run. . . 208

61 FP and FN in function of the percentage of completion of a run of the first parametric setting. After the start of a run, the number of FP decreases and that of FN increases with respect to their initial values (that correspond to 0% of completion of a run). 209

62 Accuracy, precision and recall in function of the percentage of completion of a run. While the the accuracy does not seem to vary consistently across a run, precision increase and recall decreases. . . . 210

63 TP and TN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5. 212

64 FP and FN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5. 213

65 Error rate against percentage of completion of a run for sample 2. Its values is complementary to the value of accuracy. 214

66 Accuracy, precision and recall in function of the percentage of completion of a run. 215

67 TP and TN in function of the percentage of completion of a run. A small decrement of TP and a small increment of TN are observed. . . 218

68 FP and FN in function of the percentage of completion of a run. . . 219

69 Accuracy, precision and recall in function of the percentage of completion of a run. Accuracy slightly decreases as soon as drift starts, precision increases and recall decreases. 220

70 TP and TN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5. 222

71 FP and FN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5. 223

72 Error rate against percentage of completion of a run for sample 2. Its values is complementary to the value of accuracy. 224
73 Accuracy, precision and recall in function of the percentage of completion of a run. 225

74 Latin hypercube sensitivity analysis plot of $th_{training}$ against accuracy, precision and recall. 227
75 Latin hypercube sensitivity analysis plots of th_{ID} against accuracy, precision and recall. 228
76 Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall. 229
77 Latin hypercube sensitivity analysis plot of $FIFOsize_{ID}$ against accuracy, precision and recall. 230
78 Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall. 232
79 Latin hypercube sensitivity analysis plot of $FIFOsize_{ID}$ against accuracy, precision and recall. 233
80 Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall. 235
81 Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall. 236
82 Latin hypercube sensitivity analysis plot of $FIFOsize_{ID}$ against accuracy, precision and recall. 237
83 Latin hypercube sensitivity analysis plot of th_{ID} against accuracy, precision and recall. 238

84 Example of data and boundaries. Red points indicate positive instances, while blue points are used for negative instances. The black lines represent the boundaries of the mature classifiers, while the orange dotted line maximises the distance between the center of the Gaussian distributions. At the start of the run (a) the black lines and the orange line have a similar position. After concept drift has started, the black lines are on the right of the orange one. 240

List of Algorithms

1 Pseudo-code of the training phase of the AISEC 42
2 Pseudo-code of the classfication phase of the AISEC 43
3 Pseudo-code of the updating of the detectors of AISEC 43
4 Pseudo-code of the clone-mutate phase of AISEC 44
5 Pseudo-code of the training phase of the framework 94
6 Pseudo-code of the on-line phase of the framework 97
7 Pseudo-code of the mechanism of drift inference 99

Acknowledgements

I would like to thank:

- my family and especially my parents who were always there when I needed help;
- my closest friends: Paolo, Angelo, Gaetano, Annarita, Ilaria, Barbara, Brigid, Joan, Lorenzo, Silvana, Antonio and Francesco;
- the NCR corporation and the University of York, for sponsoring my PhD. In particular, i wish to thank Simon Forrest and Heather McCracken for their help;
- my supervisors: Jon Timmis and Rogerio de Lémos;

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

26 Mar 2014

Chapter 1
Introduction

This thesis investigates the reduction of supervision for classification of data affected by concept drift, by proposing an adaptive framework that can update its model in absence of supervision. This chapter scopes the problem to be solved, outlines the solution, and identifies what is the novelty of this work and its contribution to the field. This chapter also presents an outline of the rest of the thesis.

1.1 Scope

1.1.1 Classification

Machine learning is the discipline that is concerned with the development of computer programs that are capable of learning [61]. *Classification*, also known as *pattern recognition*, is amongst the most studied areas of machine learning. The problem of classification consists of recording observations from a system along with information about what those observations represent. This information is analysed to identify patterns that can be used later on to discriminate between new observations [11, 27].

More precisely, the system under examination is probed by a set of vectors of *features* or *attributes*. Features can contain different types of data, and a vector may contain mixed types. An additional attribute, known as the "class" attribute, is attached to feature vectors. It contains values from a discrete set that provide information about what a vector represents.

Starting from a set of vectors of attributes and their relative classes, the problem of classification consists of generating, or "training", a *model* from that data. A model is a function that associates vectors of features with classes. Once it is established, it can be used to determine the class of unlabelled instances, namely,

instances consisting solely of vectors of attributes.

A way to perform classification is by using ensemble classifiers. They combine multiple classifiers and a technique to fuse their decisions. Ensemble classifiers can offer a series of advantages over single classifiers, such as increased generalisation, incremental learning (e.g., from large datasets), combination of different techniques and therefore heterogeneous data types, and the possibility of parallelisation [72].

1.1.2 Concept drift

Traditionally, classification assumes that the distribution of the data does not change. That means that the data used to train a model and the data used to test it are generated from the same distribution. This is not always the case, as in many applications, especially real-world ones, the system being examined might change, or perhaps the instruments used to collect data might be affected, for instance because of wear and tear. As a result, the data distribution should be expected to change. This problem is known as *concept drift*, and when it affects the data it causes static models to undergo performance degradation [84, 91]. A new model may be trained or the existing model may be updated if the true classes of the instances are released after these have been classified. However, this information, which is referred to as *supervision*, is not always provided.

The techniques that deal with concept drift can be categorised based on the amount of supervision they use. In particular, the categories of *supervised* techniques, *semi-supervised* techniques and *unsupervised* techniques can be distinguished. In order to update their models over time, supervised techniques require the true classes of the instances to be provided after classification. That could be problematic in some cases, as it requires the existence of an entity, the *oracle*, capable of determining the true class of each instance. Whether the oracle is a human or a machine (for example, an algorithm), supervision entails costs and its ability to provide supervision may depend on the data rate.

In order to overcome the difficulties associated with the provision of labels, some methods have considered the possibility of reducing the amount of supervision they use. In particular, semi-supervised techniques combine labelled data (for which supervision is provided) with unlabelled data (for which supervision is not provided) to update the model of a classifier. They exploit the information about the distributions of the single classes (class-conditional distributions) that is conveyed

by the labelled data, along with the information about the distribution of the data without considering the classes (class-unconditional distribution) that is conveyed by the unlabelled instances.

This issue needs to be examined carefully to comprehend whether supervision is a prerequisite for classification of data with concept drift, or if, in some cases, it can be avoided or substantially reduced without affecting the classification performance. For this reason, we need to distinguish between different scenarios. Let us consider classification in terms of the transformation of information. Every dataset has an underlying data distribution which determines the density of its data across the space of the attributes. Although a group of datasets that are generated from a common distribution are likely to be different, they should maintain information about their underlying distribution with a certain fidelity. The higher the number of instances that are sampled, the higher the information about the distribution is. In such a way, a model that is trained from a dataset also contains information concerning the distribution that generated the data. In fact, a model is a function that associates different classes to different regions of the space of the attributes.

In the light of these considerations, let us reinterpret how supervised algorithms for concept drift operate. Supervised techniques receive labelled data which conveys explicit information about the changing distributions of the classes. This data can be used to train a new model from scratch or to update an existing model on the basis of the instances that are misclassified. That information is not available to unsupervised techniques. However, there are scenarios in which the data instances without their classes carry information about the changing distribution. This information could be combined with "memory" about a previously observed distribution, that is conveyed by the current model, in order to generate an updated model of the changing distribution. The research literature shows that, in some cases, it is possible to update the model of a classifier without using supervision. In particular, a classifier that uses the prediction of its model at time t to train a model that will be used at time $t' > t$ is described in [78].

1.2 Contributions

We believe that a classifier should extract as much information as possible from the available training data regarding the classes. When the distribution changes, the

classifier should update the information of its model to classify incoming instances. In this way, a classifier should be able to adapt to concept drift without depending on an external entity that provides the true classes. The advantantages that could derive from the reduction of supervision would be multiple. For instance, the costs associated with labelling could be avoided or substantially reduced. Morevoer, a classifier would be promptly updated as soon as concept drift starts, while supervised techniques are affected by potential delay in the provision of labels.

This thesis further investigates the area of unsupervised model updating. For this purpose, we propose the implementation of an adaptive framework that is based on ensemble classification [51]. In addition, this thesis explores alternative ways to infer the presence of concept drift without making use of labels.

A set of experiments have been designed with the purpose of evaluating the framework. They account for an initial set of data with a stationary distribution, for which labels are provided. The testing phase is affected by concept drift and labels are not provided. In particular, the purpose of the experiments is to answer the following research questions:

1. Is it possible to maintain high classification performance on different types of data distributions?

2. Is the framework able to operate with different types of classification techniques?

3. Does the framework offer any advantage in terms of performance with respect to comparative techniques?

4. Is the mechanism of drift inference of the framework able to reveal the presence of drift?

5. Is the mechanism of drift inference of the framework capable of operating with different classification techniques and on different types of distributions?

6. Does the mechanism of inference of drift offer any advantage in terms of performance with respect to comparative techniques?

1.3 Novelties

The adaptive framework introduces two major novelties. Firstly, a mechanism of *feedback* of its decisions, which combines unlabelled data and the decisions generated by the classification model of the framework that are used for updating the model itself. After an initial training with supervised data, the technique of feedback updates the model of the framework without using supervision. The framework is in fact composed of multiple classifiers and can therefore be considered as an ensemble classifier. At each iteration, testing instances and the result of the voting of the classifiers are collected to train a new classifier. The model is then updated by adding the new classifier to the framework. The principle of feedback is similar to that proposed in [78]. However, the framework uses decisions generated by an ensemble of classifiers, rather than a single classifier.

Secondly, this thesis also introduces a mechanism of *drift inference* that monitors the similarity between the different streams of decisions generated by the classifiers. This mechanism drives the adaptability of the model of the framework. In fact, only when drift is inferred the model of the framework is actually updated.

1.4 Thesis structure

The rest of the thesis is structured as follows.

Chapter 2 presents an overview of the concepts that are required to understand the problem, namely, classification of data affected by concept drift with only initial provision of supervision, as well as our solution to the problem: an adaptive framework. The chapter begins with a definition of the problem of classification, which is revised later on to take into account changing distributions. The chapter also describes the different categories of concept drift and outlines the key solutions for handling concept drift. Moreover, it describes a selection of techniques that have been developed for dealing with concept drift. These are classed according to the amount of supervision they use. The chapter also describes the ideas presented in [51], which outlines a diagnostic framework that bases its adaptivity on the concepts of modularity and feedback. This has represented a solid source of inspiration for the implementation proposed in this thesis.

Chapter 3 outlines the major steps in the development of this implementation.

It describes the characteristics of the ensemble classifier and the two major novelties being introduced in this implementation of the framework: the mechanims of feedback of its decision and the inference of drift. It does so by analysing the code of the different procedures step by step.

Chapter 4 introduces a set experiments aimed at assessing the ability of the framework at dealing with concept drift under different conditions. They involve different classification techniques and different data distributions.

Chapter 5 presents the conclusions of this thesis.

Chapter 2
Literature review

2.1 Introduction

This chapter introduces the concepts related to the problem of classification of data streams affected by concept drift with limited supervision.

Machine learning is the discipline which is concerned with the development of programs that allow computers to learn, and one of its most studied problems is *classification* [54]. This is the problem of using data collected from a system and labels that determine what the data represent to generate a function or *model*, which is then used to associate labels with unlabelled data [11]. The applications of classification are numerous. Among these, a classifier can be trained to detect fraudulent bank transactions, natural language processing, anomalous internet activity or identify people from visual images [54, 63]. An assumption that is commonly made in classification is that the data is independently and identically distributed. However, when data is collected from a changing environment, and this is common in real-world systems, its distribution is likely to change. This problem is known as *concept drift* and it requires the model to be updated in order to avoid the degradation of classification performance [91]. The majority of classification techniques are not designed to deal with concept drift, and a variety of solutions have been developed. Generally, they require supervision. That is, information about the true class of a data instance is provided after the instance has been classified. This information drives model updates. A set of solutions have been developed to reduce the amount of supervision. Semi-supervised techniques compensate the scarcity of supervision with unlabelled data. Moreover, some techniques are able to update their model without supervision.

The rest of the chapter is structured as follows. Section 2.2 begins with an

overview of the field of machine learning. Attention will then move to the more specific problem of classification. A description of some of the most used classification techniques is given in Section 2.2.2.2. They include, among others, discriminant functions, decision trees and statistical methods. Section 2.3 defines concept drift, an artefact of data that is responsible for performance degradation in classifiers. It also describes the characteristics of the different types of drift. Section 2.4 reformulates the problem of classification to take into account concept drift and gives an overview of some of the supervised strategies that have been proposed in the literature to cope with this problem. The Sections 2.4.2 and 2.4.3 describe techniques that reduce the amount of supervision they require. Section 2.5 analyses the problem of clustering data streams. It concerns the generation of a model of a stream of drifting data without any knowledge about the labels of its instances. Section 2.5.3 describes semi-supervised clustering, a variant of clustering which use limited supervision to support the formation of clusters. Section 2.6 presents a selection of techniques for unsupervised detection of concept drift. A selection of frameworks that perform adaptive learning is presented in 2.7. Section 2.8 provides a summary of the chapter.

2.2 Machine learning

Machine learning can be defined as the discipline which is concerned with the development of computer programs that are able to learn. In [61], the following definition of learning for machines is given:

> A computer is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at task T, as measured by P, improves with experience E.

Machine learning is a very broad discipline, which includes the three major areas of *supervised learning*, *unsupervised learning* and *reinforcement learning*. These differ in the type of information computers are provided with and therefore in the mechanisms that they use to convert information into a representation of the world.

In *supervised learning*, experience is expressed by means of a set of observations of the world. Observations consist of input vectors of *attributes* or *features*, along with additional information in the form of output labels which state what those observations represent, for example what their *class* is. Based on this information

(feature vectors and respective labels), a supervised learning algorithm builds an internal representation of the world, which is used to associate labels to unseen instances. As an example, consider the problem of medical diagnosis [56, 67]. Let us assume that a hospital keeps records of a series of attributes about patients, such as age, sex, symptoms and results of clinical analysis. The database also contains details about the disease each patient is diagnosed with. These pieces of information can be used to train a classifier with the aim to discover combinations of values which can be indicative of a disease. Once trained, a model can be used to assist doctors in the diagnosis of patients.

Unsupervised learning differs from supervised learning in the fact that no information is provided about labels of vectors of attributes. Unsupervised learning is concerned with revealing how data is organized, for example, by assessing the importance of the attributes with regard to a particular task or by identifying groups or clusters of instances with similar characteristics. These problems are, respectively, studied by the areas known as *dimensionality reduction* and *clustering* [32]. More precisely, the goal of dimensionality reduction is to reduce complexity and improve readability of data without discarding data that could help maximise the performance in the task being considered. This can be achieved by simply ignoring redundant or irrelevant attributes, or by combination of the original attributes into a new set of attributes. As the name suggests, clustering is concerned with the identification of groups or clusters of observations which present similarities [6]. According to a distance measure, clusters are formed by grouping instances that are close together but far from other instances or clusters. Genomics is one of the applications of clustering, in which it is used to identify groups of genes that are co-expressed and, therefore, are likely to be related [33].

Semi-supervised learning is similar to supervised learning, with the difference that only part of the observations are associated with a label. The problem is to build a representation of the world by combining the information in labelled data with the information in unlabelled data.

It is not always possible to teach what decision should be taken given an observation, as in supervised learning; often because it is not known what is the best decision. However, rewards or punishments can be used to guide the learning process as in the case of *reinforcement learning*. Different from the other areas of machine learning, reinforcement learning is not only about gathering and analysing informa-

tion from the system under examination. The interaction is in fact bidirectional since the computer is able to modify the system itself. As a matter of fact, decisions take the form of *actions*, that cause the transition of the system from a state to another. Reinforcement learning has been applied fruitfully, for example, to game theory and robot control problems. In the game of checkers, algorithms based on reinforcement learning can play at the same level as good players [82, 81], while in the game of backgammon, programs compete or even outperform the top human players [77, 89, 90]. In this context, an effective way to "train" a program consists of making it play games against itself repeatedly. In robot control, this approach is used to drive the learning of robots in specific environments to find strategies that increase their chance of "survival".

The rest of this section is dedicated to supervised learning. This will define the problem and describe some of the strategies that have been developed to tackle it. Moreover, it also describes the steps that are involved in the design of a learning system. However, these are preceded by an introduction on Bayesian theory for decision making. The concepts it presents will be useful to frame the problem that is dealt with by this thesis.

2.2.1 Bayesian decision theory

Bayesian theory is a probabilistic theory which, among other things, is used to derive the probability of an event being true based on a set of obervations [11, 27]. This property makes the theory suitable to the problem of deciding what class to associate with an observation. In fact, provided that the underlying data distribution is known, Bayesian theory can minimise the probability of an observation being assigned to the wrong class. For simplicity, let us assume that the number of classes is limited to two, say ω_1 and ω_2.

The probability of an instance belonging to class ω_i can be derived using Bayes theorem:

$$P(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{p(x)} \tag{2.1}$$

$P(\omega_i|x)$ is called the *posterior probability* because it indicates the probability of ω_i being assigned to the right class after x has been observed. Intuitively, posterior probabilities can be used to assign a class to an instance, for example by choosing the class which maximises this probability. Later on, it will be shown that this choice is

well grounded since it minimises decision errors. The term $P(\omega_i)$ refers to the *prior probability*, which reflects the probability of a sample belonging to a class prior to any knowledge about that sample. If, for example, 60% of the samples that are generated have class ω_1, and the remaining ones have class ω_2, it follows that $P(\omega_1) = 0.6$ and $P(\omega_2) = 0.4$. The *likelihood function* or *class-conditional* probability density function (pdf) $P(x|\omega_i)$ indicates the distribution of samples from class ω_i across the input space. Finally, the pdf $p(x)$ is the *evidence* or *unconditional pdf* and it reflects the distribution of x independently of a particular class. The term $p(x)$ can be seen as a normalisation factor since it assures that the sum of the posterior probabilites of the classes is always 1. In fact, it can be expressed as:

$$p(x) = p(x|\omega_1)P(\omega_1) + p(x|\omega_2)P(\omega_2) \tag{2.2}$$

Substituting Equation (2.2) into Equation (2.1), posterior probabilites can be re-written in terms of only prior probabilites and class-conditional probabilites:

$$P(\omega_i|x) = \frac{p(x|\omega_i)P(\omega_i)}{p(x|\omega_1)P(\omega_1) + p(x|\omega_2)P(\omega_2)} \tag{2.3}$$

Figure 1 provides a graphical representation of the probabilites that have been described. The top-left plot displays the prior probabilites. In this case, the classes are unbalanced, as the majority of instances belong to class ω_1. The plot on the top-right shows the class-conditional pdfs, that represent the distributions of the classes. In the bottom-right plot we can see the joint pdfs $p(x, \omega_1)$ and $p(x, \omega_2)$ (respectively, red line and green line), such that $p(x, \omega_i) = p(x|\omega_i)P(\omega_i)$. They reflect the probability of instances of a class being selected when samples from both classes are drawn. Their sum (blue line) is the class-unconditional pdf. The bottom-left plot depicts how the posterior probabilites vary along the x axis. As stated above, intuitively, an instance should be labelled according to the class with the highest posterior probability. If we refer to the example depicted in Figure 1 (bottom left plot) the decision boundary corresponds to $x = \beta$. Therefore, values of x smaller than β are assigned to class ω_1, the remaining values to class ω_2.

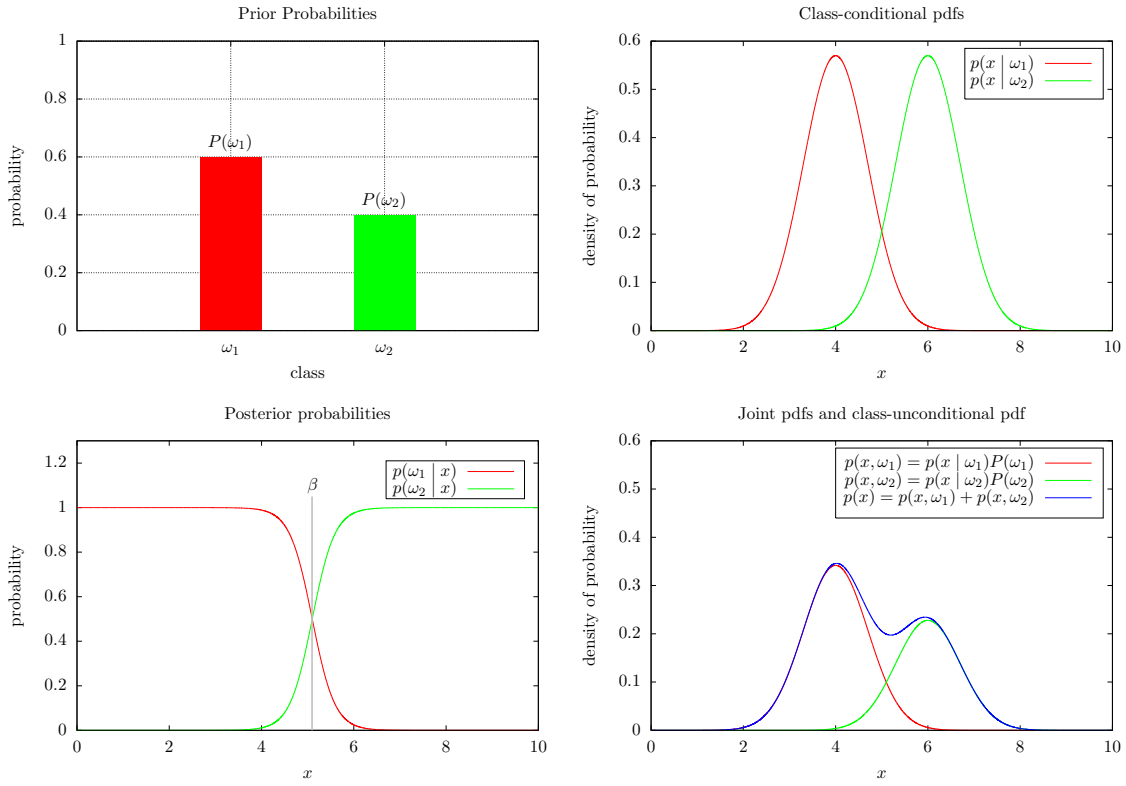


Figure 1: Visual representation of the probabilities involved in the Bayes theorem. From the top-left corner in clockwise order. Prior probabilities of class ω_1 and class ω_2. Class-conditional pdfs, they reflect the distributions of the classes. Notice that these are both normal distributions with the same standard deviation but different mean values. Joint pdfs $p(x|\omega_i)$ (green and red curves) are the product between prior probabilities and class-conditional pdfs; their sum is the class-unconditional pdf $p(x)$ (blue line). Posterior probabilites $P(\omega_i|x)$, that reflect the probability of a value belonging to a class are depicted in the bottom-left plot. Posterior probabilites are used to determine the decision boundary β (grey line).

Let us denote with $\mathcal{R}_1$ the subset of the input space in which instances are classified as ω_1, and $\mathcal{R}_2$ the remaining part in which the class associated with the instances is ω_2. The probability of an instance being associated with the wrong class can be expressed as:

$$P(error) = P(x \in \mathcal{R}_1, \omega_2) + P(x \in \mathcal{R}_2, \omega_1) =$$

$$\int_{\mathcal{R}_1} p(x, \omega_2)\, \mathrm{d}x + \int_{\mathcal{R}_2} p(x, \omega_1)\, \mathrm{d}x \tag{2.4}$$

The plot at the top of Figure 2 shows the errors in the case in which $\mathcal{R}_1$ and $\mathcal{R}_2$ are chosen in order to maximise the posterior probabilities. The area highlighted in

red represents $\int_{\mathcal{R}_1} p(x, \omega_2)\,\mathrm{d}x$, that is, the probability of instances of class ω_2 being assigned to class ω_1. Similarly, the green area represents $\int_{\mathcal{R}_2} p(x, \omega_1)\,\mathrm{d}x$, that is, the probability of instances of class ω_1 being assigned to class ω_2. If a different decision boundary is chosen, the probability of error increases, as it can be seen in the second case (bottom plot of Figure 2). In fact, in that case, the sum of the highlighted areas (which is proportional to the error) is higher than the sum of the areas of the first case (top plot of Figure 2). This implies that an instance should be assigned the class with the highest posterior probability in order to minimise decision errors.

2.2.2 Supervised learning: classification

Although Bayesian theory states how to minimise the error in assigning the wrong class to an observation, in most practical problems the underlying data distribution is unknown, and therefore the theory is not applicable.

However, if it is possible to collect information from a system in the form of instances coupled with labels, this information can then be used to label unseen instances. Supervised learning deals with this problem. In a formal way, given a dataset containing couples in the form $(\mathbf{x_i}, y_i)$ generated from an unknown probability distribution $\mathcal{D}$, where $\mathbf{x_i} \in X$ is a vector of input features and $y_i \in Y$ is an output label, supervised learning is concerned with finding a function $h : X \to Y$, which maps inputs into outputs. The function h is called the *hypothesis* or the *model*. If the set Y contains a discrete number of elements, the problem is called *classification*. If the set Y is continuous, the problem is called *regression*.

Supervised learning is traditionally divided into two phases. The purpose of the *training* phase is to generate a model from the couples $(\mathbf{x_i}, y_i)$. The criteria to generate a model depend on the *classification technique* (or *learner*) being used. Section 2.2.2.2 discusses some of the major categories of techniques. In the *testing phase*, *unlabelled* instances $\mathbf{x}_i$, are presented to the classifier, which uses the model to associate a class y_i to $\mathbf{x}_i$. However, the distinction between training phase and testing phase is not strict. In fact, there are classifiers which are able to modify their model if new training data are available. Given an hypothesis and a batch of training data, these classifiers output an updated hypothesis. We can distinguish *incremental learning*, which builds an hypothesis incrementally, from *adaptive learning* which, in addition, can cope with changes of the data distribution [60, 106]. Incremental classifiers that can learn a single labelled instance are called *online* classifiers [60, 70].

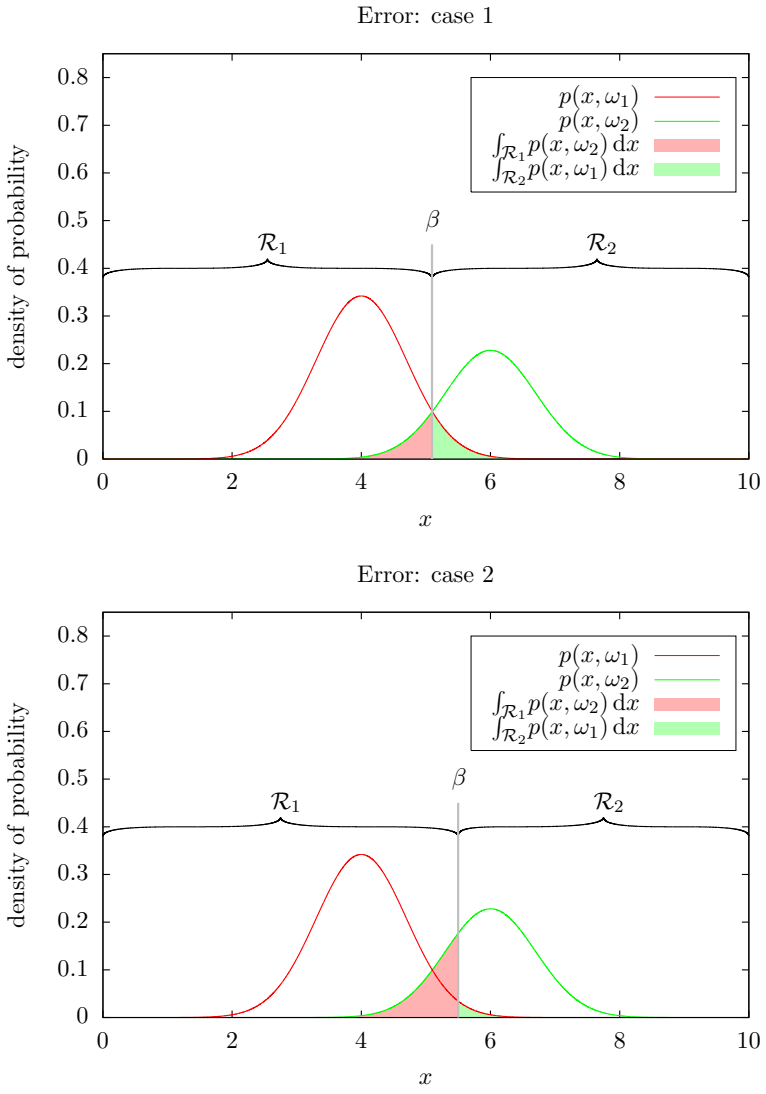


Figure 2: The areas highlighted in red represent the error of assigning class ω_1 to instances of class ω_2. Alternately, the areas highlighted in green reflect the error of assigning class ω_2 to instances of class ω_1. The boundary in the plot of case 1 of the figure maximises the posterior probabilites and, as a consequence, minimises the risk of assigning an instance to the wrong class. In fact, if a different decision boundary is chosen, this error increases, as shown for case 2.

Ideally, the higher the amount and quality of data to train a classifier is, the better. In practice, collecting or labelling sufficiently large volumes of data is not always feasible. Moreover, other factors such as noise or missing attributes might complicate the situation. This issue need to be taken into account during training. A model should abstract the right level of information from the data in order to provide sastisfactory classification performance over unseen instances. Two major problems can affect the ability of a classifier to "generalise" the training data. *Underfitting* occurs when the complexity of a model does not match that of the data. Overfitting occurs when a model is tailored to the training data so that it has low error over its instances but it misclassifies novel data. More precisely, a model h is said to overfit the data, when it is possible to find another model h' that has lower classification performance over the training data and higher performance over testing data [54]. *Underfitting* occurs when the complexity of a model does not match that of the data.

2.2.2.1 Design cycle for classification systems

After a general introduction to classification, we describe the process of tailoring a classification system to a specific problem. Although classification deals with a large diversity of problems and the design of a classification system is strongly dependent on the task in hand, it is possible to abstract a set of general, high-level steps which are common to different classification problems.

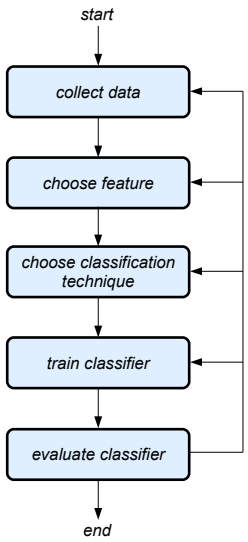


Figure 3: Flowchart of the design of a classification system (adapted from [27, 54]).

The first phase of the design cycle is *data collection*. This has the purpose of delivering a dataset which contains a proper level of information about the observed system to the following stage(s). The support of an expert in the field being considered may be required [54]. This phase can represent a large part of the cost of the final system [27]. Therefore a trade-off between cost, time, amount of data collected and number and type of features is often needed. In some cases the data is available, but it may be necessary to gather it. For example, consider the medical diagnosis system outlined earlier. Different hospitals might contain records about a specific disease, this data needs to be retrieved. In other cases data might not be available and therefore needs to be collected. On one hand, this could be problematic and it could increase costs. On the other hand, it gives more freedom over the choice of a set of features. As an example, let us consider the diagnosis again, but this time of an electro-mechanical device. In this scenario, the types and the positioning of the data sources are decided by the designer of the monitoring system.

In some cases, *pre-processing* is needed to "prepare" data for the following stage. Preprocessing may involve filling missing fields in the patients' records or reducing noise if microphone and/or cameras are used in the second scenario. In some cases large amounts of data are collected, and it is necessary to reduce the number of instances, for example by eliminating duplicates and/or by selecting a subset of instances that are highly representative of the initial set.

After collection, raw data is not always ready to be used. In particular, a new set of features might be extracted from the original set of features in order to filter out irrelevant information and make data readable to a classifier. In this process we can distinguish between *feature selection*, which eliminates irrelevant and redundant features, and *feature construction*, which is concerned with the combination of existing features to produce new features [51, 54]. By decreasing the size of the feature set and/or combining features it may possible to improve data readability, reduce the amount of computation and increase the performance of the classification process [54]. Unsupervised learning and in particular dimensionality reduction are often used to perform feature extraction, this is a case of *principal component analysis* (PCA) which extracts a set of features with low correlation from the original set [87].

Generally, a specific classification problem poses a series of design constraints

on the choice of a classification technique. For example, the available hardware might limit the choice to techniques with low computational requirements, or for example some problems might require the model to be easily inspectable by a user (*transparency*). An interesting taxonomy of classification techniques based on their properties is presented in [54]. Amongst others, the criteria used for assessment are: accuracy, training and classification speed, tolerance to noise, risk of overfitting, incremental learning.

The following steps deal with the creation of a classifier. Training involves the choice of a set of parameters and the selection of a training dataset to generate a classifier. Validation deals with the evaluation of its performance. If enough data is available, this can be split so that part of the data trains a classifier and the remaining instances are used to validate the classification performance of the classifier. When data is scarce and, therefore, using separate data for evaluation is unfeasible, it is still possible to estimate the generalisation capability of a classifier, for example by means of *cross-validation*. Training data is divided into N equally-sized and separate data partitions. These are used to generate N classifiers as follows, the i^{th} classifier is trained using all the data but the i^{th} partition, and the instances of that partition serve to test the performance of the classifier. The average of the errors of the classifiers gives an estimate of the performance of the selected technique and parametric setting. *Leave-one-out* validation is a particular case of cross-validation in which data partitions contain a single instance. Although cross-validation is very useful when data is scarce, it is computationally expensive.

2.2.2.2 Classification techniques

Classification techniques differ in the mechanisms of model construction and interrogation. This section gives an overview of some of the major approaches.

The techniques that are based on *discriminant functions* rely on a function which divides the input space into two regions (for a binary classification problem), each of which is assigned to a class. Unlabelled instances are classified by checking to which region they belong. The choice of a discriminant function depends on the problem at hand. Discriminant functions can be divided into *linear* and *nonlinear*. The perceptron (Figure 4 (a)), a technique (loosely) inspired by the functioning of neurons, is a typical example of the first category [27]. The model consists of a linear function "wrapped" by a step function. Weights and bias of the linear

function determine the model and are set by performing multiple passes over the training data, where at each iteration the weights are adjusted to reduce the errors of the classifier. When data is not *linearly separable*, that is, when no line or hyperplane

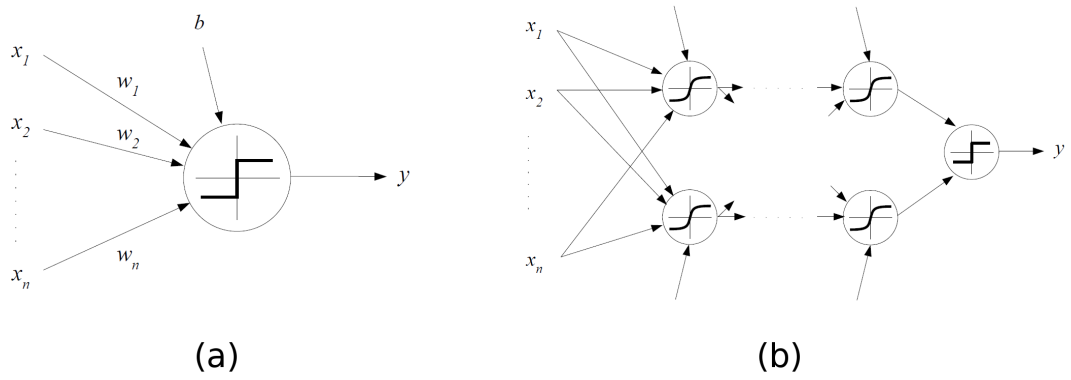


Figure 4: The linear classifier *perceptron* (a) combines the inputs $x_1, \ldots, x_n$ linearly by means of a set of weights $w_1, \ldots, w_n$ and a bias factor b. The output y is generated by discretising the linear function through a step function. Artificial neural networks (b) combine multiple perceptrons to solve nonlinear problems. The function applied to the nodes of the network is normally a sigmoid function, with the exception of the output node.

which neatly separates the classes exists, a nonlinear function is needed. The multilayer perceptron algorithm (MLP), also known as an artificial neural network (ANN), relies on a network of perceptrons that generates a nonlinear discriminant function, as depicted in Figure 4 (a). Support vector machines (SVM) are a linear technique that can also be used to classify nonlinear data [93]. Given a set of labelled data, the technique of SVM searches for the hyperplane with maximum distance from the points of the different classes. SVM resolve the problem of classifying nonlinearly separable data by mapping input instances into a higher dimensional space where the problem can be reduced to the linear case.

Decision trees classify data instances by considering one feature at a time. Using this sequential interrogation, priority is given to the features that can help classifying an instance as effectively and quickly as possible. Decision trees are hierarchical structures, in which *nodes* are associated with features. *Branches* coming out of a node are values that the feature of the node can assume and each terminal node or *leaf* is labelled with a class. Decisions are made in a top-down fashion. Given a feature vector, starting from the root of the tree, for each node, the feature associated with the node is selected from the input vector. Its value dictates which branch

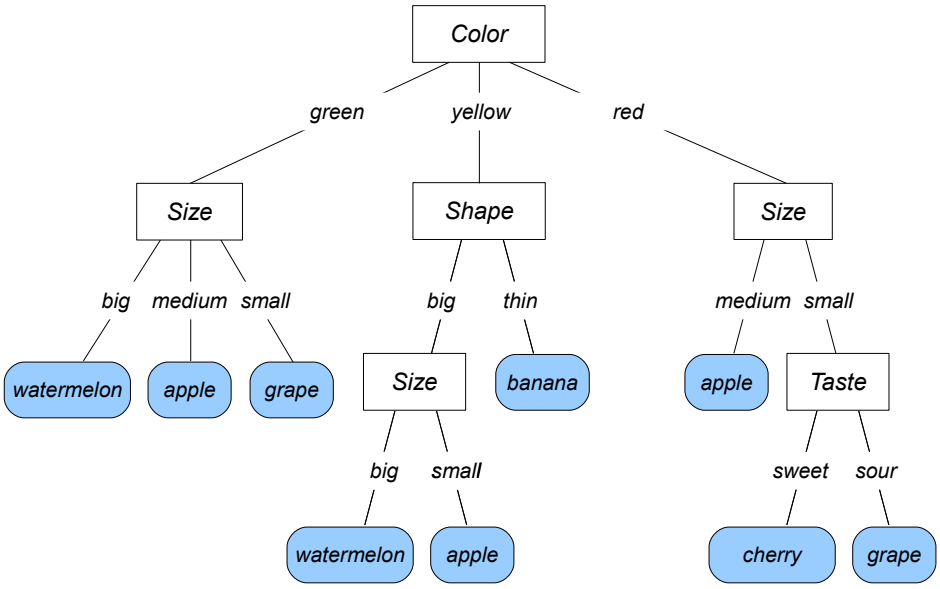


Figure 5: Example of a decision tree. In order to classify fruits the features *color*, *shape*, *size*, *taste* are used. At each node, the feature associated with the node is questioned, its value determines what node should be visited next. In this example, the root of the node is associated with the feature *color* (image from [27]).

(and therefore node) will be considered next. This process is iterated until a leaf is reached, the label of the leaf determines the class of the input vector. Decision trees are also nonmetric methods, which means they do not necessarily rely on the concept of distance and can therefore classify instances that do not contain numbers [11]. For example, the decision tree of Figure 5 handles attributes such as *shape* and *taste*, which can respectively assume the values *round* and *thin* or *sweet* and *sour*.

One point in common in all these strategies is that, after generating a model, training data is discarded. A different case is represented by instance-based techniques. These techniques do not process training data: they simply store it. Therefore training data becomes part of the model, and it is used to classify unlabelled instances. An immediate consequence is that there is no training time. However, since the computational burden is postponed to the testing phase, time and space complexities of this type of algorithm are strongly affected [54]. For this reason, instance-based classifiers are also called *lazy* classifiers, in contrast to *eager* tech-

niques [54, 61].

After this brief introduction to classification methods, a closer look will be taken at some specific techniques. In particular, attention will be given to SVM [93] and the instance-based techniques AISEC [85], naïve Bayes [62], the multilayer perceptron [76] and the incremental decision tree C4.5 [75].

Support vector machines. The support vector machine (SVM) is one of the most studied techniques for classification [54]. For the sake of clarity, explanation of the principles which govern the functioning of this technique will be divided into three phases. Firstly, SVM for linearly separable classes will be described. This is followed by an introduction to SVM for nonseparable data and then the kernel trick which allows SVM to classify nonlinear data. SVM were initially developed for classification of datasets with two linearly separable classes [92]. A dataset of l couples $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{1, -1\}$, in which the subsets of points of different classes are denoted with $I = \{\mathbf{x}_i | y_i = +1\}$ and $II = \{\mathbf{x}_i | y_i = -1\}$, is linearly separable if it is possible to find a hyperplane defined by the linear equation with coefficients $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$:

$$(\mathbf{x} \cdot \mathbf{w}) = b \tag{2.5}$$

such that:

$$(\mathbf{x} \cdot \mathbf{w}) > b \quad if \quad \mathbf{x}_i \in I \quad and \quad (\mathbf{x} \cdot \mathbf{w}) < b \quad if \quad \mathbf{x}_i \in II \tag{2.6}$$

Among all the possible separating hyperplanes SVM look for the hyperplane which maximizes the margins, defined as the distances of the hyperplane from the support vectors, which are the points of I that lie on the hyperplane $\mathbf{x}_i \cdot \mathbf{w} + b = +1$ and the points of II that lie on the hyperplane $\mathbf{x}_i \cdot \mathbf{w} + b = -1$ [16]. The distance of the first hyperplane from the origin is $|1 - b|/||\mathbf{w}||$, the distance of the second hyperplane from the origin is $| - 1 - b|/||\mathbf{w}||$, so that the size of each margin is $1/||\mathbf{w}||$ and, therefore, the distance between every side-hyperplane is $2/||\mathbf{w}||$. The problem of finding the best separating hyperplane is reformulated into the problem of finding a hyperplane which maximises the margins by minimising $||\mathbf{w}||^{\mathbf{2}}$ and satisfies the constraints:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 \quad for \quad y_i = +1 \tag{2.7}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \quad for \quad y_i = -1 \tag{2.8}$$

which can be combined in a single expression:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \forall i \tag{2.9}$$

When an unlabelled instance $\mathbf{x}_i$ is presented, it is classified depending on $sign(\mathbf{w} \cdot \mathbf{x}_i)$. If this is positive, the instance will be associated with the class "$+1$", otherwise with the class "-1". The problem can also rewritten by means of Lagrangian multiplers. In this case the function to minimise is:

$$W(\alpha) = -\sum_{i=1}^{n} \alpha_i + \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) \tag{2.10}$$

subject to the constraints:

$$\sum_{i=1}^{n} y_i \alpha_i = 0 \tag{2.11}$$

$$0 \leq \alpha_i \leq C \quad \forall i \tag{2.12}$$

the separating hyperplane is derived through:

$$\mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{l} \alpha_i y_i \quad and \quad b = y_{sv} - \mathbf{w} \cdot \mathbf{x}_{sv} \tag{2.13}$$

where $(\mathbf{x}_{sv}, y_{sv})$ is a support vector such that $\alpha_{sv} < C$.

There are cases in which it is not possible to find a hyperplane that neatly separates the instances of different classes. To overcome this limitation, the use of slack variables ξ_i, with $i = 1, \ldots, n$ was proposed in [22]. These are subtracted from the distance of the instances responsible for nonseparability from the hyperplane. This technique, which is known as *soft-margins* SVM, reduces the problem to the separable case. For soft-margins SVM, the problem of finding the best separating hyperplane can be reformulated as the minimisation of:

$$\frac{1}{2}||\mathbf{w}^2|| + C \sum_{i=1}^{n} \xi_i \tag{2.14}$$

subject to the constraints:

$$y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 1 - \xi_i \quad \forall i \tag{2.15}$$

$$\xi_i > 0 \quad \forall i \tag{2.16}$$

In either case (linearly separable or soft-margins SVM), the discrimination function is a linear combination of the input variables (it is a hyperplane). Some data distributions require non-linear functions in order to effectively classify their instances.

A solution to this problem is proposed in [13]. The data points are mapped into a higher dimensional space where they can be separated by means of a hyperplane. Technically, this is achieved by simply replacing the dot product in the Expression 2.7 with a kernel function. Some of the most used types of kernel functions are: polynomial, radial-basis, sigmoid and hyperbole [16].

The technique of transductive SVM deals with data that is only partially labelled. The problem is solved by finding a set of labels for the unlabelled training instances, along with $\mathbf{w}$ and b, to maximise the margins of the hyperplane [93].

The limited classification time and the high scalability are the main advantages of SVM. The main disadvantage is the time required for training [54, 73].

AISEC. (Artificial Immune System for Email Classification) [85] is an instance-based classification algorithm inspired by the clonal selection principle of the immune system of vertebrates [17]. This technique was originally created to tackle the problem of classification of incoming emails to a mailbox based on the interests of the user. However, the version that will be considered here was specifically modified to handle real numbers. The model consists of a set of detectors, a detector is a point in a multi-dimensional space and a threshold, which is the same for all the detectors. There are two sets of detectors, *memory* and *naïve*. Memory detectors are those which have shown better performance and therefore have a longer lifespan. Naïve detectors can become memory detectors if they classify instance with high *affinity*. The measure of affinity denotes how close a detector and an instance are in the feature space. It is inversely proportional to the Euclidean distance. During the training phase IMC elements of the training set are randomly selected to populate the pool of memory detectors. Such detectors are also assigned a stimulation count MCS which determines their lifetime. If the affinity between a memory detector and training instance is higher than the affinity threshold AT, then a set of clones is created using *cloneMutate(mem_detector, testing_instance)*. The procedure *cloneMutate(detector, instance)*, in fact, generates a set mutated clones, number and mutation rate of the clones are respectively directly and inversely proportional to the affinity between *detector* and *instance*. If the affinity of a clone is higher than the affinity of the memory detector which generated it, then the clone is added to the naïve pool and its stimulation count is set to NCS.

Unseen instances are compared with both memory and naïve detectors in a

sequential way. If the distance of the instance from the center of the detector is less than the threshold CT, then the instance is classified accordingly, otherwise the instance is assigned to the other class.

AISEC also accounts for adaptivity. In this case an oracle intermittently provides the true classes of the instances and this information is used to update the set of detectors by deleting those ones which fail to classify correctly and creating new ones to classify unseen instances. In particular, if an instance is classified correctly, after incrementing the stimulation count of the naïve detectors, a set of clones of the best naïve detector (that with the highest affinity to the instance) is added to the naïve pool and the best naïve detector is recomputed. If its affinity is higher than the affinity of the "best" memory detector with the instance, then the best naïve detector becomes a memory detector and its stimulation count is updated to MCS. In addition, the size of the memory pool is reduced by eliminating the detectors that are similar to the new entry.
If, instead, the instance is misclassified, all the detectors that are "close" to the instance are removed. Regardless of whether an instance has been classified correctly or not, the stimulation count of the naïve detectors is decremented, and if this reaches 0 the detector is deleted.

Algorithm 1 Pseudo-code of the training phase of the AISEC

```
1: procedure TRAIN(D_train, CT, AT, CC, MC, NCS, MCS, IMC)        ▷ inputs
2:     memory_pool ← ∅
3:     naive_pool ← ∅
4:     initialise memory_pool with IMC random elements from D_train
5:     for mem_detector in memory_pool do
6:         mem_detector's stimulation count ← MCS
7:     for instance training_instance in D_train do
8:         for mem_detector in memory_pool do
9:             if affinity(mem_detector, training_instance) > AT then
10:                clones ← cloneMutate(mem_detector, ti)
11:                for clone in clones do
12:                    if affinity(clone, training_instance) ≥ affinity(mem_detector, training_instance)
   then
13:                        naive_pool ← naive_pool ∪ {clone}
14:    return memory_detector, naive_detector
```

Algorithm 2 Pseudo-code of the classfication phase of the AISEC

```
1: procedure CLASSIFY(testing_instance)
2:     for detector in memory_pool ∪ naive_pool do
3:         if affinity(detector, testing_instance) > CT then
4:             return 1
5:     return -1
```

Algorithm 3 Pseudo-code of the updating of the detectors of AISEC

```
1: procedure UPDATEPOPULATION(testing_instance)
2:     if classification was correct then
3:         for naive_detector in naive_pool do
4:             if affinity(testing_instance, naive_detector) > AT then
5:                 increment naive_detector's stimulation count by 1
6:         naive_best ← naive detector with highest affinity to testing_instance
7:         naive_pool ← naive_pool ∪ cloneMutate(naive_best, testing_instance)
8:         memory_best ← memory detector with highest affinity to testing_instance
9:         if affinity(naive_best, testing_instance) > affinity(memory_best, test-
   ing_instance) then
10:            remove naive_best from naive_pool
11:            naive_best's stimulation count ← MCS
12:            memory_pool ← memory_pool ∪ {naive_best}
13:            for memory_detector ∈ memory_pool do
14:                if affinity(naive_best, memory_detector) > AT then
15:                    decrement memory_detector's stimulation count
16:    else
17:        for detector in memory_pool ∪ naive_pool do
18:            if affinity(detector, testing_instance) > AT then
19:                delete detector from system
20:    for naive_detector in naive_detector do
21:        decrement naive_detector's stimulation count
22:    for detector ∈ memory_pool ∪ naive_pool do
23:        if detector's stimulation count = 0 then
24:            delete detector
```

Algorithm 4 Pseudo-code of the clone-mutate phase of AISEC

```
1: procedure CLONEMUTATE(detector, instance)
2:     clones = ∅
3:     aff = affinity(detector, instance)
4:     num_clones = int(aff*CC)                    ▷ number of clones to generate
5:     mutate_size = float((1−aff) * MC)           ▷ mutations for each clone
6:     for i in num_clones do
7:         new_clone ← detector
8:         for j in [1, ..., |new_clone|] do       ▷ for each feature j of new_clone
9:             new_clone[j] = new_clone[j] + random.gauss(0, mutatesize)
10:            clones = clones ∪ {new_clone}
11:    return clones
```

Naïve Bayes. Section 2.2.1 introduced Bayesian theory, a probabilistic theory that states how to perform optimal decisions about the class of an instance, assuming that the distributions of the classes are known. Naïve Bayes is a classification technique that draws upon Bayesian theory.

For the sake of this explanation, let us consider a multidimensional formulation of Bayesian theory, where x_i indicates the i^{th} attribute of a vector $\mathbf{x} \in \mathbb{R}^n$. In particular, Equation 2.1 can be rewritten as:

$$P(\omega_j|x_1, x_2, \ldots, x_n) = \frac{p(x_1, x_2, \ldots, x_n|\omega_j)P(\omega_j)}{p(x_1, x_2, \ldots, x_n)} \tag{2.17}$$

The technique of naïve Bayes works under the assumption that the attributes are independent, that means that the probability of an instance belonging to a class does not depend on the values assumed by the other attributes, for example:

$$p(x_i \mid \omega_j, x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n-1}) = p(x_i \mid \omega_j). \tag{2.18}$$

Applying iteratively the following rule, also known as the "chain" rule:

$$\begin{aligned} p(x_1, x_2, \ldots, x_n|\omega_j) &= p(x_1|\omega_j)p(x_2, \ldots, x_n|\omega_j, x_1) = \ldots \\ \cdots &= p(x_1|\omega_j)p(x_2|\omega_j, x_1) \ldots p(x_n \mid \omega_j, x_1, x_2, \ldots, x_{n-1}), \end{aligned} \tag{2.19}$$

and assuming that the attributes are independent (Equation 2.18), the Bayesian formula (2.17) can be rewritten as:

$$P(\omega_j|x_1, x_2, \ldots, x_n) = \frac{\prod_{i=1}^{n} p(x_i|\omega_j)P(\omega_j)}{p(x_1, x_2, \ldots, x_n)} \tag{2.20}$$

In order to identify the class $\hat{\omega}$ with the highest posterior probability, it is sufficient to choose the class with the highest numerator of Equation 2.20, as the denominator

is the same for all classes:

$$\hat{\omega} = \underset{\omega_j}{\operatorname{argmax}} \prod_{i=1}^{n} p(x_i|\omega_j)P(\omega_j) \tag{2.21}$$

The probability $\prod_{i=1}^{n} p(x_i|\omega_j)$ can be estimated by assuming that that each attribute has a normal distribution. In this way, the distribution of an attribute is estimated by measuring its mean value and its standard deviation from the data [62]. Prior probabilities $P(\omega_j)$ are estimated by counting the number of instances of the classes, and by dividing them by the number of instances in the training set.

Multilayer perceptron. The multilayer perceptron (MLP) is a technique that uses a discriminant function to classify data. The functional unit of a MLP is the *node*. Basically, a node is similar to the perceptron, an algorithm that was briefly introduced earlier in this section. It consists of a set of input connections, a set of output connections and an activation function. A MLP is organised in layers of nodes: the *input layer*, one or more *hidden layers* and the *output layer*. Each node of the input layer provides a signal to the nodes of the first hidden layer. A node of the i^{th} hidden layer computes the sum of its inputs, which is applied to the activation function in order to generate an output signal. This is distributed to each node of the subsequent layer, after multiplying it by a weight w_{ij}, where j indicates the j^{th} node of that layer. The signals propagate through the network and reach the output layer. Each of its nodes generates a discrete output by applying a step function.

The weights of a network determine the function that is computed by that network. The *back-propagation algorithm* is a well-known method for calculating the weights of a MLP [54, 97]. The weights are initialised with random values. Training instances are applied sequentially to the (input nodes of the) network in order to generate a prediction. Then, the *delta values* of the nodes are calculated starting from the last hidden layer. The delta value of a node is the sum of the differences between the expected values of the output nodes and their predictions (the errors), multiplied by their respective weights. The delta value of each node of the second to last layer is computed in a similar way. However, the delta values (of the nodes of the last hidden layer) are used, rather than their errors. This step is repeated until the delta values of all the hidden nodes have been determined. Then, the weights of the network are calculated. Starting from the first layer, the weight

of an input connection of a node is multiplied by the input signal, the gradient of the activation function, the delta value of that node and a parameter, called *learning rate*, in order to recompute its new weight. This process is repeated for all the training instances and for a number of *epochs*. In particular, the learning rate controls the speed of convergence and high values may obstacle the convergence to a solution. An additional parameter called *momentum* can be used to control the rate of learning of a network across the training phase in order to increase speed of convergence and performance of the model [54].

C4.5. C4.5 is a very well-known algorithm for building decision trees [54, 75]. The algorithm is divided into two phases. Firstly, a tree is created, then it is pruned to avoid overfitting. The information gain is a measure of the capability of an attribute to distinguish among different classes. Starting from the top of the tree, the attribute with the highest information gain is associated with a node. The information gain of an attribute is computed from the entropy of the dataset and that of the dataset considering only that attribute. In order to reduce the risk of overfitting, C4.5 prunes branches of the tree after this has been generated [52].

2.2.3 Ensemble classification

In many hard problems, such as the diagnosis of a rare disease, it is common to consult multiple experts, potentially with different backgrounds, before making an important decision. Taking into account multiple opinions can help having a more thorough understanding of the problem, which makes clearer what the best solution to take is. The same principle has been applied to classification problems, and between the end of the 80's and the beginning of the 90's a series seminal works started investigating the possibility to combine more classifiers [15, 83]. The term used to identify these systems is "ensemble classifiers" and they consist of a group of classifier and a technique to fuse their decisions. The classification technique that is adopted is known as the "base learner" and it can be any classification technique, in some cases more than a base learner can be used.

There are different reasons to combine classifiers. The first reason is statistical. In fact, consulting a single classifier could be hazardous, as it might overfit the training set. However, if multiple classifiers are trained on, for example, subsets of the training dataset (bootstrapping), the probability of the classifiers of misclas-

sifying the same instances decreases. As a consequence, the generalisation of the ensemble increases with respect to using a single classifier. The techniques that take most advantage from ensemble learning are *unstable* classifiers, such as neural networks and decision trees [72]. A classification technique is said to be unstable if small differences between training datasets cause large variances in the models of the classifiers trained from those datasets [27]. Moreover, ensemble classifiers are particularly useful when the training set does not reflect accurately the underlying data distributions [72].

Another key characteristic of ensemble classifiers is the capability to handle very large datasets. When the amount of data collected is of the orders of gigabytes or terabytes it could impractical if not unfeasible to train a single classifier. A solution consists of dividing the data in smaller batches, each of which will train a classifier, and then fuse the decisions of the classifiers. Moreover, in some cases the whole dataset might not even be immediately available and therefore data should be learned incrementally. Also in this case, ensembles might come into hand. As soon as each batch of data is available, a new classifier might be trained and added to the ensemble.

Given a technique, the complexity of certain data distributions might be intractable by a single classifier. Figure 6 depicts a dataset containing two different classes (blue and red) for which no single linear classifier that perfectly separates the instances can be found. However, the problem can be overcome by combining three linear classifiers (grey lines) into an ensemble. The classifiers divided the input space into the regions $A_1, A_2, \ldots, A_6$, the use of majority voting ensures that in each of these regions the decisions of the ensemble match the true classes of the instances. The decision boundary of the ensemble is highlighted in Figure 6 by using a black bold line, it can be seen that the boundary perfectly separates the two classes.

It is very common to use heterogeneous data sources in data fusion. While data diversity increases the information that is extracted from the observed system, it can also increase its complexity. As a consequence, it could happen that there is no classification technique that is able to process all the different data formats together [72]. This problem is often solved by using ensemble classification, in which, for example, each classifiers is applied to a data source. As an example, let us consider again the medical diagnosis scenario, data sources might be blood tests, magnetic resonance imaging (MRI), electrocardiography (ECG) [72]. The diversity of these

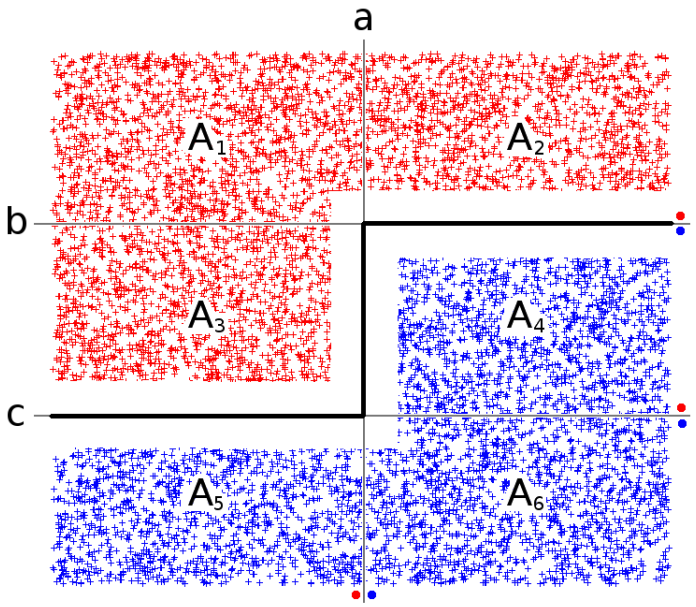


Figure 6: Example of an ensemble classifier. Since the three base classifiers are linear, none of them is able to classify correctly the blue and the red instances. However their combination through majority voting (bold line) allows to discriminate with the highest accuracy the instances of the blue class from the instances of the red class.

types of data makes the problem intractable by a single classifier. However, this problem can be dealt with by using an ensemble classifier, and training each classifier on a single data type or on combinations of a subset of the types.

In order to provide examples of ensemble classifiers, a description of two of the well-known classification techniques is presented. These are: *bagging* and *boosting*.

Bagging. The bootstrap aggregating algorithm, commonly known as bagging, is one of the simplest ensemble classifiers. Given a dataset D_{train}, the algorithm creates a set of M bootstrap training sets from the original training set [15]. Each bootstrap training set is created by randomly sampling N instances from the original training set. The ensemble contains classifiers that are trained on different bootstrap training sets. Random sampling increases the diversity of the classifiers of the ensemble and, as a consequence, the correlation between their errors decreases. In order to classify unseen instances, the decisions of the single classifiers are fused by means of majority voting. The bagging ensemble method suits particularly well to unstable classifiers and it is not very sensitive to noisy data [15].

Boosting. This technique represents one of the first ensemble learning techniques [83]. It was presented as a proof that a *weak* classifier, that is a classifier that performs slightly better than random guessing, can perform as well as a *strong* classifier, or a classifier that has low error on unseen instances. This result is achieved by combining three weak classifiers in a ensemble in [83]. The version of boosting we will consider is Adaboost, which allows to use an arbitrary size for the ensemble. AdaBoost works under the assumption that every classifier that is added to the ensemble should reduce the classification error generated by existing classifiers. Different from bagging, for which data instances have the same probability of being sampled, every instance is associated with a weight which controls that probability. Initially all the samples have the same chance of being selected. At every iteration, the weights are recomputed based on the errors of the ensemble. In this way, the instances that are misclassified have higher probability of being selected in the next iterations.

2.2.4 Semi-supervised classification

For certain applications, providing labels for all the instances is not possible. Semi-supervised classification techniques create a model from labelled data and (generally larger amounts of) unlabelled data [104].

One of the earliest ideas in the field of semi-supervised classification is *self-learning*, also known as *self-training* or *self-teaching*. Initially, a classifier is trained only on labelled data. Then, an unlabelled instance is selected and a label is associated with it by means of the model previously established. The instance and its label become part of the training data, and a new classifier is trained on that data. This process is repeated until the pool of unlabelled data is empty. Self-training is a wrapper algorithm and, therefore, it can be used with different classification techniques. Its performance depends on the classification algorithm and the data being employed [104].

The generative approach is an alternative to the discriminative approach to classification described in Section 2.2.2. *Generative models* firstly use training data to estimate the distributions of the classes, then Bayesian theory is applied to classify new instances [69]. In the context of semi-supervised learning, an estimation of the class-conditional distributions $p(x|y)$ is generated from the class-unconditional distribution $p(x)$ of labelled and unlabelled data by using the available labels [19].

This approach assumes that the type of distribution (e.g., Gaussian or a mixture of Gaussians) of the data is known, and its parameters are fitted using a technique such as the Expectation Maximisation (EM) algorithm [23]. The choice of a model that suits the data is key to the classification performance. In fact, while choosing an appropriate model can improve the performance with respect to the case in which only labelled data is used, choosing an unsuitable model could potentially worsen the performance with respect to the same case [104].

For certain problems, it could be reasonable to expect the instances of a class to have similar characteristics. Since the goal of clustering is to identify groups of instances with "similarities", for those problems, clustering could be used to perform semi-supervised classification. As a first step, labelled and unlabelled instances are grouped into a set of clusters. Then, each cluster is assigned the majority label of its instances. When an unlabelled instance is presented, it is classified with the label of the cluster to which it is more likely to belong.

Co-training can be used in those cases in which the features of a dataset can be divided into two separate subsets of features, and the performances of classifiers trained on those separate datasets are high [104]. Initially, two classifiers are trained on the subsets of features using labelled data. Then, each classifier starts classifying the unlabelled instances. Those instances that are classified with high confidence are made available to the other classifier for retraining. This is repeated until all the unlabelled instances have been processed.

Some semi-supervised methods are specific to a classification technique. For instance, transductive support vector machines (T-SVM), illustrated in Section 2.2.2.2, use the properties of this technique to assign labels to instances. In this case, the term "transductive" maybe misleading. In fact, the purpose of transductive learners is to assign labels only to the unlabelled instances of the training set. However, T-SVM are an inductive learner, as their objective is to classify unseen data.

The transductive, discriminative *graph-based* methods represent a training dataset by means of a graph. Each node is related to an instance, and the weight of an edge represents the "similarity" between the instances associated with the nodes it connects [104]. Several different methods related to graph-based semi-supervised learning have been proposed. For a binary classification problem, the algorithm presented in [12] considers points with label " + 1" as "sources" and points with

label " -1" as "sinks". Then, it finds the minimum set of cuts of edges that isolate sources from sinks. The labelled instances assign their classes to the unlabelled instances with which they are connected.

2.3 Concept drift

When the problem of classification was described in Section 2.2.2, it was assumed that the distribution from which training and testing data are generated is the same and, in particular, such distribution does not change. However, this is not always the case. In fact, there are many scenarios in which the observed system changes and this may affect the distribution of data collected from it. This problem is known as *concept drift* [84, 91]. The problem of concept drift potentially occurs in every application which involves data streams. Unless the characteristics of the system being examined are very constrained, it is reasonable to expect that at some point factors will intervene to modify the distribution of the data.

Consider the medical diagnosis example presented in the introduction. Concept drift may be caused, for instance, by the introduction of an innovative analysis technique that replaces an old one, by wear and tear of devices (instrument for analysis such as ECG), by new variants of some diseases (such as new flu strains) or even new diseases. Any of these circumstances would require retraining of the model of the classifier.

Regardless of a particular type of change, as seen in 2.2.1 (Equation 2.3), a classification problem is described by prior probabilities and class-conditional probability distributions of the data that is collected from it. When data is affected by concept drift, prior probabilities and class-conditional probabilites are also function of time, so they can be re-expressed as $P(\omega_j, t)$ $p(x|\omega_j, t)$ [66].

2.3.1 Categories

Although concept drift regards data with very diverse characteristics, it is possible to identify four major types of drift: *abrupt*, *gradual*, *incremental* and *recurring* [106]. These categories are not necessarily distinct as there can be concept drifts that present combined characteristics.

Drift is *abrupt* when change occurs in a sudden way, or alternatively, it is possible to identify a precise time in which the distribution changes. Suppose that

a supermarket collects data about the preferences of its costumers, for example via online shopping. Let us also suppose that a user buys the same products over a period of time and therefore his basket does not change much over the weeks. At some point, the supermarket advertises the launch of a new product. The customer decides to try it and, becoming quite satisfied with the new product, the customer stops buying the previously preferred product. Notice that there is a precise time in which the new product replaces the previous one, this causes an abrupt change of the data collected from the supermarket.

There are cases in which concept drift is distributed over a timespan. The literature often uses the same term to refer to conceptually similar but practically different variant of this type of drift [106]. We can distinguish between *gradual* drift and *incremental* drift [106].

Gradual drift occurs when a new concept gradually takes over another concept. In order to provide an example of gradual drift, let us reconsider the supermarket scenario presented before. Suppose that the customer tries the new product, but at first he considers the price too high. Initially, he might buy a small amount of the new product along with some stocks of the similar product normally preferred. After a few weeks, the costumer decides that the new product is definitely worth its price and from that time on that will become his preference. In a different scenario, for example the malfunctioning of a cash machine, let us suppose that the machine log files only contain "normal-state" codes. At some point, one component of the cash machine starts malfunctioning. Initially, this generates a few errors, however as the problem worsens, the number of errors increases. After some time, only errors are raised and the functioning of the machine is compromised. For both examples three phases can be distinguished. Initially, only a concept is observed, whether it is a product or a machine state. When drift starts a new concept is introduced, which could be a new product or a faulty machine state depending on the scenario (in this phase both concepts coexist). However, the newly introduced concept will gradually become predominant, so that in the third phase it will be the only concept to be observed.

There is another way in which concept drift can be gradual. In *incremental drift*, the change from a distribution to another is smooth, in the sense that the change is continuous. Suppose that the cash machine of the previous example contains sensors which monitor its functioning. Sensors might be used to measure

physical quantities, such as, temperature, light and vibration. Since cash machines are deployed in changing and diverse environments (day and night, seasonal changes, different user interactions), the distribution of the data that is collected from them is expected to change. For example, temperature, which is low at night, grows increamentally as the sun raises. This is different from incremental drift, which accounts for two distinct concepts (old/new product, normal state/error) and on the fact that one concept gradually takes over another concept. In incremental drift, the data distribution changes through a series of intermediate states in which the concept itself changes, but new concepts are not introduced.

Recurring drift is a concept drift in which a previously observed concept is presented again. The ways in which the distribution varies can have different characteristics (abrupt, gradual or incremental).

2.3.2 Datasets

The problem of classification embraces fields with different characteristics: from text mining to network security, from detection of fraudulent transactions to biomedical applications [54, 63, 86]. Such a variety of applications means that the data that is sampled can have very diverse features. For instance, data might embody continuous features (such as real-valued ones), discrete (boolean for example) or nominal features (text documents). Some problems account for data with missing attributes [54]. In some cases the sampling probability can vary depending on the class, in which case we say there is a class *imbalance* [91, 24]. In addition, several types of drift can determine how the distribution changes. They are: abrupt, incremental, gradual and recurring drift as it was shown in Section 2.2.2.

There are several datasets with concept drift that feature real-world data, although their number represents a small fraction of the machine learning repositories [9]. Using synthetic data is the most common choice in the literature, since it leaves to the designer the decision of which characteristics to include in the data [9]. Testing an algorithm on real-world data, however, allows to assess its effectiveness on a practical problem rather than a constrained testbed with "dry" characteristics [29]. Probably the major limitation of this data is that, since information is collected from a real-world system, it is not always possible to determine when concept drift is ongoing, as noted in [4]. This can be a problem if the properties of a drift detection algorithm need to be tested. On the contrary, this problem does not affect synthetic

datasets, for which it is possible to decide when and how frequently drift starts and the types of drift the data has to embody [60].

2.3.2.1 Examples of Datasets

In order to give an idea of concept-drifting data, we present the description of some of the datasets used to test the algorithms in Section 2.4. These datasets contain real-world or synthetic data and they represent the four different types of drift described in Section 2.3: abrupt, gradual, incremental and recurring.

One of the most used testbeds for adaptive algorithms is the learning problem defined in [84], also known as STAGGER dataset. In particular, data instances consist of three nominal features, each of which can assume three distinct values. More precisely, $size \in \{small, medium, large\}$, $colour \in \{red, green, blue\}$ and $shape \in \{square, circular, triangular\}$. The type of drift of this dataset is abrupt and the target concept changes after, respectively, one third and two thirds of the total number of instances have been processed.

A variety of datasets containing real numbers has been developed to test algorithms for concept drift [24, 60, 64, 66, 88]. A common method consists of generating points uniformly in a multidimensional space and then assigning a class to every instance according to a function which changes over time. For instance, the function might be a moving hyperplane, that is a hyperplane whose inclination and position vary. These are determined by the variables $< w_0, \ldots, w_d >$, where the equation of the hyperplane is:

$$\sum_{i=1}^{d} w_i x_i = w_0 \tag{2.22}$$

The class of an instance is decided according to its position with respect to the hyperplane, in this way, intances for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are assigned, for example, a positive class, while instance such that $\sum_{i=1}^{d} w_i x_i < w_0$ would receive negative class. Different types of concept drift are obtained by changing the orientation of the hyperplane, which is performed through the change of its weights $< w_0, \ldots, w_d >$ [60]. Although using an hyperplane is the simplest and probably most diffused option, in a similar manner, other functions can also be used. Among these, there are examples of moving circles and sine functions [60].

Other real-valued datasets use combinations of Gaussian distributions [24, 64, 78]. In particular, the dataset defined in [24] contains "clusters" of instances gen-

erated by Gaussian distributions with different characteristics. Concept drift is obtained by varying the characteristics of the Gaussian distributions. In particular, their mean values and their standard deviations are changed based on different patterns across the length of the dataset. The type of drift is incremental, since positions and the standard deviations of the clusters change gradually according to linear functions. An instance can belong to one of four Gaussian clusters C_1, C_2, C_3 and C_4. C_1 is the only cluster associated with class "$+1$", the remaining clusters are associated with class "-1". The dataset consists of $|D_{online}| = 1,550$ instances. The prior probabilities of the classes generate a high imbalance of the classes. In fact, only 3% of the instances belong to class "$+1''$. The concept drift of this dataset is divided into three parts, as illustrated in Table 1. For the first $\lfloor |D_{online}|/3 \rfloor$ instances, the clusters do not change their positions. In fact their centers, identified by the coordinates μ_x and μ_y, do not change. However, the standard deviations (σ_x and σ_y) of the clusters C_2, C_3, C_4, are modified according to the linear functions defined in the top part of Table 1.

$1 \leq t < \lvert D_{online}\rvert/3$				
	μ_x	μ_y	σ_x	σ_y
C_1	8	5	1	1
C_2	2	5	1	1+6t
C_3	5	8	3-6t	1
C_4	5	2	3-6t	1
$\lvert D_{online}\rvert/3 \leq i < 2\lvert D_{online}\rvert/3$				
	μ_x	μ_y	σ_x	σ_y
C_1	$8 - 9(t - 1/3)$	5	1	1
C_2	2	5	1	3
C_3	5+9(t-1/3)	8	1	1
C_4	5+9(t-1/3)	2	1	1
$2\lvert D_{online}\rvert/3 \leq t \leq \lvert D_{online}\rvert$				
	μ_x	μ_y	σ_x	σ_y
C_1	$5 - 9(t - 2/3)$	5+9(t-2/3)	1	1
C_2	5-9(t-2/3)	2	1	3-6(t-2/3)
C_3	8	8	1	1
C_4	8	2	1	1

Table 1: Concept drift of the dataset presented in [24]. The positions (μ_x, μ_y) and the standard deviations (σ_x, σ_y) of four Gaussian distributions ($C1, C2, C3$ and $C4$) data vary according to three patterns, described by the linear equations of, respectively, the top, the middle and the bottom sections of the table, in a temporal sequence.

The drift of the intermediate portion of the dataset, defined in the central part of the same table, causes the centers of C_1, C_3 and C_4 to move along the x-axis. In the third section of the concept drift (bottom of Table 1), the positions of the clusters C_1 and C_2 change, as well as the shape of C_2, while the characteristics of C_3 and C_4 are not affected. Figure 7 gives a graphic representation of this distribution.

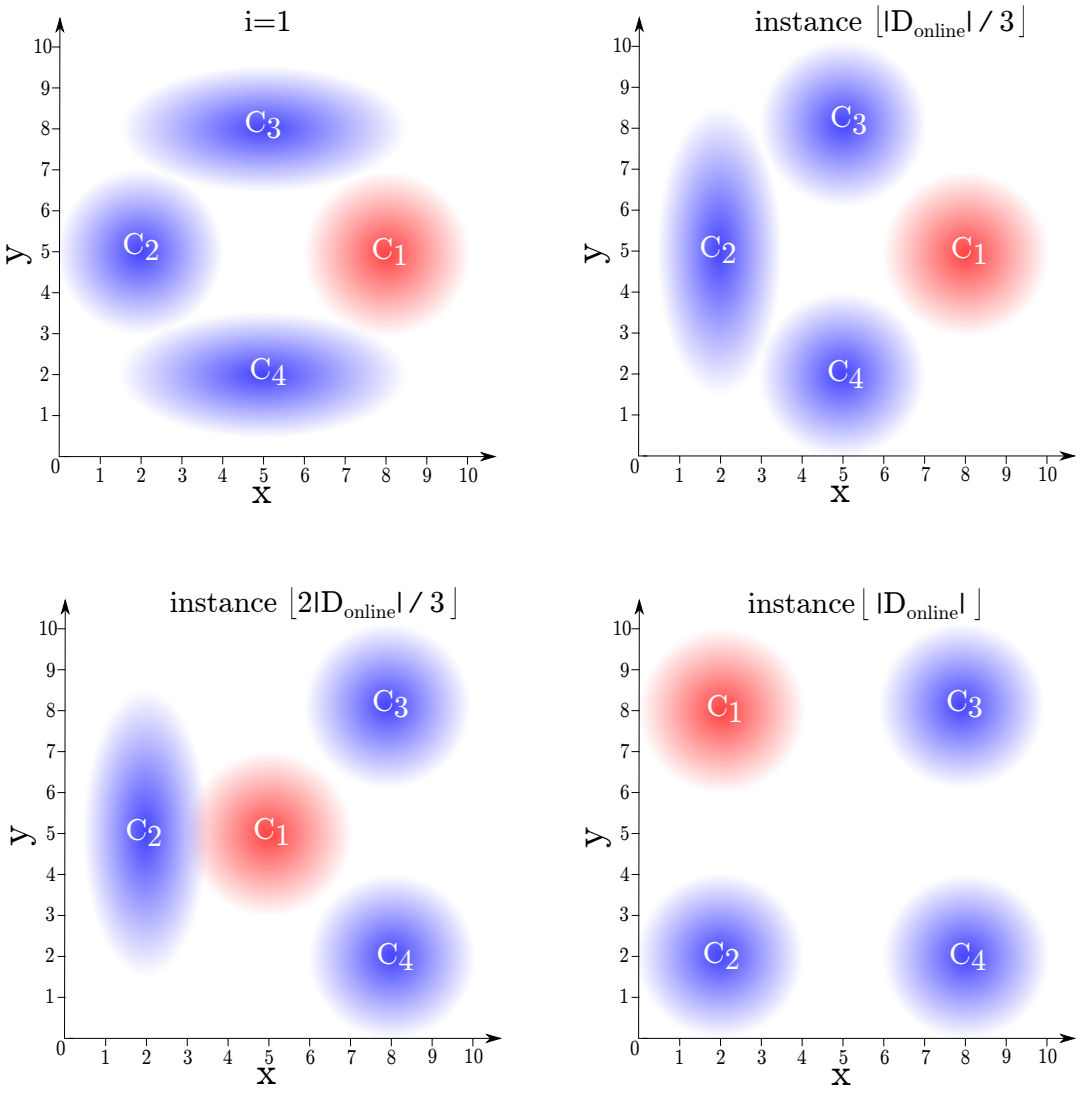


Figure 7: Visual representation of the dataset proposed in [24].

One of the problems of machine learning is the classification of items such as documents, scientific articles, books, commercial products or emails according to the interest of a user [96]. This task presents a series of difficulties. For instance, a

common way to represent a document consists of counting the number of occurences of the words that are contained in it or, more simply, recording the presence of a specific word. In this way, each word is regarded as a feature, and considering the number of words that can appear in vocabulary (in fact, all the documents have to refer to the a common vocabulary in order to be classified), the number of features is of the order of several tens of thousands. Moreover, since new words can be added to the vocabulary, the size of the feature set is not fixed as in many other classification problems.

As an example of a real-world dataset, we describe the Electricity Market dataset [31]. It contains data collected from the Australian New South Wales electricity market with the aim to predict its future fluctuations. With that purpose, several attributes are recorded, such as, day of week, time stamp, electricity demand. The purpose of training a classifier with this data is to use past observations to predict the price of electricity. Other real-world datasets contain data of internet traffic, poker tournament hands, credit card fraud detections [4, 9].

2.4 Classification of data with concept drift

After providing several examples of datasets, we present a selection of classification algorithms designed to handle data with concept drift. The descriptions will take into account the different amounts of supervision that these algorithms require in order to train or update their models. In particular, we distinguish among supervised updating, unsupervised updating and semi-supervised updating of the model of a classifier.

2.4.1 Supervised model updating

This section and the following sections are based on [105] and [48]. Many environments are dynamic and so is the distribution of data that is collected from them. This clashes with the assumption that is traditionally made in classification that the data distribution is stationary. The definition of the problem of classification given in Section 2.2.2 needs to be reformulated to take into account concept drift.

Similar to the static case, the problem consists of determining a function h (the model) from a set of observations $\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x}_t, y_t)\}$ to classify unseen instances $\mathbf{x}_{t+1}$. However, because of changes in the data distribution, the classification per-

formance of a model may degrade if it is not updated. After classification, the true class of an instance becomes available. This information is used to update the model. In this way, data is provided in the form of a stream of instances, in which the unlabelled instances that are used to test the algorithm serve also as training data when their true classes are provided. Differently from the static case, in which all the training data is generally used to build the model, the presence of concept drift entails that different data instances might be representative of different snapshots of the changing distribution. In this scenario, a classifier should use the data that is representative of the current distribution, in order to classify unseen instances.

2.4.1.1 Assumptions about the future distribution

One of the problems of using data with concept drift is that the way the distribution changes is assumed to be unpredictable. In order to simplify the classification system, a vast majority of approaches assume that the future distribution is the same as the current [106]. This choice is reasonable if the amount of change is small, changes are not frequent and the consequences of a misclassication are not critical. However if concept drift is severe and frequent, such a strategy would provide poor performance.

Some techniques, however, try to predict rather than simply classify. By analysing past history, they extract patterns of shifts among concepts, if these patterns occur again in the future then such techniques attempt to predict the future concept [102]. This, however, requires that the type of concept drift is recurring.

2.4.1.2 Adaptivity

When dealing with concept drift adaptivity is key. For instance, a classifier should be able to react quickly to change, and if it fails at doing so the consequences would be catastrophic for some applications. Adaptivity is not measured only in terms of speed, adaptivity is also about extracting as much information as possible from the data and using it in the best way to reduce misclassification. That is, a model should show *plasticity*, which is the ability to embody new information, and *stability*, that is the ability to retain relevant information and discard information that is no longer needed [35, 64].

There are several ways to achieve adaptivity, for example, by using adaptive

classification techniques. Apart from being able to learn data incrementally, they also implement mechanisms to forget outdated information, which allow them to cope with concept drift. These techniques perform adaptation by rearranging the whole model or a portion of it. This is, for example, the case of AISEC that uses the supervision of a user to delete those detectors that fail at classifying correctly [85]. Decision trees can also be adaptive, by pruning those branches that do not reflect the current distribution [39].

Another way to achieve adaptivity concentrates on formation of training data. This entails the selection of a subset of the labelled data that have been observed to date to create a training set, which is then used to create a classifier. Notice that the classification technique need not be adaptive. Training set formation strategies can adopt different criteria to decide whether an instance should stored in memory or it should be discarded. Instances can be consecutive in time, in the case of *windowing*, or nonconsecutive in the case of *instance selection* [106]. Windowing works under the assumption that recent instances are the most representative of the current distribution, which is reasonable if the whole distribution changes. A small window size provides quick adaptation when the distribution changes rapidly. However, if concept drift is rather slow, a larger window is a more sensible choice since the higher amount of data collected affords better generalisation. When the size of a window is fixed, very strong assumptions need to be made over the speed of drift. By contrast, variable-width windows are more flexible, since they avoid the trade-off between generalisation capability and speed of adaptation. The decision whether to stretch or enlarge the window is generally based on the detection of concept drift [7]. The advantages of a variable window come at the cost of an increased complexity, that can be computational and/or related to parametric tuning.

Instance selection, which assumes instances need not be consecutive in time, is effective when concept drift does not regard the entire distribution. As a matter of fact, data generated from the part of a distribution that has not changed are equally important, no matter how old they are. While, instances drawn from the part of the distribution affected by concept drift should be recent.

Recall from Section 2.2.3 that ensemble classifiers exploit the diversity of a set of classifiers by fusing their decisions. Besides improving performance for stationary distributions, ensemble classifiers are also used for concept drift. In this case adaptivity is obtained by modifying the way classifiers are combined, which reflects

in a change of the model, even when batch techniques are used. For instance, if a classifier performs better than others on recent data, then its influence in the decision making is increased by increasing its weight, or perhaps, its decision can be selected to be the decision of the ensemble. Ensemble classifiers fits particularly well to recurring concepts. Classifiers are trained on different concepts, new data is labelled by selecting the classifiers that are specific for the concept of the data.

2.4.1.3 Strategies for concept drift

Algorithms for concept drift can be divided into *trigger-based* and *evolving* [106]. Trigger based approaches rely on a signal that indicates when drift has occurred. This is handled by the classifier that reacts to the change by modifying or retraining the model. In evolving classifiers, by contrast, there is no explicit signal to trigger adaptivity. Evolution is simply driven by the goal to minimise classification errors over incoming instances [106].

A large number of evolving classifiers use ensemble classification. Base classifiers of an ensemble are normally trained over different data batches. There is no fixed rule, but commonly evolution is an iterative process that consists of the following steps: collect data plus labels, train a classifier, add the classifier to the ensemble, re-compute the weights of the classifiers according to their performance, classifiy unlabelled instances, and so on. Another strategy consists of using overlapped windows of different length, which contain respectively the last n examples, the last $2 * n$ examples, and so on. Base classifiers are trained on the windows, and the classifier with the lowest error is selected to classifiy unseen instances. The criteria to assign weights to classifiers or to select the output of a specific classifier try to maximise the performance of the ensemble over future data. For example, weights can be a function of the performance over past iterations, or they can be based on selective cross validation, or again they can be based on performance estimates that are generally specific to a classification technique. Ensemble classifiers are suitable to gradual drift and incremental drift, in fact they adapt gradually by weighting the classifiers of the ensemble [49]. Instance selection is another option, the probability of an instance being selected for training is generally dictated by a weight. Higher weights are assigned to instances that have been misclassified. This approach is similar to boosting. Weights can also be used in a different way, weights can in fact dictate the degree to which instances influence the construction of a model.

Different from instance selection, in *instance weighting* all the training data is used, but they give different contributions to the model construction. In *Feature-space* methods adaptivity is achieved by operating on the set of features. For example, this is common in classification of text documents.

Concept drift detection is the most used method to trigger adaptivity. Detection can be pursued at different levels of a classification system: directly on its features (input level), on the parameters of its model (internal level), or on the errors of the decisions (output level). When concept drift is detected, the model is generally retrained on new data and history about previous distributions is deleted. *Training windows*, another triggering strategy, counts on the comparison of the performance of classifiers trained on windows of different length. If they perform differently, then drift might be occurring and the model should be modified. The ability of triggering techniques to forget old information makes them particularly suitable to abrupt drift.

2.4.1.4 Evolving techniques

The distinctive characteristic of evolving methods is that they do not use explicit information whether concept drift is occurring. They simply evolve over time to minimise some performance measures. Here we present a review of evolving algorithms, this is divided into ensemble methods, instance weighting methods and feature extraction methods, according to [106].

Ensemble-based. Ensemble classifiers are based on the principle that in certain conditions a set of classifiers can outperform a single classifier. Although they were initially developed for static distributions (Section 2.2.3), ensemble methods have also found a successful employment for concept drift [53, 88]. Different from the stationary distribution case, the models of the classifiers of the ensemble are created from data collected at different times or perhaps data that are representative of separate concepts. The reason of such a large diffusion probabily lies in the modularity of ensemble classifiers which brings several advantages such as reduced storage requirements (that follows from the fact that the information of the data is converted into a model) or the possibility to parallelize the decision making, as well as, the updating of the model of the ensemble if an incremental or an adaptive classification technique is used.

There are two major ways ensembles can be used to deal handle concept drift. In the first type, new classifiers are periodically trained by a data stream of labelled data. In this way this strategy maintains "memory" about past data observed in the form of models, by combining these models by means of weights it is possible to obtain more detailed information about the current distribution and, assuming this does not differ considerably from the distribution to observe, this information can be used to predict the class of new instances. For this reason, this ensemble strategy is better suited to gradual or incremental concept drift. The second strategy consists of mantaining an ensemble of classifiers, where each classifier is specific to a particular concept and can be selected when the corresponding concept is observed [7]. For this reason, the second type is better suited to recurring concept drift as it presumes that the same concept will occur repeatedly over time. The techniques that are considered in this section belong to the first type.

A large variety of algorithms have been presented, they differ in the way data is sampled, in the way classifier performance is evaluated, in the number and type of classification techniques that are used or that can be used. Two ensemble-based classification algorithms are used: one with fixed size and one with variable size of the ensemble.

One of the earliest examples of ensemble methods is the Streaming Ensemble Algorithm (SEA) [88]. Initially, the ensemble classifier of SEA is empty. When enough training data is available a new classifier is created and added to the ensemble until this reaches its maximum size. When the ensemble is full, the classifier created at iteration $i-1$ is added to the ensemble only if it outperforms an existing classifier, which would then be deleted. The performances of the classifiers are assessed using the dataset of iteration i. Experiments were performed to test different decision fusion techniques such as a weighted majority voting based on the accuracy of the classifiers or a gating mechanism to exclude from the voting the classifiers that are expected to perform badly. However, these techniques do not provide a consistent improvement over the simpler majority voting. Although the SEA algorithm is a generic algorithm in the sense that it is not designed for a specific classification technique, the experiment in [88] only covers the decision tree technique $C4.5$ [74].

Another example of an evolving ensemble method is the algorithm dynamic weighted majority (DWM) [53]. It consists of an ensemble classifier of variable size which makes use of weighted majority voting. The algorithm copes with concept

drift by reacting every time an error occurs. In this way, the higher the number errors, the more frequently the model is modified. For example, if a classifiers fails at predicting the class of an instance its weight is reduced or if it is the ensemble that misclassifies an instance then a new member will be added to it. DMW starts by creating an ensemble containing a single classifier, with an initial weight of 1. When an instance is presented, if it is misclassified then the classifier's weight is reduced by a factor $\beta < 1$. Since initially the decision of the only classifier is also the decision of the ensemble, a new classifier is added and its weight is set to 1. To prevent the size of the ensemble from growing, classifiers whose weight is under a threshold θ are removed. However, this does not affect recent classifiers. In order not to penalise older classifiers, at each iteration, all the weights are normalised so that the maximum weight is 1. This algorithm represents an intuitive example of evolving methods for concept drift. For instance, the contribution of a classifier into the ensemble and the decision whether to remove it are determined by the same variable, that is its weight.

Ensemble classifiers exploit the combination of diverse classifiers. Altough there many versions have been proposed, a general strategy consists of using weights to modulate the contribute of each classifier. New classifiers are generated using the streaming data and uneffective classifiers are normally deleted. The use of multiple classifiers brings several advantages over single classifiers. They afford better generalisation and tolerance to noise, while the availability of different models helps mitigate the influence of models with poor performance. Moreover, ensembles are easy to parallelise to reduce the interrogation time. Parallelisation can also be applied to the training if adaptive techniques are used. Moreover, if the characteristics of drift change over time, by using ensemble methods it is possible to include techniques that are effective for those types of drift. All these characteristics make ensemble classifiers particularly suitable for gradual types of drift such as incremental ones, However they can also be used to deal with recurrent drift [7].

Instance weighting and instance selection. The majority of approaches that deal with concept drift handle instances in the same way when these are processed. In fact, it is common to assume that the training data, in a window for instance, have all the same importance. However, this is not necessarily the case, and considering factors, such as age or region of the input space of the instances, might help increase

the performance of classification. Nonetheless, two categories of algorithms, *instance weighting* and *instance selection*, exploit different strategies.

Similar to the way weights are used to control the influence of the classifiers in an ensemble, weights can also be associated with training instances. In this way, it is possible to regulate the contribution of each instance to the construction of a model. The data instances to be weighted can appear in a window (this entails that older instances are discarded) or all the instances that are observed can be used. One of the limitations of data weighting lies in the fact that it requires the classification technique being used to be able to process weights along with data instances. This restricts its applicability only to a small number of techniques such as SVM and decision trees, for which ad-hoc versions have been developed. While in instance weighting all the training instances are actually maintained and their contribution to the construction of a model is regulated by their weights, in instance selection only a subset of instances is actually sampled from the original dataset. This entails that instance selection does not require ad-hoc techniques to process the data. In a similar way to instance weighting, instances can be drawn from a window of a certain size or alternatively any of the instances observed so far can be selected. Although, in some cases instance selection makes use of weights (for example in boosting-like techniques), these are not directly used by the learning algorithm to build the model. They only serve to select a subset of instances from the original set.

Weighting is often used to give different importance to instances depending on their age. This is based on the assumption that data becomes gradually less representative of the recent distribution as it ages. For this reason, higher importance should be given to recent instances, for example by assigning them higher weights than older instances. For example, [55] utilises a linear function to weigh the instances of a window in order to avoid the abrupt forgetting of training windows. In this way the contribution of an instance decreases linearly with its age. This requires that instances are forgotten after some time, but the forgetting is gradual.

The principle of weighting instances according to their age is also used in [48]. In this case, however, the weighting process regards all the instances that have been observed, not only the most recent ones as in [55]. The function that is used to weigh the classifiers is nonlinear. It is in fact the exponential function $w(x) = e^{-\lambda t_x}$, where t_x is the time in which x was observed. By tuning the parameter λ it is possible

to control the rate at which the algorithm forgets data. When $\lambda = 0$ the same importance is given to all instances regardless of their age, while for positive values of λ, the higher its value the lower the weight an old instance is given. For all those cases in which the characteristics of drift change over time, using a fixed value for λ might limit classification performance. Allowing λ to vary would be much more effective to contrast drift.

Boosting is a technique of ensemble construction, which consists of adding classifiers to an ensemble in order to correct the errors generated by the existing classifiers. A description of boosting was given in Section 2.2.3. The ideas of boosting ensemble construction have also been applied to problems with different data specifications. For example, [70] presents a version of AdaBoost capable of incremental learning when used in combination with online techniques. In this algorithm, the probability of a training example being selected to be part of the training set of a new classifier is dictated by the Poisson distribution. This mechanism guarantees diversity within the ensemble. More precisely, assuming that the ensemble has fixed size, when a new instance is presented to each classifier, the probability that a classifier will be updated using the instance is ruled by the Poisson distribution. This algorithm, however, is only capable of incremental learning, since it is not specifically designed to cope with concept drift. Therefore it might not be able to react effectively to a changing distribution. A modified version of [70] to cope with the problem of concept drift is proposed in [71], which is given the name Online Nonstationary Boosting (ONSBoost). The baseline of the algorithms is the online boosting algorithm [70]. The major addition consists of the replacement of the classifiers that do not contribute positively to the performance of the ensemble. This happens when the perfomance of the ensemble increases if a classifier is not considered by the decision making. This mechanism allows the algorithm to drop uneffective classifiers and therefore update its model. Similar to other ensemble methods, since "young" classifiers normally show poorer performance compared to older classifiers, initially they are given immunity from deletion in order to mature. More precisely, a classifier must have been trained with at least a minimum number of instances before it can be deleted. This number is dictated by a parameter of the algorithm. The algorithm ONSBoost performs slightly better than simple online boosting [70] (the version boosting for incremental learning) on different data sets, including STAGGER.

Feature space. In addition to using past data or combining existing models, it is also possible to tackle concept drift by operating on the set of features. This is the case of text mining applications, of which classification of streams of documents is one of the biggest problems.

An example of an algorithm that deals with classification of a stream of documents is described in [44]. The algorihtm possesses a vocabulary of words that, when a new labelled document is presented, is checked to verify if the words in the document had been observed before. If this is not the case, those words are added to the vocabulary. Then, the algorithm updates the statistics of each word, by modifying two counters that, for each word, keep track of the number of documents which contain the word and the number of documents which do not. Separate counters are kept for each class. These statistics are used to evaluate the relevance of each word. This information is used to associate a class with a new document. More precisely, only the top N features are used to classify incoming documents. This feature selection method can be used only with classification techniques that can handle dynamic spaces of features, such as k-NN and naïve Bayes. However, [44] suggests that using k-NN should be avoided, since it cannot deal effectively with the large amounts of training data that are normally observed for this problem. Another examples of feature space classification for concept drift is described in [29].

Density-adaptive forgetting. An alternative approach to maintaining instances in memory is based on the idea that these should be forgotten at a rate proportional to the rate of arrival of new instances in their locality [79]. In this way, if a region receives new instances, older instances of that region should be discarded. On the contrary, if no instances are added to a region of the input space, then existing instances should be maintained. This is different from using fixed-size or adaptive-size windows, as, in that case, the entire set of instances is affected by the forgetting mechanisms.

Density-based forgetting in [79] is implemented through a mechanism of weighting of instances that considers the number of neighbouring instances, as well as their proximity. When a new instance $\mathbf{x}_{new}$ is provided, it is assigned a unitary weight. After that, the weights of the k nearest instances $\mathbf{x}_{\{1\}} \dots \mathbf{x}_{\{k\}}$, are decreased by a factor γ that depends on the parameters τ and m, according to the following

espression:

$$\gamma(\mathbf{x}_{new}, \mathbf{x}_{\{i\}}) = \tau + (1-\tau)\frac{d_(\mathbf{x}_{new}, \mathbf{x}_{\{i\}})^2}{d_(\mathbf{x}_{new}, \mathbf{x}_{\{m\}})^2}, \quad i = 1, \dots k \tag{2.23}$$

where $\mathbf{x}_m$ is the m^{th} nearest instance, with $m \geq k$. The weights of the other instances are not affected. An instance is deleted when its weight decreases under a threshold θ. The notion of density is also used for learning [79]. In fact, density-adaptive learning uses trees to generate partitions of an input space with different resolutions. In particular, this algorithm is applied to the problem of reinforcement learning [79].

2.4.1.5 Triggering methods

The particularity of triggering methods is that changes in the model are determined by a signal indicating the presence of concept. Triggering method are generally used to deal with abrupt concept drift. If drift is detected the actual model is discarded and a new one is built from scratch. We can distinguish between change detection and training windows.

Change detection. One of the most used triggering approaches entails the use of concept drift detection mechanisms. Detection can be based on the different levels of a classification system, that is: inputs, internal parameters of the classifiers or their outputs, as highlighted in [50]. Concept drift can be detected by looking at raw input data, if for example the estimated distribution of a class changes. Concept drift detection at the intermediate level is based on anomalous change of internal parameters of the model, such as complexity of the set of rules or the decision tree. Drift detection that is based on the output, generally monitors change in performance indicators such as accuracy. It should be noticed that, while drift detection mechanisms based on input data or output performance are general and can, therefore, be applied to any classification technique, drift detection at the intermediate level is tied to the particular technique being used.

Let us consider a static classifier that processes a stream of instances generated by a static distribution. Under these conditions, the probability that n misclassifications occurred after i instances have been processed, is dictated by the binomial distribution. The probability of misclassifying an instance $p(error)$, a central parameter of the binomial distribution, is very unlikely to be known in

practical applications. However, an estimate is provided by the error rate ρ_i at instance i. The higher the number of instances that have been observed, the lower is the standard deviation σ of the error rate, which is given by the formula $\sigma_i = \sqrt{\rho_i(1-\rho_i)/i}$ [31]. This means that, for static distributions and static classifiers, the error rate tends to converge to a certain value. A large variation of the error rate would be suspicious, since it might conceal a change in the distribution. The detection method presented in [31] is based on this assumption. In order to monitor variations from the expected error rate, the algorithm uses the variables ρ_{min} and σ_{min} to keep memory of the lowest error rate observed so far and its variance. In fact, the error rate is expected to diminish as the classifier learns new instances. The detection mechanism works as follows: suppose that when the instance $i = k_w$ is presented, then if $\rho_i + \sigma_i \geq \rho_{min} + 2\sigma_{min}$ then a warning is raised. If after a few iterations, let us say when the instance $i = k_d$ is presented, it happens that $\rho_i + \sigma_i \geq \rho_{min} + 3\sigma_{min}$ then the window is reset and a new window collecting the most recent instances starting from k_w is used to re-train the classifier.

As noted by [8], one of the problems of the method described in [31] is that there might be a significant delay before drift is detected. For this reason, the paper introduces ADWIN (ADaptive WINdowing), an algorithm that automatically determines the length of the training windows by exploting the concept of Hoeffding bounds [38]. The authors argue that the ADWIN increases considerably the reactivity of drift detection. When a new data instance arrives, it is automatically added to the window, then the length of the window W is recalculated. In order to do so, the algorithm considers every possible couple of separate subwindows of W, and it calculates the mean values of each subwindow. If the difference between the two values obtained does not satisfy the Hoeffding bounds, that is, the difference is too large according to the number of instances in the windows, then the oldest element of the window is dropped. This process is repeated until a suitable window size is obtained. Given a window, calculating the average of every possible couple of nonoverlapped subwindows is computationally expensive. For this reason [8] proposes ADWIN2, an evolution of ADWIN, which reduces the cost to optimise the window size. An interesting point of ADWIN is the fact that, differently from other algorithms, rigorous guarantees of performance are provided.

Training windows. In training windows methods, adaptivity is triggered by the comparison of different windows with different characteristics. An example of triggering method based on training windows is presented in [4]. The algorithm introduces a fairly simple but effective solution to detect concept drift. This relies on two online classifiers, a "reactive" classifier is trained on a window of fixed size containing only recent data, while a "stable" classifier is trained incrementally on a larger window of variable size. The algorithm monitors the performance of the classifiers over the most recent instances by means of a circular list of binary values C. If the class assigned to a novel instance by the reactive classifiers is correct, but the classification of the stable classifier is not, then the current entry of the list is set to 1, otherwise it is set to 0. Then the pointer is moved to the next position. If the number of 1s in C is higher than a threshold θ this is interpreted as a change in the data distribution. This triggers the copy of the model of the reactive classifier into the stable one, thus discarding the old model and the setting of all the bits of C to 0. Although it contains multiple classifiers, the system cannot be considered an ensemble since only the stable classifier is actually used to predict the class of unseen instances, while the reactive learner is merely used to detect drift.

2.4.1.6 Other approaches

Adaptive techniques are designed to modify their models based on the supervision they receive. As a consequence, they do no need meta-algorithms such as ensembles or concept drift detection mechanisms.

Decision trees represent a very intuitive classification technique that can be used for real-valued data and nominal data. Conventional decision trees require the entire training dataset in order to be built. That is a problem when the entire dataset is not immediately available, for example, if it is presented under the form of a stream. In order to address the problem, [25] proposes Very Fast Decision Tree (VFDT), an algorithm capable of incremental learning. The tree, initially composed of a single leaf, processes labelled data instances one by one. The label that is associated with a leaf is the one with the highest number of occurrences. If a leaf processes instances of different types this means that the leaf need to be converted into an internal node. The Hoeffding bounds [38] are used to identify the attribute to be associated with the newly generated node. In order to cope with the drifting data a modified version of the algorithm was proposed in [39], it was given the name Concept-

Adapting VFDT (CVFDT). The algorithm CVFDT keeps its model consistent with a sliding window. If, because of a change in the distribution, an internal node precedently established stops satisfying the Hoeffding test, new subtrees are grown and eventually one of them will replace the subtree starting from the selected node. The algorithm CVFDT also embodies a mechanism to vary the size of the window and check the consistence of the tree if drift is suspected. A change in the window size can be triggered, for example, by a sudden increment in the rate at which data are provided or an increment in the number of nodes having their associated attribute reconsidered. According to [20], the CVFDT algorithm requires large amounts of data in order to update the tree, as well as, large training time.

Another adaptive classification technique is AISEC. In particular, AISEC contains procedures to update its set of detectors as soon as supervision is provided. In fact, the earliest implementation of that algorithm was designed to deal with the problem of classifying emails according to the interest of a user. That is a problem that require frequent updates of a model.

As discussed in Section 2.4.1.1, there are approaches to dealing with concept drift, evolving and trigger-based. The algorithms presented in [102] propose an alternative approach. Rather than simply reacting by retraining, or using a history of previous observations, the algorithm attempts to predict what concept is likely to be observed next. This approach, of course, suits particularly well scenarios in which concepts are likely to reoccur. The algorithms maintains a history of the concepts that have been observed, which is used to build a prediction matrix. Upon the detection of concept drift, the prediction matrix is queried to find the concept that has the highest probability of being observed next, so that to chose a model. If, according to the history, two or more concepts have similar probabilities to occur, the algorithm searches a model among the available ones with satisfactory classification performance over the new data. If no such model can be found, this could mean that the current concept has not been observed before, therefore a new classifier is trained. The detection method uses a window with variable but limited size and a threshold over the number errors. If the window has reached its maximum size and the error has trespassed the threshold, the first instance of the window is identified as the point in which drift started. Otherwise, the beginning of the window is shifted to the first misclassified instance. The algorithm also exploits a method to evaluate the correlation of two models. When concept drift is detected and a new classifier

is being trained on the new distribution, this mechanism compares the new model with the past ones. If the correlation between the currently trained model and an existing model is above a threshold then the concept is considered as recurring and the old model is used for classification of following instances.

2.4.2 Semi-supervised model updating

One of the problems that afflicts classification of drifting data is that for large data streams it is not always possible to label all the instances after these have been classified, especially when the labelling is done manually. For this reason, recent research on concept drift has seen a growing interest in semi-supervised strategies, which require labels only for a small subset of the instances in order to update the model [47]. To classify new instances, semi-supervised techniques exploit the information of labelled data along with the information carried by unlabelled data.

The algorithm introduced in [59, 103] is based on an ensemble in which every member combines clustering and classification to label unseen instances. Different from the problem we presented in Section 2.4, a batch containing labelled and unlabelled instanced is provided. Labelled instances and part of the unlabelled instances form the training set, while the remaining (unlabelled) data is used for testing. Training data is processed in two stages. Initially, clustering is used to extract statistics about the data, which constitute the model of a nearest-neighbor classifier. A modified version of k-means is applied to the data. In addition to intra-cluster connectivity and inter-clusters distance, the algorithm also maximises the *purity* of a cluster. A cluster is considered pure when the labelled instances it embodies belong to only one class. Information extracted from each class regards, size of the cluster, ratio between between labelled and unlabelled of points, number of points for each class, centroid, and a vector of sums of the values of each dimension of the points. These statistics represent the model of a k-nearest neighbor classifier. More precisely, the nearest neighbor algorithm also considers the percentage of labelled instances in the cluster rather than just distance. The new classifier is added to the ensemble to replace a classifier with low accuracy.

Experiments show that the algorithm performs comparably with supervised techniques, but it requires only 5% of labelled instances. The idea of combining clustering and classification is interesting. However, the way it is implemented does not allow the use of classification techniques other than k-nearest neighbors.

This entails that the algorithm is affected by the typical limitations of k-nearest-neighbor, namely, the curse of dimensionality, time and space complexity, sensitivity to irrelevant attributes.

Assuming that a data stream is divided into contiguous batches, the algorithm in [103] proposes to divide the points in a window of a data stream into four categories: I) labelled points for which the distribution does not change with respect to the previous batch, II) labelled points with a similar distribution to the previous batch, the categories III) and IV) are respectively similar to I and II, with the difference that the class of a point is not provided. A set of solutions for classifying mixtures of points of different categories is proposed. Points from categories I and II, which are labelled and have with slightly different distributions, can be processed using a variant of SVM. Two weights determine the contribution of each category to the construction of the model. Moreover, by using transductive SVM, it is also possible to exploit points of the third category (nondrifting and unlabelled). In order to embody information from unlabelled drifting points into the model, a variant of k-means is combined with the variant of SVM for the categories I, II and III.

Along with the problems of concept drift and partially-labelled data, the algorithm described in [57] also deals with the problem of recurrent concepts. The algorithm features an incremental decision tree. When an instance, labelled or unlabelled, is processed, it is associated with a leaf of the tree. If a sufficiently high number of instances are associated with a leaf, k-means is used to assign a class to the unlabelled ones. The unlabelled instances in each cluster receive the majority class of the cluster. The algorithm features a mechanism to distinguish between noise and concept drift. When a leaf receives new instances, it measures the radius of the cluster that is generated. By means of this value, the radius of the precedentedly observed cluster (whose statistics are kept in memory) and the distance between the centroids of the clusters, the algorithm distinguishes concept drift from noise. These statistics are saved at different times over the computation, thus allowing for detection of recurrent concepts. A pruning mechanism avoids overfitting of the decision tree to training data.

Another hybrid approach (classification and clustering) is proposed in [1]. Different from other algorithms in which a classification algorithm is aided by clustering, the algorithm in [1] is developed around the online clustering algorithm presented in [2] (described in Section 2.5.2). The algorithm maintains a set of micro-clusters, and

it is possible to select a time horizon of the micro-clusters to observe. A classification step based on k-nearest-neighbour is used to assign a class to the points of the clusters.

2.4.3 Unsupervised model updating

Even in absence of supervision it is possible to update the model of a classifier. The algorithm described in [78] uses supervision only in the initial phase of training of a classifier. A dataset D of instances with labels is used to generate the model h. After that, an unlabelled instance is presented and the model associates a class with it. The instance and its predicted class are added to the dataset D and the oldest instance of the dataset is discarded, thus generating a dataset D'. A new model h' is then trained on D'. The process is repeated for every new instance that is presented.

This method can be used to update the model of classifiers that are not designed to handle data streams, even with concept drift. In particular, the algorithm in combination with nonadaptive classification techniques managed to classify data streams with concept drift [78].

2.5 Clustering of data with concept drift

This section is concerned with the definition of the problem of clustering for data streams with concept drift, along with the presentation of some of the related techniques. In order to prepare the discussion, an introduction of the more general problem of clustering is given. This section also covers the problem of semi-supervised clustering that concerns the use of labelled data to aid the formation of clusters [34].

2.5.1 Introduction: clustering approaches

Clustering is concerned with identifying groups of observations with similar characteristics. In order to do this, distance is used to measure the similarity between couples of observations. Clusters of observations are formed by grouping together instances that are close to each other and far from the instances of other clusters [40].

The result of a clustering can have different forms. For instance, *partitional*

clustering techniques deliver a single partition of the set of observations, while *hierarchical clustering* techniques generate a hierarchy, which gives more flexibility as multiple solutions can be derived from the hierarchy [34]. The types of data that are processed by a clustering technique can also be different. These include numerical data and categorical data. While some methods make use of representations of the data, such as centroids or distribution estimates, in order to define clusters, others merely rely on pairwise comparisons between observations. The use of representations generally reduces computational costs [34]. There can also differences in the membership of an instance to a cluster. *Fuzzy* techniques account for instances that belong to multiple clusters at the same time, with continuous degrees of membership, while for *crisp* techniques, an instance cannot be associated with more than one cluster [40].

A first category of partitional clustering techniques uses prototypes to represent clusters. In particular, it is common to use centroids as prototypes. These techniques search for the prototypes that minimise the overall sum of the squared distances of the instances from their respective prototypes [34]. The algorithm *k-means* belongs to this category [36]. This algorithm works under the assumption that the number of clusters k is known in advance. k-means generates an initial set of centroids randomly. Then, each point is assigned to its closest centroid, thus generating a set of clusters. These are gradually refined by recomputing the centroids and repeating this process iteratively, until a stopping criterion is met.

Density-based techniques use a different approach. In fact, they follow the principle that a cluster should be formed by points from a region of the space of the attributes with high density, while different clusters should be separated by low-density regions. The algorithm DBSCAN represents a typical example of density-based clustering [28]. This technique is based on the concept of *density-connectivity*. A data point p_2 is *directly-density-connected* with another point p_1 if their distance is not higher than a predefined maximum distance ϵ, and p_1 is surrounded by at least a certain number of points. The concept of connectivity can also be extended to distant points. In particular, two points p_1 and p_n are *density-connected* if is possible to identify a sequence of points $p_1, \ldots, p_n$ such that, p_{i+1} is directly-density-connected with p_i, where $i = 1, \ldots, n-1$. DBSCAN considers a subset of points that are density-reachable as a cluster.

Mixture-resolving techniques assume that the clusters have a certain distribution

(for example Gaussian) of which they estimate the parameters in order to fit the data. Hierarchical clustering techniques generate a hierarchy or *dendogram*. That allows choice from a set multiple solutions by simply pruning the dendogram in different ways [34]. This gives high flexibility in the choice of a suitable clustering solution. An approach to hierarchical clustering, known as *agglomerative* clustering, intitially considers each instance as a cluster and then merges clusters together until some desired properties are reached. Another approach, called *divisive* clustering, considers all the data as a single cluster, that is split repeatedly into subclusters [65].

2.5.2 Online clustering

This section provides a brief overview of online clustering techniques. In particular, it focusses on two different approaches to clustering data streams: the *prototype-based* approach and the *density-based* approach. The former assumes that the data is clustered around a predefined number of centroids, the latter identifies clusters of points with sufficiently high density.

2.5.2.1 Prototype-based online clustering: CluStream

A first category of techniques uses prototypes to represent clusters. It is common to use centroids as prototypes. These techniques search for the prototypes that minimise the overall sum of the squared distances of the instances from their respective prototypes [34].

The algorithm CluStream is a prototype-based technique for data streams [2]. It is based on the principle that retaining in memory all the instances observed up to a time point is unfeasible for data streams. For that reason, CluStream introduces *micro-clusters*. These are data structures that maintain statistics about a set of data points. When a point is represented by a micro-cluster, the statistics of this are updated and the point is discarded. In this way, the amount of information that is stored can be reduced drastically [2]. Micro-clusters, which are updated online when new instances are provided, act as points to an offline clustering algorithm that is run by a user when needed. More precisely, a micro-cluster is a tuple of five elements $(n, \overline{CF1^x}, \overline{CF2^x}, CF1^t, CF2^t)$, where n is the number instances added to the microcluster, $\overline{CF2^x}$ is a vector containing the sums of the values of the instances

for each attribute, $\overline{CF1^x}$ is a vector of the sums of the squared values of the instances for each attribute, $CF1^t$ is the sum of the time-stamps at which the instances were added and the sum of the squares of the same time-stamps $CF2^t$.

When a new instance is presented, it can be added to an existing micro-cluster, if the instance is sufficiently close to the micro-cluster (within a radius r from its center), or it can generate a cluster of its own. The statistics $\overline{CF1^x}$ and $\overline{CF2^x}$ are used to make this decision. As the number of micro-clusters is fixed, if a new cluster is formed, an existing cluster must be deleted, for example if if this is too old. Otherwise the closest two clusters are merged. The updating of the set of micro-clusters is performed in an online fashion.

The higher level clustering is performed offline. The centers of the micro-clusters are processed as points by a clustering algorithm in order to identify a set of high-level clusters. A modified version of k-means, is used for the offline step. This takes into account the number of points into a micro-cluster in order to initialise the algorithm.

The time-based statistics $CF1^t$ and $CF2^t$ allow for the selection of potentially different time horizons. This is implemented through a pyramidal time frame that stores snapshots of the micro-clusters at different times. The granularity of the snapshots varies according to the time at which these were recorded. The algorithm uses the subtractive properties of micro-clusters to generate a set of micro-clusters representative of a specified time interval [2].

2.5.2.2 Distance-based online clustering: DenStream

According to [18], CluStream suffers from several limitations. Since it uses a variant of k-means as an offline clustering technique, CluStream is only able to identify spheric or ellipsoidal clusters. For the same reason, as k is pre-defined, CluStream cannot deal with data whose number of clusters changes over time. The use of a fixed number of clusters also makes CluStream not suitable to handle outliers [18].

In order to overcome these limitations, a density-based algorithm called DenStream is proposed in [18]. The algorithm features a two-level clustering similar to that proposed in [2], comprising an online stage and an offline stage. DenStream also implements micro-clusters, however, these are different from those observed in [2]. In fact, they are divided into potential micro-clusters (p-micro-clusters) and outlier micro-clusters (o-micro-clusters). The former ones are meant to be representive of

groups of points with high density, while the latter represent outliers or newly-formed groups of points that could potentially grow into p-micro-clusters. More precisely, a p-micro-cluster is a triple $(w, \overline{CF1^x}, \overline{CF2^x})$, where $\overline{CF1^x}$ and $\overline{CF2^x}$ are the vectors used by CluStream, while w is a weight. A p-micro-clusters that does not receive new points for several iterations is deleted if its weight decreases under $\beta\mu$. Where μ is a variable that determines the magnitude of the weights of p-micro-clusters, while β is a multiplying factor for μ, with $0 < \beta \leq 1$. The weight of a micro-cluster depends on the weights of the points it represents. The weight of a point decreases with time according to the formula $weight(p_i) = e^{-\lambda t}$, where t is the time elapsed since the point was presented, while λ tunes the rate of forgetting. In this way, points with different weights give different contributions to the construction of the center and the radius of the cluster they belong to. The set of p-micro-clusters is initialised with an run of DBSCAN over the first instances that are presented. An outlier micro-cluster, has the same fields as a p-micro-cluster, plus an additional field t_0. That is the time at which the cluster is created and is used to distinguish outliers from potentially-growing clusters.

When a new point is presented, it can be added to the closest p-micro-cluster if the new radius is below a threshold ϵ. Else, it is added to the closest o-micro-cluster, subject to the same check of the radius. In this case, if the weight of the o-micro-cluster becomes higher than $\beta\mu$, it is promoted to p-micro-cluster. Otherwise, the point will generate a new o-micro-cluster.

A modified version of DBSCAN is used for the offline clustering process. In this case, the concept of density-reachability is used in place of that of density-connectivity. In particular, two micro-clusters are directly-density-reachable if their radiuses are tangent or overlapped. A set of density-reachable points forms a cluster.

2.5.3 Semi-supervised clustering

Section 2.4.2 showed that semi-supervision could help increase the performance and reduce the amount of supervision of a classifier. In particular, it was shown that semi-supervised classifiers use large amounts of unlabelled data to build better classification models.

Semi-supervision is not only limited to supervised techniques. As a matter of fact, additional information can be used to guide the formation of clusters [21]. For example, there could be contraints dictating that two points must belong to the

same clusters (*must-link*) or that they must belong to different clusters (*cannot-link*). Alternatively, labelled data can be used to generate clusters. In this case, points with different classes should belong to different clusters.

In this context, two types of semi-supervision can be distinguished. In *similarity-adapting* methods the measure of similarity takes into account the constraints dictated by the additional information that is pro1vided [10, 21, 34]. In *search-based* methods, it is the algorithm that utilises constraints or labelled data in order to search for a solution [34].

The variants of k-means described in [5] represent an interesting example of search-based semi-supervised clustering. The techniques is called *seeding* and consists of using a small amount of labelled data to guide a clustering. While in the original version of k-means the initial set of centroids is generated randomly, both of the variants presented in [5] use the centroids of the points of each class are used as "seeds" for the clusters. In such a way, the information contained in the labelled data helps forming better clusterings when compared with standard k-means [5]. In addition, the second variant at each "refinement" step checks that points of different classes belong to different clusters.

2.6 Unsupervised drift detection

The methods presented in Section 2.4.1.5 require supervision in order to detect drift, as they measure the error of a classifier. There are other methods that do not make use of supervision to deal with this problem. These methods are related to the areas of signal processing, comparison of density estimations, statistical analysis, among others [26]. The descriptions of two of these methods are presented. They operate on the input features rather than the decisions of a classifier.

The algorithm in [37] exploits supervised techniques to perform unsupervised drift detection. The true classes of the instances, however, are not used. In fact, given two different datasets, the algorithm assigns positive class to the instances of a dataset, and negative class to the instances of the other dataset. Then, a classifier is trained on the modified datasets. If these are generated from the same distribution, the expected value of accuracy p_{bin} should follow a Bernoulli distribution. Hence, if a dataset has size m, the other has size n, and $N = m + n$, the expected accuracy

of that classifier is:

$$p_{bin} = \frac{max\{m, n\}}{N} \tag{2.24}$$

Given a confidence level α and a value of accuracy p, the null hypothesis, that states that the datasets are generated from the same distribution, is rejected according to the result of the following inequality:

$$\sum_{i=Np}^{N} \frac{N!}{i!(N-i)!} p_{bin}^{i} (1 - p_{bin})^{N-i} \leq \alpha \tag{2.25}$$

The Wald-Wolfovitz test is a nonparametric test that measures the similarity of two sets of univariate values [95]. After assigning ranks to the values of the datasets, it counts the number of *runs* R, where a run is a sequence of consecutive values from the same dataset within the ranking. When the distributions of two datasets are different, small values of R are likely to observed. For instance, if the two sets of values lie in distinct parts of the domain of the variable, all the values of each dataset will be contiguous within the ranking and, therefore, $R = 2$. By contrast, if two datasets are similarly distributed, after ranking, observing a long sequence of ranks associated with values from a single dataset is unlikely, and the value of R will be high. An extension of this method to problems that use multivariate data is proposed in [30]. The algorithm firstly creates a graph that connects nodes representing the instances. An edge is weigthed with the euclidean distance between the instances associated with the nodes it connects. Then, the algorithm finds the *minimum spanning tree* (mst) of the graph. Among the trees that connect all the nodes of a graph, the minimum spanning tree is the tree that minimises the sum of the weights of its edges. Subsequently, the edges that connect nodes related to instances of different sets are removed. The number of subtrees that is generated is equivalent to the number of runs for the univariate case. Therefore, that number is also referred to as "number of runs" and it is indicated with R. Let us consider two datasets, containing respectively m instances and n instances. If the datasets are generated from a common distribution, the mean value of R is calculated by the following:

$$\mu = \frac{2mn}{N} + 1 \tag{2.26}$$

where $N = m + n$. The variance of R is expressed as:

$$\sigma^2 = \frac{2mn}{N(N-1)} \times \left(\frac{2mn - N}{N} + \frac{C - N + 2}{(N-2)(N-3)} \times (N(N-1) - 4mn + 2) \right) \tag{2.27}$$

C is defined as $\left(\sum_{i=1}^{N} d_i(d_i-1)\right)/2$, where the *degree* d_i of the node i is the number of edges that are connected with it. According to the central limit theorem, the quantity:

$$W = \frac{R-\mu}{\sigma} \tag{2.28}$$

is normally distributed [30]. Therefore, according to a confidence level α, if R lies outside the interval defined by α, the null hypothesis is rejected.

2.7 Adaptive frameworks for classification

Researchers are looking at ways to increase the level of adaptivity of classification, in order to develop systems that are able to adapt to environments with different and potentially unexpected characteristics. For that purpose, solutions have been proposed. They present different solutions that implement the principles of modularility, reconfigurability and interconnection, among others.

An example of an adaptive architecture is proposed in [41]. That architecture distinguishes three levels of information processing: *computational path level*, *path combination* and *meta level*. The computational level is concerned with the creation, modification and deletion of computational paths. These are sequences of information processing stages that transform input data into decisions. Computational paths combine components selected from a library of pre-processing techniques (e.g., for feature selection or feature extraction) and a library of computational techniques (e.g., SVM and multilayer perceptrons). Different computational paths can be trained on different regions of the input space. In fact, an input space may have regions with diverse characteristics that may require different classification techniques. Given the variety of components that can be used, this is the level with the largest diversity. The different channels of information generated at the computational path level are merged at the combination level. That level exploits the diversity of the computational paths by combining them in different ways in order to generate more accurate predictions. The meta level has the purpose of optimising the performances of the components (local), as well as the performance of the architecture (global). Actions taken at the meta level may involve the change of the parameters of a technique, the creation of new computational paths or the choice of an alternative combination method. The meta level requires measurements of the performance of the components within the architecture.

This architecture can operate in the traditional batch mode, which comprises training, evaluation and testing performed over predefined datasets. Alternatively, it can operate in incremental mode, that accounts for data provided in the form of a stream. This architecture provides perfomance evaluators such as the mean square error (that makes use of the target values of the data that are occasionally provided). However, additional performance evaluators (potentially based on different principles of operation) can be included by an external user. The architecture has been applied to real datasets, on which it outperforms state-of-the-art techniques [42].

Evolving connectionist systems (ECOS) are systems that adapt to deal with different and potentially unexpected conditions provided by the environment in which they operated [42]. The ECOS framework defines a set of modules that perform different tasks such as feature selection, representation of information, high-level decision making, knowledge storage, action and adaptation. In particular, modules are implemented trough the paradigm of fuzzy neural networks [43]. A characteristic of the framework is the bidirectional interaction with the environment in which they operate. In fact, besides collecting information from an environment, ECOS systems can also act on it. In particular, the evolving fuzzy neural network model (EFuNN), that performs both supervised and unsupervised learning, has shown to be effective at including new input attributes at runtime.

Although the framework presented in [51] is only outlined, it proposes several interesting ideas with the intention to increase adaptivity of classification systems. For example, the concept of an "ensemble", which is normally associated with classification, here is extended to the lower level of *feature extraction* and higher level of *decision making*. Figure 8 (a) shows the model of the framework. The term "information vector" refers is used to indicate vectors that can convey different types of information (signal, feature or decision). Input information vectors are processed in a parallel way by a set of stovepipes. Depending on the level, a stovepipe can be a feature extraction technique, such as PCA, mean or kurtosis; a classifier such as a *MLP* or *SVM*; or a technique to fuse decisions, for example voting. The outputs of the stovepipes are combined by the "Data Fusion" module to generate the output information vectors IV_{output}. In the simplest case the data generated by the stovepipes can be simply collected into the output vector IV_{output} without any processing, alternatively, the data can be combined, for example, by means of

voting, averaging or analogous technique. The model of the adaptive framework also introduces the concept of *feedback*. Besides considering feedback as generated by an external entity or connecting internal components of the framework, feedback is also considered as the reuse of the output of a component as an input to another component. For example, the prediction of a classifier can be used to train another classifier.

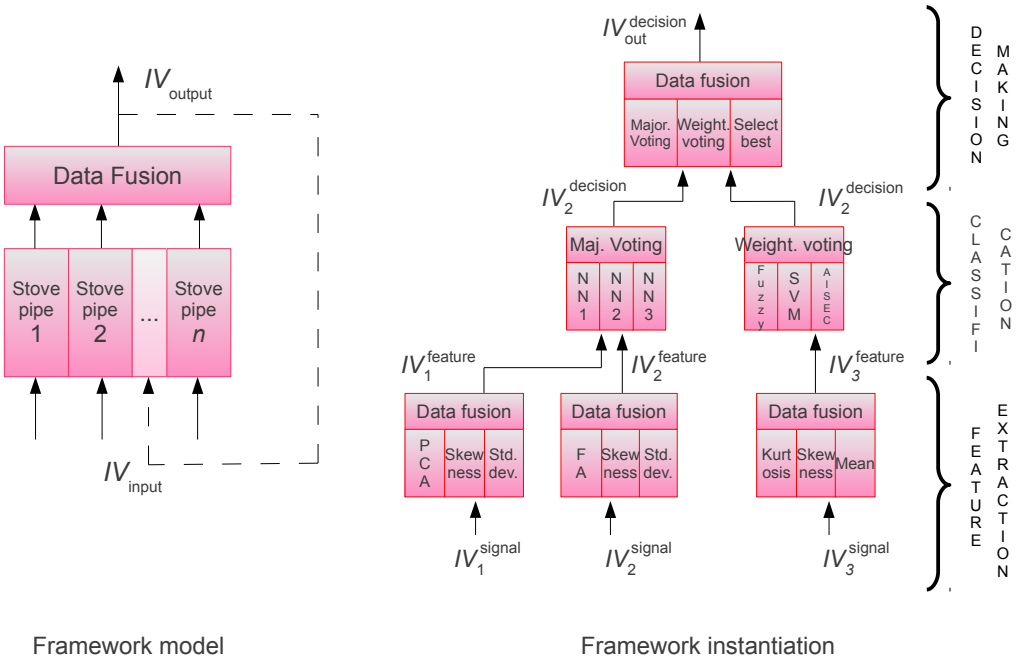


Figure 8: Model of the adaptive framework (left) and a possible instantiation (right) [51]. In the model of the framework (left), the input information vector IV_{input} is processed by a set of stovepipes. Their output is combined by the data fusion module to produce the ouput vector IV_{output}, which can also be fed-back to the input. The diagram on the left shows a possible instantiation of the framework.

When it is instantied, the framework may look like the example shown in Figure 8 (right). Starting from the bottom, signal data generated by sensors is provided to a set of feature extraction "boxes" or ensembles, where feature extraction techniques, acting as stovepipes, process the inputs in parallel. The outputs from the stovepipes are combined into a single output vector of features per box. The three features vectors generated $IV_1^{feature}$, $IV_2^{feature}$ and $IV_3^{feature}$ constitutes the inputs to the classification boxes, that are substantially ensemble classifiers. These can contain classifiers of the same type, as shown in the ensemble on the left, or they can mix

different techniques as shown in the ensemble classifier on the right. The top level, fuses the decision vectors produced by the ensemble classifiers to generate a single label to associate to the inputs.

The framework affords adaptivity by changing its structure or the parameters of its components, we refer to them using the terms *structural adaptivity* and *parametric adaptivity*. Structural adaptivity concerns the creation or the deletion of an component of the framework, such as a stovepipe, a data fusion module or an entire box, but it can also regard a change in the connections between existing elements. Parametric adaptivity does not affect the structure of a system, only its parameters. For example, a stovepipe which with low performance might require tuning of its parameters or, for example, the weights of a data fusion module may be revisited. Structural adaptivity could be used to delete or replace uneffective components.

The way information is processed by the adaptive framework is similar to that of the adaptive architecture. In fact, both use different channels of information processing and can use different techniques to combine predictions. However, the development of the adaptive architecture is at an advanced stage, as implementions have been tested on real-world problems. By contrast, the framework was only outlined in [51]. However, that gave us freedom to develop new ideas. In particular, we investigated the concept of feedback it presents, with the aim to increase the adaptivity of a classifier. Despite its versatility (related to its wide range of adaptive modules and their high interconnection), the ECOS framework is centred around the technique of neural network. That can be a limitation if other techniques need to be used. On the contrary, the adaptive framework and the adaptive architecture do not limit the types of techniques that can be used.

2.8 Summary

Classification is concerned with extracting patterns from a dataset, and use those patterns to classify unseen instances (Section 2.2). Traditionally, the problem of classification assumes that the data distribution does not change. This is not the case of numerous applications, in which concept drift affects the data. In order to avoid a model to become obsolete, this is updated with new data over time. Different levels of supervision can be provided with such data. For supervised techniques ((Section

2.4.1)), receive the true class of each instance that they process. In this way, they can measure the error rate of their models. Among the supervised techniques, we can distinguish between triggering techniques, that use explicit detection of drift to trigger adaptation and evolving techniques, in which adaption is driven by the goal of maximing classification performance. In particular, ensemble-based techniques evolve a set of classifiers. Using multiple classifiers, in general, affords better generalisation and is an effective approach to incremental learning (Section 2.2.3). Semi-supervised techniques use large amounts of unlabelled data to improve their performance when little supervision is available (Section 2.4.2). Unsupervised techniques do not use supervision to update a model (Section 2.4.3).

The problem of concept drift also concerns the area of clustering, for which several solutions have been proposed (Section 2.5). In addition, semi-supervision is also related to clustering. In this case, labelled data is provided to improve clustering performance. Unsupervised drift detection methods use raw data to reaveal the presence of drift. They are different from the methods proposed in (Section 2.5.3), for which supervision is necessary in order to detect drift.

In particular, the solution that is presented in this thesis draws from the areas of unsupervised model updating and ensemble classfiers. The techniques of online clustering and semi-supervised clustering could also be used to deal with this problem. Unsupervised drift detection techniques will be compared with the inference of drift of the implementation of the framework that is ollustrated in the next chapter.

Chapter 3
Adaptive framework

3.1 Introduction

The previous chapter presented an overview of the techniques that have been developed to deal with data containing concept drift. These can be categorised on the basis of to the amount of supervision they employ. Some maintain their classification performance over time by means of intermittent supervision that is used to update the model of a classifier. This can be a problem for those applications in which providing supervision is not practical because of cost or time reasons [59]. In this case, semi-supervised techniques provide for the scarcity of labelled data by making use of unlabelled data, which is easier to collect for several applications. Other techniques make use of no supervision altogether. After generating a model, they update it only by means of the information extracted from unlabelled data. The goal of this thesis is to contribute to the reseach in this field by presenting a framework that does not require supervision to update the model of a classifier. In particular, it presents an implementation of the ideas of the adaptive classification framework outlined in [51]. The framework makes use of multiple and diverse techniques at the different levels of feature extraction, classification and decision fusion. Moreover, it also suggests the use of feedback to increase the adaptivity of classification systems. The implementation of the framework being proposed introduces two major novelties. The first novelty is the combination of ensemble classification and a mechanism of training of classifiers that reuses the decisions of a model that, under particular concept drift conditions, allows supervision to become unnecessary. This mechanism, which draws upon the concepts of self-training, is similar to the method proposed in [78]. The second novelty consists of a mechanism for comparison of the decisions of the classifiers of an ensemble. This mechanism

has the purpose of inferring the presence of drift and is responsible for the updating of the model of the framework.

The rest of the chapter is structured as follows. Section 3.2 defines the problem and motivates the need for further investigation in this direction. Section 3.3 illustrates our implementation of the framework. The description of the algorithm starts with the ensemble of classifiers which represents the base of the framework. Then, a mechanism that reuses the decisions of generated by the framework is illustrated. It allows the framework to adapt to concept drift. Finally, a method to infer concept drift is described. Section 3.4 presents a summary of the chapter.

3.2 Motivation

This thesis focusses on the problem of classification of streams of data affected by concept drift. The problem of concept drift, described in Section 2.3, affects the performance of a classifier as it changes the distribution of the data on which the classifier operates. Several approaches have been proposed to deal with this problem. They can be classified according to the amount of supervision that they require.

Supervised learning algorithms, presented in Section 2.4.1 update the model of a classifier with the true class of an instance, which is provided after that that instance has been classified. This allows for measurement of the error rate of a classifier, and, if it increases, update its model. Since providing supervision may be expensive, impractical or even not feasible in some cases, several approaches have been developed to reduce the amount of labelled data required by a classifier. Similar to supervised classifiers, semi-supervised classifiers (Section 2.4.2) are also updated by providing the class of an instance after classification. However, that information is not available for every instance. On several testbeds, these algorithms perform comparably with supervised techniques, although semi-supervised techniques use only small fractions of the labelled data that supervised classifiers require. The possibility of updating a model without supervision has also been considered in the literature, as shown in Section 2.4.3.

Although this area of research has a lot of potential applications, and increasing attention is being placed on the reduction of supervision for classification, more effort has been spent on the investigation of supervised and semi-supervised learning rather than unsupervised model updating. We believe that, for certain problems,

learning algorithms should gradually become less demanding of supervision. For that reason, the goal of this thesis is to gain a better understanding of the problem of unsupervised model updating. In particular, this investigation is characterised by an analysis of the conditions that are required in order to perform unsupervised model updating. This thesis also proposes a framework for unsupervised model updating over concept drifting data. The proposed approach is compared against alternative methods that deal with the same problem. This comparison has the purpose of identifying the advantages and the limitations of each method under different conditions, and therefore getting a better insight into the more general problem.

We now reconsider the problem of updating a model of data with concept drift through the lens of Bayesian theory and of the different probabilites that it involves (Section 2.2.1), in the different scenarios of supervised, semi-supervised and unsupervised model updating. Let us start from the supervised case. If the time interval over which data is collected is sufficiently small with respect to the rate of drift, we could assume that the instances have the same distribution. The data collected from the stream is representative of the joint distributions that generated the data, one per class. Joint distributions carry information about the class-conditional distributions, the prior probabilities of the classes and, as a consequence, also the class-unconditional distribution. This information could potentially be used to deal with several types of drift. In fact, for example, that information could be used to estimate the posterior probabilites by using generative models. In this way, even if the classes are highly unbalanced or "overlapped", if classes are deleted, swapped or new classes are introduced, or if the rate of drift is high it should in theory be possible to maintain a relatively high classification performance. This reasoning also applies to the semi-supervised case. In fact, although the smaller number of labelled instances conveys less information about the joint distributions, this information can be reconstructed using unlabelled data. This, of course, is not possible for every scenario.

The context of unsupervised model updating is different from the supervised case and the semi-supervised case. The method described in Section 2.4.3 showed that it is possible to update a model without using supervision if the data has simple characteristics (e.g., separable classes). Let us interpret those results by analysing the probabilites that are involved. We suggest that training data is used to build

a representation (the model) of the classes and their joint distributions. After the training phase, only unlabelled data is provided. This data conveys information about the class-unconditional distribution. In this context, an unsupervised learning algorithm should update the information about the classes (extracted from the labelled data during the training phase) by means of the information of the class-unconditional distribution that is conveyed by the unlabelled data. In order to cope with concept drift, this information is updated as new unlabelled data is processed. According to this interpretation, if the class-unconditional distribution is not affected by concept drift, we would not be expect the updating of a model to be possible without supervision. An investigation of these considerations in presented in Chapter 4.

We now present a selection of unsupervised techniques that are used or that could potentially be used to deal with concept drift. The technique described in Section 2.4.3 uses a sliding window of fixed size. Initially the window only contains labelled instances. When an unlabelled instances is presented, it is classified by an existing model. Then, the instance with its associated prediction is added to the window and the oldest instance in the window is discarded. After that, a new model is generated from the data in the window. This technique was able to classify that data affected by concept drift. Another potential way to tackle the problem of unsupervised model updating of drifting data could draw upon the area of unsupervised learning. Section 2.2.4 showed that clustering can also be used to perform semi-supervised classification. After clustering of a dataset, the majority label contained in a cluster is assigned to it. When a point is presented, it is classified with the label of the cluster that is closest to it. This idea could be extended to the classification of data with concept drift by using online clustering techniques. As a matter of fact, initially, an online clusterer could be "trained" on some labelled data, in order to identify a group of clusters and associate labels to them. When, supervision is no longer provided, the clusters could be used to classify unlabelled instances. Moreover, after classification, the same instances could be used to update the clusters to the changing distribution. A similar result can be achieved using a different approach. Section 2.5.3 shows that labelled data can be a support to the generation of clusters. This is known as semi-supervised clustering. Semi-supervised clustering also accounts for the assignment of labels to clusters. In an online context, training labelled data could be used to seed and label a set of clusters of an online

clustering technique. Afterwards, the model that is established could be used to classify incoming unlabelled instances.

The aim of this thesis is to further investigate the problem of classification of drifting data without information about the classes of the instances for updating a model. Section 2.2.3 described the advantages brought by ensemble classifiers to the field of classification. Among these, increased generalisation capability, possibility of combining different techniques, incremental learning of large datasets, robustness to noise. A large number of methods that make use of ensemble learning have been developed to deal with data affected by concept drift. A selection of these methods was presented in Section 2.4.1.4. This thesis intends to investigate whether ensemble learning could bring advantages to the area of unsupervised model updating. For instance, we aim at determining whether the use of ensembles could lead to better generalisation, higher performance, reduced memory requirements. Another goal is to investigate new methods of drift detection. In a context in which it is not possible to use the reference of labelled data, a solution to this problem may come from the comparison of the different characteristics of the classifiers within an ensemble. In this context, the term "inference" is preferred to the term "detection". In fact, while using statistical tests or using labelled data is absolute as it allows to measure an error, the information available that comes from the pairwise comparison of the classifiers is only relative.

3.3 Framework implementation for classification

This section proposes an adaptive framework to deal with the problem of unsupervised classification of concept drifting data. This includes a mechanism that reuses its output decisions, thus not requiring knowledge about the true classes of the instances. This principle is similar to the retraining of classifiers described in [78]. We refer to this mechanism of retraining as *feedback*. This is different from updating the model with the true classes of the instances, which here is referred to as *supervision*. Moreover, the updating of the classification model of the framework is driven by a mechanism of drift detection, which is based on the comparison of the outputs of different classifiers.

As mentioned in Section 2.7, the model of the framework accounts for two types

of components: *stovepipes*, which process input data in parallel, and a *data fusion* module, that combines the outputs of the stovepipes into a single vector as shown in Figure 8. Although the model can be instantiated at different levels, this thesis focusses on the intermediate level of classification. A box (that consists of a set of stovepipes and a data fusion module) instantiated at that level resembles in many aspects an ensemble classifier. In particular, central attention was put given to the shaping of the concept of feedback suggested in [51]. Despite the definition of feedback provided in [51] is intentionally generic, as at that time it was only outlined and not implemented. It was suggested that the output of a box (or a measure derived from it) could be used as an input or, more generally, to affect the internal state of a box.

Special attention has been paid to the analysis and the definition of the roles of the components and to the shaping of their relationships. This process has been facilitated by the use of modelling techniques. In particular, UML class diagrams have been used to define the relationships between the components, while activity diagrams were used to generate prototypes for the structural and the parametric adaptivity, as they help identifying drawbacks and benefits of the potential solutions. The diagrams that were developed also served as a preparatory phase for the subsequent implementation of the framework.

Figure 9 depicts the UML class diagrams of the model of the framework. As shown in the top of the figure, the framework model accounts for an arbitrary number of boxes. An instantiation of the framework can, in fact, contain multiple boxes at different levels (for example, feature extraction, classification, decision level) as well as multiple boxes at the same level. In such a way, an instantiation can be expanded vertically as well as horizontally. The diagram also states that a box must contain at least one stovepipe. Each box can contain multiple data fusion modules, although only one is actually used, the diagram however also considers the possibility of the data fusion module missing from a box to allow boxes made of a single stovepipe. The diagram accounts for two different types of connections. Internal connections only regard a single box, their purpose is to represent feedback, for example by linking the output of a data fusion module to the input of a stovepipe of the box itself. External connections regard distinct boxes, more precisely, the output of a box "feeds" another box. It should be noticed that in the definition of external connections there is no constraint that prevents the output of a box from being used

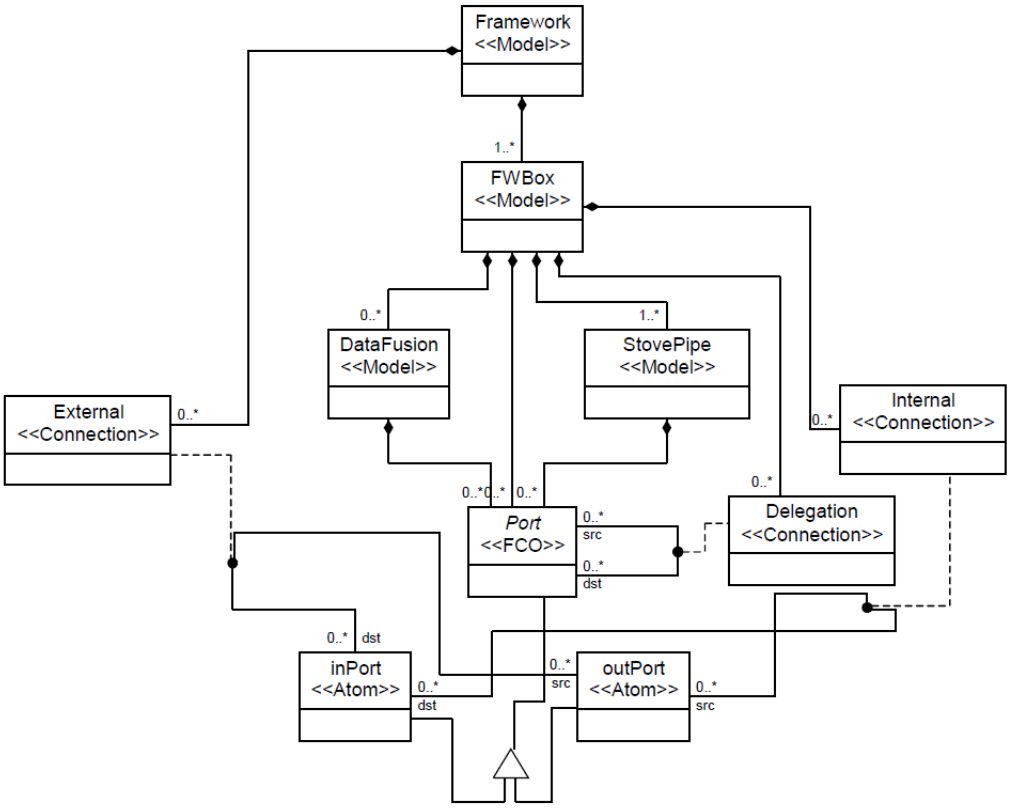


Figure 9: UML class diagram of the framework.

at a lower level. This type of connection could as well be considered as a form of feedback which involves different boxes, if this establishes a closed loop.

The central motivation for the framework is adaptivity, namely, the ability to react to changing conditions [3]. Adaptation can be driven by a number of reasons such as varied type and number of data inputs, concept drift, changing data rates or computational requirements. Attention has been dedicated to the analysis of the different forms of adaptivity the framework should be able to perform. This has brought to the identification of two general forms. Adaptation can regard the parameters of a component, in this case we talk about *parametric adaptivity*. For instance, a stovepipe for feature extraction can have its parameters modified or a data fusion module, such as a voting mechanism, might reconsider its weight to react more effectively to drift. Adaptation can also regard the structure of an instance of the framework, we refer to it as *structural adaptivity*. If the performance of a component degrades it might be a sensible choice to delete it, remove it, or perhaps replace it with a new component. Creation, deletion and therefore replacement can happen to any type of component: a stovepipe, a data fusion module and even an entire box. Structural adaptivity can also operate on the connections of an instantiation of the framework. For instance, if the combination of a classification technique with a feature extraction technique becomes ineffective, the classification technique can be associated with a different feature extraction technique. Both types of adaptivity should collaborate to guarantee the best performance in different conditions. For instance, parametric adaptivity could be used to optimise continuously the performance of a component, if parametric adaptivity is unsuccessful, then structural adaptivity can intervene, for example, by replacing the uneffective component with a new one.

The presentation of the implementation the framework is divided into three parts. It starts with the backbone of the framework: the box (an ensemble classifier), of which we delineate the characteristics of stovepipes and data fusion module. The second part introduces the mechanism of *feedback* of its decisions. This mechanism of structural adaptivity allows for replacement of uneffective stovepipes (classifiers in this case) with new ones, and therefore affords the updating of the classification model of the framework. Then, a mechanism of drift inference which drives the replacement of stovepipes is illustrated in Section 3.3.3.

3.3.1 Ensemble classifier

The box containing stovepipes and data fusion modules constitutes the base of the framework. Its composition is not dissimilar from many other approaches that handle concept drift and it can, in fact, be assimilated to an ensemble classifiers. For this reason the terms "box" and "ensemble classifier" will be used interchangeably. Moreover, considering that this thesis deals with the problem of classification, we take the freedom to use the more specific term *classifier*, along with the more general term *stovepipe*.

From the review of the literature it is possible to distinguish different strategies to maintain the population of the ensemble as well as different mechanism to fuse the decisions of the classifiers. Concerning the size of an ensemble, three options have been explored: constantly growing, fixed size and variable size, while the most used techniques to fuse decisions are averaging, majority voting and weighted voting. For this implementation of the framework we opted for a basic configuration. In fact, the ensemble has fixed size and simple majority voting is used to fuse the decisions of the classifiers. In this way, it should be easier to analyse the mechanism that are behind the functioning of the implementation.

The ensemble is divided into the pools of *mature* classifiers E_M and *naïve* classifiers E_N. Mature classifiers generate decisions that train new naïve classifiers. Moreover, naïve classifiers do not participate in the decision making. They are continuously replaced within their pool and therefore contain up-to-date information about the data distribution. The purpose of the training phase is to populate the pool of mature classifiers E_M.

The pseudo-code 5 illustrates the training of mature classifiers. We are assuming that the ensemble processes a stream of data instances, and that, only for this phase, their classes are provided. After the training phase, only unlabelled instances are generated. Since data is presented in the form of a stream (one instance at a time) and considering also that batch techniques are being used, a variable $th_{training}$ divides the stream in a sequence of batches, each of which trains a mature classifier. That parameter guarantees that each batch should contain at least $th_{training}$ instances of each class. This has the purpose of avoiding the generation of batches that contain predominantly instances from one class. For simplicity, we refer to the stream of labelled instance as D_{train}. The mature pool E_M is initially empty (line 2). The

variable i maintains a count of the number of instances that have been observed, while j represents the ensemble size (lines 3-4). D_{temp} is a temporary storage for the instances of the stream D_{train} (line 5). It contains data for the training of a classifier, after which it is emptied. The training of new classifiers continues until labelled data is provided (line 6).

Algorithm 5 Pseudo-code of the training phase of the framework

```
1: procedure TRAIN(D_train, th_training)
2:     E_M ← ∅                                              ▷ the mature pool
3:     i ← 1                                                ▷ pointer for data instances
4:     j ← 1                                                ▷ pointer for classifiers
5:     D_temp ← ∅                                           ▷ temporary storage
6:     while i ≤ |D_train| do
7:         D_temp ← D_temp ∪ {D_i}
8:         i ← i + 1
9:         if (|{D_k ∈ D_temp|class(D_k) = +1}| ≥ th_training) ∨
   (|{D_k ∈ D_temp|class(D_k) = −1}| ≥ th_training) then
10:            train(E_Mj, D_temp)
11:            E_M ← E_M ∪ {E_Mj}
12:            j ← j + 1
13:            D_temp ← ∅
14:    end while
15:    return E_M
```

The inputs to the training procedure are the data stream D_{train} and the threshold $th_{training}$. The first lines (2-5) of the pseudo-code have the purpose of initialising E_M, i, j and D_{temp}. In particular, the pool E_M and the temporary storage D_{temp} are initially empty, while i and j are initialised with the value 1. Every new instance that is presented is added to D_{temp} and the counter i is incremented (line 6-8). When D_{temp} contains at least $th_{training}$ of class ω_1 and at least $th_{training}$ instances of class ω_2 (Line 9), a classifier E_{M_j} is trained from the data it contains (line 10). E_{M_j} is added to the mature pool (line 11). Then, the counter j (it represents the size of E_M) is incremented (line 12) and D_{temp} is emptied (line 13). In this way, $th_{training}$ and the length of the stream of labelled instances determine the number of mature classifiers in the ensemble. When the stream of labelled instances ceases, the ensemble is returned (line 14) and the classification phase begins.

3.3.2 Decision feedback

We now present the mechanism of feedback of the decisions of the ensemble, which is responsible for the updating of a classification model.

Updating the model of a classifier can be achieved in two ways. The first option involves revisiting the internal parameters of a classifier, such as the weights of the hyperplane of a SVM, the structure of a decision tree or the set of rules of a fuzzy classifier. This implies that knowledge about how a particular technique structures the models of its classifiers is required (this knowledge would be used to develop strategies to change the model). This option therefore contrasts with the requirement that the framework has to be technique-independent, since it would entail that a classifier is not treated as a black box. The second option implicates the retraining of a model or part of it. A common strategy in ensemble methods for concept drift involves the deletion of a classifier and the training of a new classifier. However, labels are needed in order to train a classifier. This contrasts with the assumption that our implementation of the framework processes only unlabelled data. For that reason, the framework implementation features a mechanism that combines a prediction and the input vector that generated it, in order to train a classifier. Notice that supervision is not required. In fact, the labels that are used are the result of the classification of the instances, they do not represent the true classes of the instances, which are unknown. This mechanism is similar to that described in Section 2.4.3. However, for that algorithm, the decisions are generated by a single classifier, while we propose the generation of decisions by means of an ensemble of classifiers through voting. More precisely decisions are generated by mature classifiers, since naïve classifiers do not participate in the voting. Processed instances and labels are combined to train a new model that is added to the pool of naïve stovepipes.

The way data is processed is similar, for some aspects, to the supervised phase described in Section 3.3.1 and, for this reason, the two pseudo-codes share some variables. E_M and E_N represent, respectively, the pools of mature and naïve stovepipes. We are assuming that the training phase is finished, and therefore E_M is already formed, while E_N is empty. The stream of unlabelled instances is represented by D_{test}. The variable i counts the number of instances that have been processed, while j counts the number of stovepipes generated during the unsupervised phase.

The variable th_{online} is similar to $th_{training}$. In fact, it has the purpose of avoiding the collection predominantly instances from one predicted class. D_{temp} has the same function as in the training phase (Section 3.3.1): it is a temporary storage for new instances and it is emptied after a new classifier has been generated from its data. The variable $ratio_{naiveMature}$ determines the maximum size of the naïve pool of the ensemble.

The first lines (2-5) of the pseudo-code 6 have the purpose of initialising the variables i, j, D_{temp} and E_N. Then, the algorithm starts processing the instances of the stream one at a time (while loop, line 6). The unlabelled instance $\mathbf{x_i}$ is then classified by the mature classifiers through voting. Then, the instance $\mathbf{x_i}$ and its prediction are combined into d_i (line 7). After that, d_i is added to the temporary storage D_{temp} (line 8). This is the core of the feedback mechanism, in fact, every prediction is fed back and combined with the unlabelled vector that generated it. Then, the matrix used for the inference of drift is updated (line 9) and the i is incremented (line 10). When at least th_{online} instances of each class have been collected (line 11), a new naïve classifier E_j is trained with the data contained in D_{temp} (line 12). After that, the mechanism of drift inference is tested (line 13), j is incremented and E_j (line 14) is added to the naïve pool (line 15). That is added to the naïve pool and therefore becomes a candidate for replacing a mature classifier (lines 10-12). The first $\lfloor ratio_{naiveMature} * |E_M| \rfloor$ iterations of the online phase have the purpose of populating the naïve pool until its size reaches a fraction of the size of the mature set, dictated by the parameter $ratio_{naiveMature}$ (line 16). These steps are required to initialise the mechanism of drift inference, and more generally, the structural adaptivity of the framework. After that, if concept drift is inferred, a mature classifier is deleted (line 18) and a classifier is selected from the naïve pool (line 19) to replace a mature classifier (line 20). The naïve classifier that is selected is the one which best "generalises" the naïve pool. In particular, it is the one with the minimum distance from the other naïve classifiers. The mature classifier to be deleted is the most "distant" from the naïve classifiers. In this context, the distance between two classifier represents the number of instances in a window on which two classifiers generate different decisions. A more detailed description of distance is given in Section 3.3.3, where the mechanism of drift inference is illustrated. Alternately, if no drift is inferred, the newly trained classifier E_j is added to the naïve pool (line 22). Then, D_{temp} is emptied (line 23).

Algorithm 6 Pseudo-code of the on-line phase of the framework

```
1: procedure TEST(E_M, E_N, D_test, th_online)
2:     E_N ← ∅                                          ▷ pool of naïve classifiers
3:     i ← 1                                            ▷ pointer for data instances
4:     j ← 1                                            ▷ pointer for classifiers
5:     D_temp ← ∅                                       ▷ temporary storage
6:     while (more instances available) do
7:         d_i ← (x_i, voting(E_M, x_i))
8:         D_temp ← D_temp ∪ {d_i}
9:         updateHammingMatrix(M, x_i, E_M, E_N)
10:        i ← i + 1
11:        if (|{d_i ∈ D_temp | label(d_i) = +1}| ≤ th_online) ∨
       ({d_i ∈ D_temp | label(d_i) = −1}| ≤ th_online) then
12:            train(E_j, D_temp)
13:            drift ← InferDrift(H, FIFO_ID, th_ID)
14:            j ← j + 1
15:            E_N ← E_N ∪ {E_j}
16:            if |E_N| = ⌊ratio_naiveMature * |E_M|⌋ then
17:                if drift = true then
18:                    deleteClassifier(E_M)
19:                    E_l ← select(E_N)
20:                    E_M ← E_M ∪ {E_l}
21:                else
22:                    removeClassifier(E_N)
23:            D_temp ← ∅
24:    end while
25:    return
```

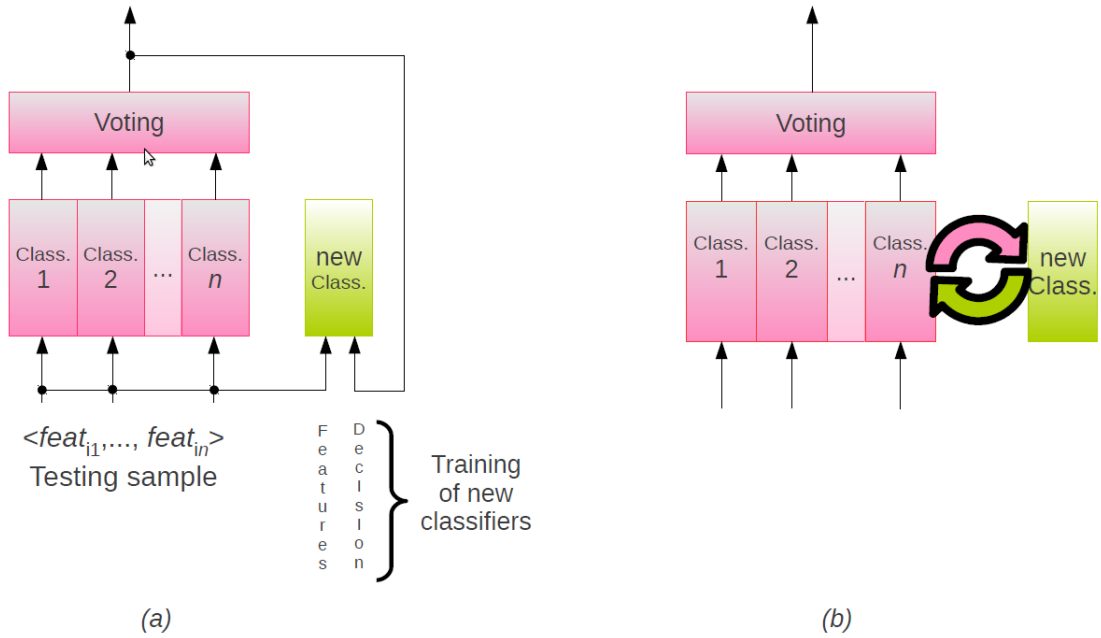


Figure 10: Feedback of the decisions of the framework.

3.3.3 Inference of drift

Some of the existing mechanisms that use the output of a classifier for detecting concept drift, use true classes and predicted classes to calculate an error. If such error is higher than a threshold, this is interpreted as concept drift occurring Section 2.4.1.5. More refined drift detection methods estimate the generalisation error through cross validation, also in this case knowledge about the true classes of the instances is required. While calculating an error is only possible when supervision is provided, if the true classes are unknown, different strategies can be used. As explained in the paragraph of Section 2.6 (concerning unsupervised drift detection), it is possible to analyse input data to detect concept drift.

A different avenue was pursued for the framework. In particular, the mechanism being proposed infers the presence of drift based on the analysis of the output decisions of the framework. The criterion that is adopted is based on the consideration that, since naïve classifiers are trained with data whose features are representative of $p(x)$ and the labels are generated from the mature pool, if the distribution is stationary, mature and naïve classifier should similar. By contrast, if the distribution changes in such a way that affects $p(x)$, naïve classifiers should embody that change and, consequently, they should differ from mature classifiers. We used the term "difference". However, what does difference exactly mean in this context, and how

can it be measured? Under the hypotheses that concept drift is ongoing and it affects $p(x)$, the models of naïve classifiers should start making different decisions on some of the input instances, therefore the Hamming distance between naïve and mature classifiers should increase over time.

The mechanism of inference of drift makes use of a matrix M of size $(|E_M| + |E_N|) \times (|E_M| + |E_N|)$, to monitor the Hamming distance between every pair of classifiers. In particular, the element M_{ij} of the matrix contains the distance between the classifiers i and j over the data in D_{temp}.

Algorithm 7 Pseudo-code of the mechanism of drift inference

```
1:  procedure InferDrift(H, FIFO_ID, th_ID)
2:      temp ← 0                                          ▷ temporary variable
3:      for j in [1, ..., |E_M|] do
4:          for i in [1, ..., i] do
5:              temp ← temp + M_ij
6:      d_M ← 2 * temp/(|E_M| * (|E_M| + 1))              ▷ determine D_M
7:      for j in [|E_M| + 1, ..., |E_M| + |E_N|] do
8:          for i in [|E_M| + 1, ..., i] do
9:              temp ← temp + M_ij
10:     d_N ← 2 * temp/(|E_N| * (|E_N| + 1))              ▷ determine D_N
11:     for i in [|E_M| + 1, ..., |E_M| + |E_N|] do
12:         for j in [|E_M| + 1, ..., |E_M| + |E_N|] do
13:             temp ← temp + M_ij
14:     d_MN ← temp/(|E_N| * (|E_M|))                     ▷ determine D_MN
15:     pop(FIFO_ID)
16:     if d_MN > d_N AND d_MN > d_M then
17:         attach(FIFO_ID, 1)
18:     else
19:         attach(FIFO_ID, 0)
20:     return sum(FIFO_ID) > ⌊th_ID * FIFOsize_ID⌋
```

In fact, during the construction of D_{temp}, the matrix M is updated based on the discordancies among the classifiers over each unlabelled instance $\mathbf{x}_i$ by means of the function $updateHammingMatrix(M, x_i, E_M, E_N)$ (line 8 of the pseudo-code 6). If two classifiers i and j assign different classes to an instance $\mathbf{x}_i$, then the element M_{ij} of the matrix is incremented by 1. When enough data is collected in D_{temp} and therefore a new classifier is trained from it, the matrix M is queried to infer the presence of concept drift. In order to infer drift, the framework calculates the mean distance among mature classifiers D_M, among naïve classifiers D_N, and

the mean distance between mature and naive classifiers D_{MN}. The distances are represented in Figure 11, which depicts an example of the matrix M. According to that example, both pools contain four classifiers. The areas highlighted in red, green and blue indicate respectively the regions of the matrix used to calculate, respectively, D_M, D_N, D_{MN}, corresponding to the lines 3-6, 7-10 and 11-14 of the pseudo-code 7. If D_{MN} is higher than D_M and D_N a 1 is added to the FIFO queue $FIFO_{ID}$ (line 17), otherwise a 0 is added to it (line 19). If the sum of the elements of $FIFO_{ID}$ is higher than $\lfloor th_{ID} * FIFOsize_{ID} \rfloor$ drift is inferred.

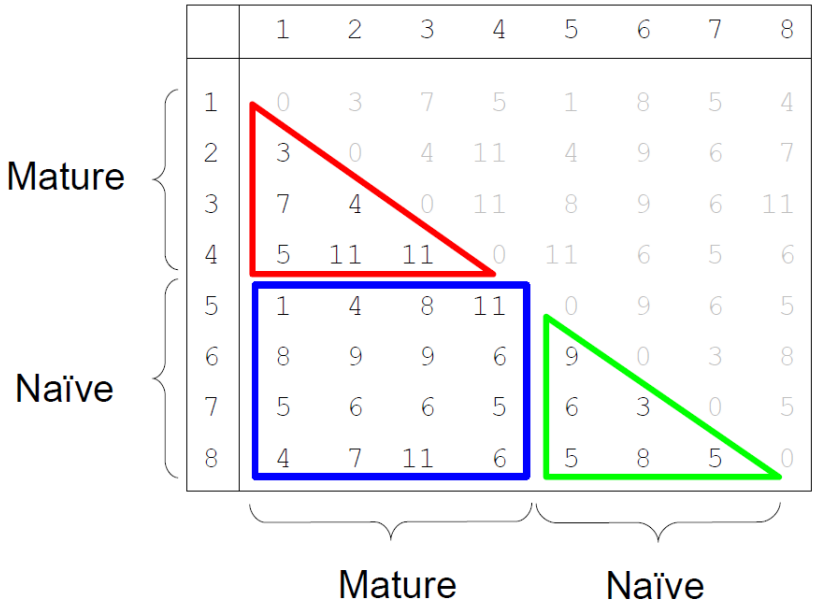


Figure 11: Example of the matrix of the distances M for an ensemble consisting of four mature and four naïve stovepipes. The matrix is square and symmetrical. The first four lines and columns are associated with mature stovepipes, while the remaining are associated with naïve stovepipes. The distances among mature stovepipes, naïve stovepipes, mature and naïve stovepipes are calculated by averaging the values in the areas highlighed using respectively the colors red, green and blue.

3.4 Summary

This chapter has presented an implementation of the ideas of the adaptive framework outlined in [51], with aim of reducing the amount of supervision utilised by classifiers. This investigation led to the development of an ensemble of classifiers that uses a mechanism of feedback of its decision to update its classification model. By means

of feedback, the system gradually retrains its model by adapting information about the classes, that is provided in the initial training phase, according to changes in the distribution revealed by the class-unconditional pdf. The members of the ensemble are divided into *mature* and *naïve* classifier. In particular, mature classifiers generate decisions that train naïve classifiers.

A mechanism for inferring concept drift is also proposed. It monitors the similarity between the classifiers of the ensemble in order to infer concept drift. This mechanism is based on the observation that, when concept drift is changing the data distribution, naïve classifiers should be affected by that change and therefore should differ from mature classifiers.

The next chapter evaluates the implementation of the framework, and place it in the context of the existing techniques. Several experiments are presented, they assess the performance of the framework over different distributions and classification techniques.

Chapter 4
Experiments

4.1 Introduction

This chapter describes the experiments that have been performed to evaluate the implementation of the framework being proposed. The goal of these experiments is to assess the effectiveness of the implemented framework at classifying drifting data without using supervision.

In particular the experiments evaluate the classification performance and the drift detection performance of the framework with different classification techniques and data distributions. Moreover, a set of comparative studies are performed in order to determine potential advantages or limitations of the framework with respect to existing techniques. For such comparisons, a selection of techniques for unsupervised classification and unsupervised concept drift detection methods are used.

Section 4.2 evaluates the framework on a dataset with changing $p(x)$ and on a dataset with changing distribution but fixed $p(x)$, in order to show potential differences in its ability to deal with these distributions. The experiments of Section 4.3 measures the performance of the framework on different classification techniques. Those experiments include analyses of the sensitivity to reveal the effects of variations of the parameters of the framework on its performance and rationales for the results. In order to identify potential benefits and limitations of the framework, a comparison with a set of unsupervised learners is provided in Section 4.4. Section 4.5 evaluates the framework and its unsupervised comparative techniques on a dataset containing multiple clusters with changing shapes and high overlapping of the distributions of the classes. The experiments of Section 4.6 evaluate the mechanism of drift inference. They use data with separable classes and data with

overlapped classes. The inference of drift is also compared with two methods that use different principles of detection. The conclusions are presented in Section 4.7.

4.2 Investigation of data requirements for unsupervised model updating

The aim of this section is to study the performance of the framework on data with different characteristics. In particular, we intend to verify the statements made Section 3.2, which hypothesised that the class-unconditional distribution $p(x)$ of a dataset must be affected by concept drift in order for a classifier that does not use supervision to correctly classify its data. For this purpose, we test the framework on a dataset whose concept drift causes a change of $p(x)$, and on a dataset with concept drift but fixed $p(x)$. An experiment involving the use of Gaussian data is presented in Section 4.2.1, while Section 4.2.2 presents an experiment with a uniformly distributed dataset. Section 4.7 presents a summary of results of the experiments.

4.2.1 Classification of Gaussian data with the SVM

The goal of this experiment is to establish whether it is possible to update a classification model over data with concept drift and changing $p(x)$. For this purpose, an instance of the framework with the SVM as a base learner is tested on a dataset with nonseparable classes and Gaussian distribution.

Section 4.2.1.1 describes the experimental setup of the framework: the characteristics of the data distribution, the parameters of the framework, the parameters of the SVM and the performance measures. It also defines the hypothesis and explains how to determine the number of runs that are needed to test the hypothesis. Section 4.2.1.2 presents the analysis of the sensitivity of the parameters of the framework. Section 4.2.1.3 provides a comparison between the framework and a collection of supervised techniques, while Section 4.2.1.4 provides a rationale for the results.

4.2.1.1 Experimental setup

This section describes the characteristics of the data, the parametric setting of the framework and that of the SVM, the measures that are used to assess the performance of the framework.

Gaussian data distribution. The distribution of the data has simple characteristics: its number of classes is fixed and its concept drift is incremental. The data distribution is bi-dimensional and it contains the classes " $+1$" and " -1". Each class is defined by a Gaussian bivariate pdf. Initially, the position of the Gaussian distribution of class " $+1$" is given by the coordinates $(1, 1)$, while the center of the distribution of class " -1" is identified by the coordinates $(1.7, 1)$. The same standard deviation is used for both Gaussian distributions and both features. In particular, $\sigma = 0.2$.

The first 3000 instances of the data stream are labelled and are not affected by concept drift. This information is used by the framework to establish a model, through the training procedure 5 described in the Section 3.3.1. After the training phase, the stream only generates unlabelled instances that are affected by concept drift. Unlabelled instances are classified using the Procedure 6 described in Section 3.3.2. Concept drift is simulated by changing the position of the centers when a new instance is generated. In particular, the number of testing instances that is generated is $2,000,000$, and the positions of the Gaussian clusters are shifted by two units along the x axis at the end of a run, with respect to their initial positions. Figure 12 shows the position of the classes before concept drift starts (a) and at the end of the experiment (b). Notice that the relative position of a Gaussian cluster with respect to the other cluster does not change along the experiment. Different data streams are generated by initialising the data generator with different seed numbers. There are multiple reasons for such a design of the distribution. The distributions of the classes change, as their centers move. However, since the relative position of a Gaussian cluster with respect to the other does not vary, we can assume that an ideal classifier should maintain its performance unvaried along the experiment. This allows to measure a performance decrement caused by concept drift. Moreover, the fact that the Gaussian clusters move by two units along the x-axis, causes the distribution at the end of the experiments to differ considerably from the initial distribution. In this way, we expect the performance of a static classifier to drop over such a prolonged drift. By contrast, testing the framework on a less extended drift might not reveal the magnitude of the degradation of its performance.

In addition, the two million instances that are generated during concept drift should provide sufficient information to a classifier in order to update its model. If

a small number of instances is used, a classifier might not have enough data in order to generate accurate models across a run.

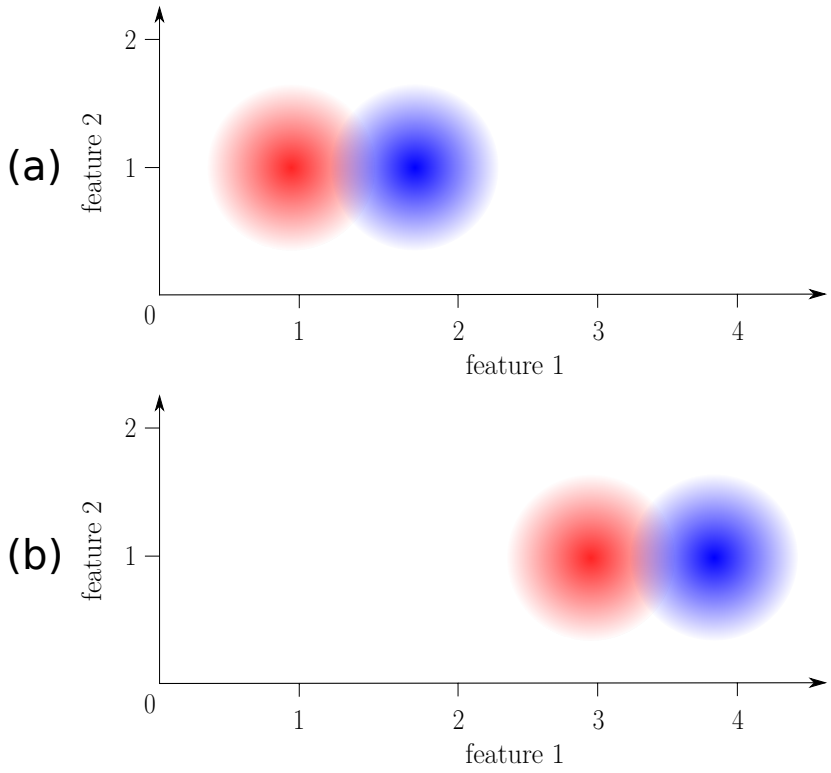


Figure 12: Representation of the Gaussian data. Distribution at the beginning of the experiment (a) and at the end (b). Notice that the position of the clusters moves along the x axis, but their relative position does not change.

Parametric settings. The configurations of the parameters of the framework and the parameters of the SVM are shown in Table 2.

The type of kernel is a radial basis function, while the cost function, that tunes the fitness of a model to the data, has a value of 10.

Concerning the framework, the parameter $th_{training}$ dictates how many instances of each class have to be collected from the initial stream of supervised data in order to train a mature classifier and add it to the ensemble. Considering that 3000 is the number of labelled instances and $th_{training} = 250$, generally a mature classifier is trained with at least 500 instances (at least 250 instances of each class).

	Parameter	Value
framework	$th_{training}$	250
	th_{online}	200
	$ratio_{NaiveMature}$	1.0
	$FIFOsize_{ID}$	7
	th_{ID}	0.3
SVM	-s (problem)	0 (classification)
	-t (kernel function)	2 (radial basis)
	-c (cost)	10

Table 2: Parametric configuration of the framework for the experiment involving the SVM and Gaussian data.

Therefore, the number of mature classifiers the framework will contain is 5, on average. Similarly, the value of th_{online} determines that at least 200 instances classified as "$+1$" and at least 200 instances classified as "-1" need to be collected in order to train a new naïve classifier. The value of the parameter $ratio_{NaiveMature}$, in this case, dictates that the number of naïve classifiers equals the number of mature ones. The values of the parameters $FIFOsize_{ID}$ and th_{ID} imply that the distance between mature and naïve classifiers d_{MN} must be higher than the distance between mature classifiers d_M and the distance between naïve stovepipes d_N in at least than 3 of the last 7 iterations, where an iteration involves the training of a naïve classifier (based on the description that of Section 3.3.3). In fact, the activation threshold is $\lfloor FIFOsize_{ID} * th_{ID} \rfloor = \lfloor 0.3 * 7 \rfloor = 2$ (according to the pseudo-code 7 of Section 3.3.3).

In order to evaluate the performance of the framework, *accuracy*, *precision* and *recall* are measured. Accuracy represents the percentage of instances that are correctly classified and is calculated by the formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

A true positive TP is an instance with label "$+1$" that is classified correctly, a misclassification of the same instance would generate a false negative FN. In a similar way, a true negative TN is a instance with class "-1" that is classified correctly, while a false positive FP is recorded when a negative instance is classified as positive. Precision, which represents the percentage of true positives among the instances that are classified as positive, is calculated by the formula:

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall represents the ratio of positive instances that are classified correctly among all the positive instances:

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

Hypothesis. This experiment evaluates the ability of the instance of the framework with the parameters of Table 2 to classify the Gaussian distribution. The null hypothesis is formulated as:

> *The performance of an instance of the adaptive framework featuring the SVM, when tested on the Gaussian dataset, is not statistically differerent from the performance of an instance of the framework without adaptivity, that is with a static model.*

In order to test the hypothesis, the framework was run 50 times over different datasets generated by the Gaussian distribution, and the median values of accuracy, precision and recall were recorded. An instance of the framework without adaptivity, and therefore with a static model, was tested on the same data. Then the two sets of results were compared by means of the Mann-Whitney test [58, 98]. Given two sets of values, the test evaluates the likelihood that values drawn from one set are larger than values drawn from the other sets. Since it is a nonparametric test, no assumption is made about the distribution of the data.

The test reveals that the hypothesis can be rejected with a confidence of 0.995. In fact, the probabilities that the values of accuracy, precision and recall of the static instance and those of the adaptive instance are generated from the same distribution are respectively $7.79E-10$, $7.813E-4$ and $7.79E-10$.

The box and whisker plots in Figure 13 display the spreads of the median values of accuracy, precision and recall across the 50 runs. The plot at the top shows the performance of the adaptive instance of the framework, while the results of its static counterpart are shown by the plot at the bottom. Notice that the plots adopt different scales. In fact, while for the adaptive instance of the framework the median values lie in a small interval, for the static instance that interval is considerably larger. For the adaptive instance, the spread of the values of accuracy, precision and recall around the 50^{th} percentile is limited. This suggests that the adaptive instance of the framework performs similarly over the 50 runs. Although the majority of the values for the static instance are distributed around their respective 50^{th} percentile

(bottom plot), the 0^{th} percentile of the values is much smaller.

The intervals of the values of accuracy of the two implementations do not overlap, this implies that the two implementations perform differently on this distribution. The same is observed for recall, but not for precision. An explanation for this is given in Section 4.2.1.4, along with an explanation of the reason why the values of accuracy lie inbetween those of precision, which are higher, and those of recall, which are smaller.

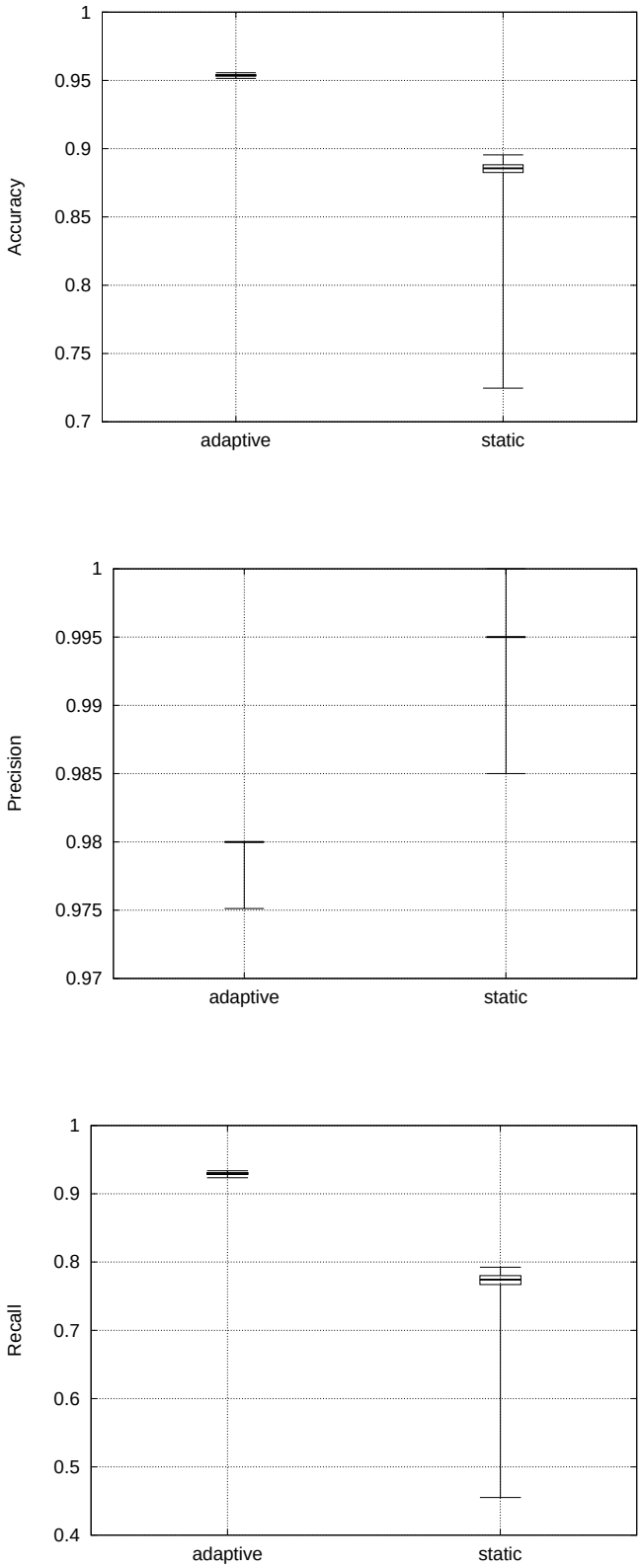


Figure 13: Comparisons of the values of accuracy, precision and recall of the adaptive framework and the static framework. Notice that different scales are used.

Determining the number of runs. The *A-test* was used to determine how many runs are needed for each experiment [45, 94]. That test is a statistical instrument that estimates the difference between two different sets of values by assigning ranks to their values. The A-test returns a score in the interval $[0.5, 1]$. It represents the probability of a randomly selected value from one set being greater than a randomly selected values from the other set. In [45], three different interval are defined, in particular if the score lies within the interval $[0.5, 0.56]$ the difference between the two distributions is considered small, the difference is medium in the interval $[0.56, 0.66]$, it is large for values within $[0.66, 0.73]$, while the difference is very large for A-test scores higher than 0.73.

In the case of the framework, the A-test was used to determine how many times the same experiment needs to be repeated in order for the distribution of an output measure to be considered "stable", in the sense that it does not change significantly if the same number of runs are repeated. In order to do this, we considered the median accuracy value of each run, and we compared sets having sizes of respectively 1, 5, 10, 20, 50, 100 and 200 runs. To achieve a greater confidence about the result, it was decided to compare ten rather than only two sets of values. In particular, a set of values that is used as a reference is compared against the remaining nine sets and the highest A-test score is recorded. Figure 14 shows the results the A-test scores in correspondence of different numbers of runs. For the values 5 and 10 the difference between the distributions of the median values of accuracy, precision and recall is large or very large, starting from 20 runs the difference can be considered medium. Only when 200 runs are performed the maximum difference between the ten distributions becomes small.

Although performing 200 runs would provide smaller statistical difference between the sets of values, the observation of the plots of the median value of each run convinced us that repeating the experiment 50 times is sufficient. This decision was taken in the light of the fact that all the median accuracy values lie within the interval $[0.949, 0.956]$, a very small subset of the set of values of accuracy. However, time also played a role in this decision. Since a single run requires about 20 minutes to complete, performing 200 runs would have been impractical.

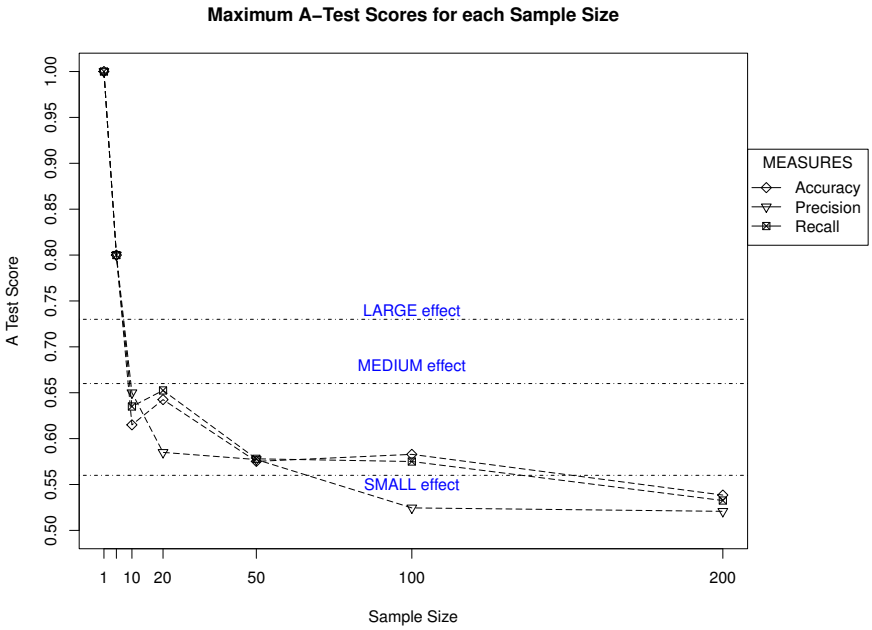


Figure 14: A-test results. Despite the lowest scores are obtained for 200 runs, we considered that repeating the same experiment 50 times is sufficient, given that all the median accuracy values fall in a very narrow interval.

4.2.1.2 Sensitivity Analysis

The analysis of the sensitivity of a system examines the influence of the input parameters on the output variables of the system. It involves the variation of the values of the input parameters and the measurement of the outputs, in order to study how the system reacts to such variation. Two distinct strategies can be identified. *One-at-a-time* techniques vary each parameter in turn with respect to a reference value, while keeping the other parameters unchanged. Although this strategy is generally easy to implement, it suffers the limitation that large regions of the space of the parameters are not covered by the sampling process. In this way it is not known how the system (the framework, for example) performs in those regions [80].

A much clearer view of the relationships between inputs and outputs is obtained if the parameters are varied simultaneously, this is known as *global* sensitivity analysis [80]. One of these techniques is the Latin hypercube sampling method. It consists of dividing the input space into a grid of cells across which a group of samples is generated [80]. For a bidimensional case, samples are generated in such a way that each row and each column do not contain more than a single instance. Plots are generated to show how different values of a parameter affect a performance measures.

In order to analyse the framework we opted for the Latin hypercube implementation in [45]. The input space is defined by the intervals of the parameters in Table 3, while the number of points that are sampled from the input space is 200. Concerning the outputs, the values at the end of a run are considered, rather than the medians. In fact, the measures at the end of a run show more clearly the ability of an instance to cope with drift. 4.

Parameter	**Interval**
$th_{training}$	[95, 480]
Th_{online}	[50, 500]
$ratio_{NaiveMature}$	[0, 1.0]
$FIFOsize_{ID}$	[2,20]
th_{ID}	[0, 1.0]

Table 3: Intervals of the parameters for the sensitivity analysis.

Correlation table and plots. The Latin hypercube implementation in [45] generates a table of the values of correlation between inputs and outputs, as well as, a series of plots. The correlation values that are produced by the sensitivity analysis are shown in Table 4.

Parameter	**Accuracy**	**Precision**	**Recall**
$th_{training}$	0.344	-0.469	0.351
th_{online}	-0.00535	0.47	-0.00139
$ratio_{NaiveMature}$	-0.285	0.302	-0.291
$FIFOsize_{ID}$	-0.124	0.102	-0.122
th_{ID}	-0.602	0.599	-0.598

Table 4: Correlation coefficients between parameters and output measures.

The values in the table suggests that the parameters that most largely affect the performance of this implementation of the framework are th_{ID} and $th_{training}$. The influence of $ratio_{NaiveMature}$ is lower, while the effect of the variation of $FIFOsize_{ID}$ seems to be irrelevant. Concerning th_{online}, it is not clear whether this parameter affects the performance of the framework, as different correlation values are produced for the output measures.

The correlation table also shows that accuracy and recall have always the same sign and therefore are positively correlated between each other, while precision has always opposite sign with respect to the other two measures. An explanation of this phenomenon is presented in Section 4.2.1.2. That section shows that high values of accuracy and recall and low values of precision are associated with instances of the framework that can classify data with concept drift correctly. By contrast, an instance of the framework that cannot cope with the drift of the dataset would show high precision, but low accuracy and recall.

Each plot generated by the Latin hypercube sensitivity analysis depicts the two hundred samples in function of an input parameter and an output measure. In all the plots it is possible to identify two distinct clusters of points, one at the top and one at the bottom, as shown for the plots of th_{online} of Figure 15. The plots of the other variables are described in the Appendix C.1. This indicates that the framework can handle concept drift or it can fail, intermediate values are a minority. Such a characteristic can be explained by the fact that as long as the updating of the framework model is able to keep track of the concepts (the clusters) the performance remains high. However, if the updating of the model cannot keep the pace of the drifting data, the performance of the framework drops.

The sensitivity analysis reveals that the most influential parameters are the threshold of the FIFO queue, th_{ID}, that is used to infer drift, and the training threshold $th_{training}$. The first result is not surprising. In fact, the threshold th_{ID} determines the number of times in which the distance d_{MN} has to be higher than both d_M and d_N in the last $FIFOsize_{ID}$ iterations in order to infer drift, as described in Section 3.3.3. Therefore, the higher the value of th_{ID}, the stricter is the condition that is needed to infer drift.

We interpret the positive correlation between $th_{training}$ and accuracy and recall (negative in the case of precision) with the fact that higher values of $th_{training}$ correspond to smaller numbers of classifiers. An ensemble with fewer classifiers has a smaller "inertia" than one with many classifiers. In the sense that the replacement of a classifier has a larger effect on the model of the ensemble if the ensemble has a small size, therefore it takes fewer replacements to change the model with respect to a larger ensemble.

Although the correlations between th_{online} and accuracy and recall does not seem

significant, the correlation between the same measure and precision reveals that th_{online} may indeed affect the performance of the framework.

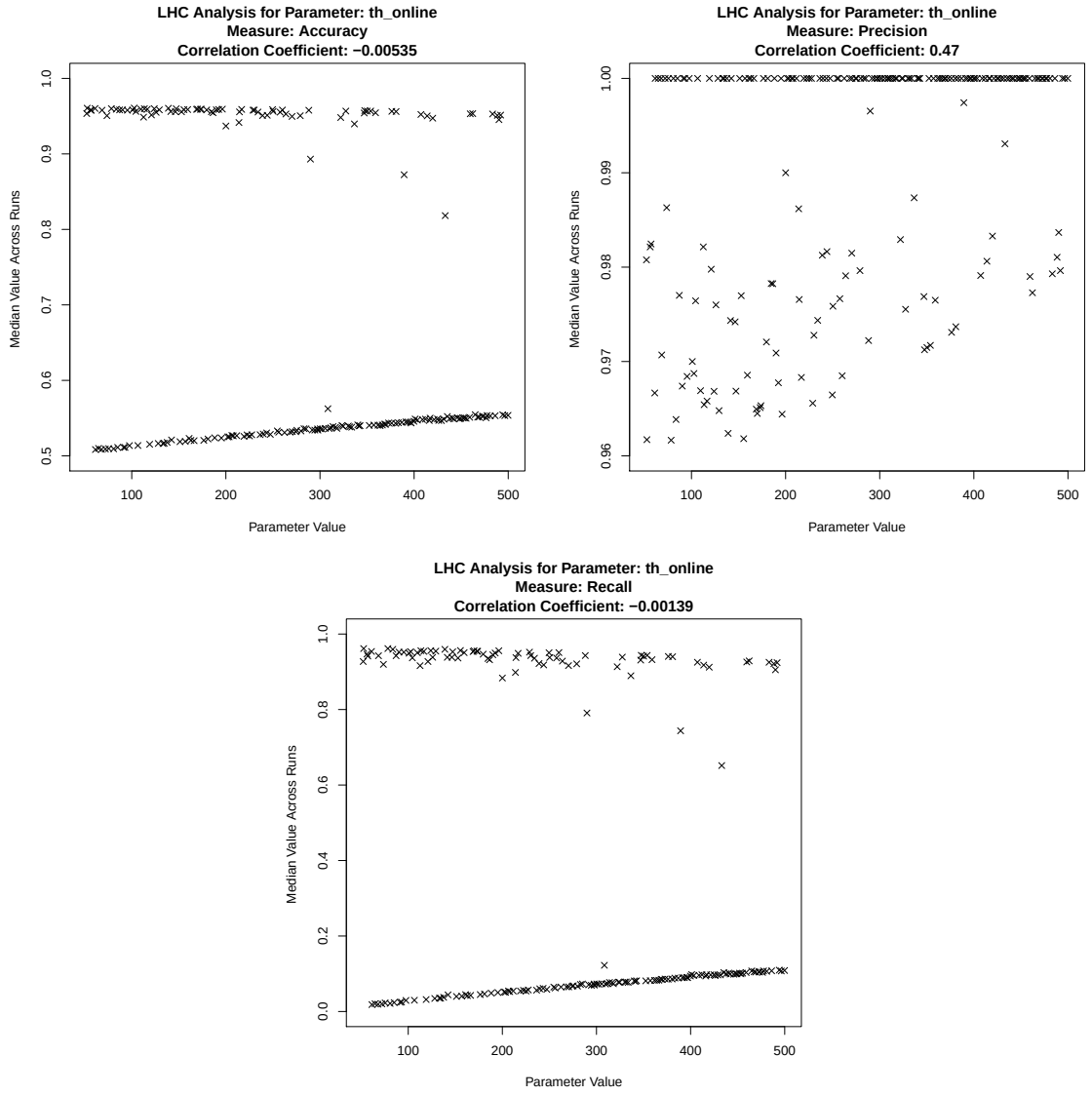


Figure 15: Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall.

A second look at the plot of accuracy and recall, highlights that the cluster of points at the bottom has a linear trend as it can be seen in Figure 15. We believe this gives a positive contribute to the correlation between accuracy and the output variable th_{online}, as well as the correlation between recall and the same variable. However, this phenomenon is probably caused by the fact that higher values of th_{online} cause the framework to stop before than it would do for smaller values. For this reason, the last accuracy (or recall) value that is recorded is higher for high

values of th_{online}, but that does not mean that the framework can deal with drift.

The parameter $ratio_{naiveMature}$ seems to have little influence on the performance of the framework, however it appears that small values cause higher performance. In addition, the low correlation between the parameter $FIFOsize_{ID}$ and the outputs indicates that that parameter does not have a significant influence on the framework performance. Therefore, varying the length of the FIFO queue does not affect the performance significantly.

The plots for the parameters $th_{training}$, $ratio_{naiveMature}$, $FIFOsize_{ID}$ and $ratio_{naiveMature}$ are shown in the Appendix C.1.

Interpretation of the sensitivity analysis. This section provides and explanation of the fact that precision is negatively correlated to accuracy and recall. For this purpose, we show that high values of accuracy and recall are associated with the framework being able to deal with concept drift, while high values of precision indicate that the framework cannot classify the data correctly.

In particular, we consider two samples, each associated with a configuration of the parameters of the framework, amongst the two hundred that were generated for the analysis of the sensitivity. When these are displayed in the plots of Figure 15, they belong to different clusters. Firstly, we examine a sample for which the framework maintains high performance in terms of accuracy and precision. Then, we consider a sample that generates higher precision than the first sample, but lower accuracy and recall. In order to explain this result we will show how TP, TN, FP and FN vary across a run.

For the first sample, the normalised number of TP and that of TN do not vary considerably across the iterations. Their values are respectively around 0.47 and 0.49. By contrast, the variations of FP and FN are more pronounced. In particular, the normalised number of FP, that right after the training phase has the median value of 0.02166, at as early as 10% of completion of a run is almost halved, as its median value is 0.01092. The number of FP remains low for the rest of the computation, and its value at the end of a run is 0.01189. The median value of FN after training of the model is 0.02017, which is comparable with the initial value of FP. However, FN and FP have different trends. In fact, the median value of FN increases right after training and reaches the value of 0.03521 at the end of the computation. The fluctuation of the values of FP and FN reflects on the measures

of precision and recall. To be more precise, the median values of recall decrease from the initial value of 0.96146, and they stabilise at around 0.93. By contrast, the precision increases from 0.95648 to 0.97. The values of accuracy, which are between 0.95 and 0.96, do not seem to vary much across a run.

We now consider a second set of parameters that generates lower accuracy and recall, but higher precision with respect to the first set. The initial values of TP, TN, FP and FN are similar to those observed for first sample. This means that both configurations perform well at the beginning of the testing phase. The situation changes as more unlabelled data is classified. In fact, we observed a noticeable decrement of the TP towards 0, and an increment of the normalised number of TN to about 0.5. Similarly, the number of FP, that is initially low, furthely decreases and tends to 0, while the FN tend to 0.5. These variations cause the value of accuracy to decrease to 0.5 and that of recall to decrease to 0. However, in this case, precision contrasts with the other two measures. In fact, its value at the end of the runs is around 1. This is similar to what was observed for the other configuration of the parameters.

Figure 16 provides an interpretation of why recall and accuracy decrease while precision increases, and therefore these measures are negatively correlated. The top of the figure, case (a), shows the Gaussian clusters at the beginning or a run (overlapped grey circles), the black line represents the boundary of the model of the framework.

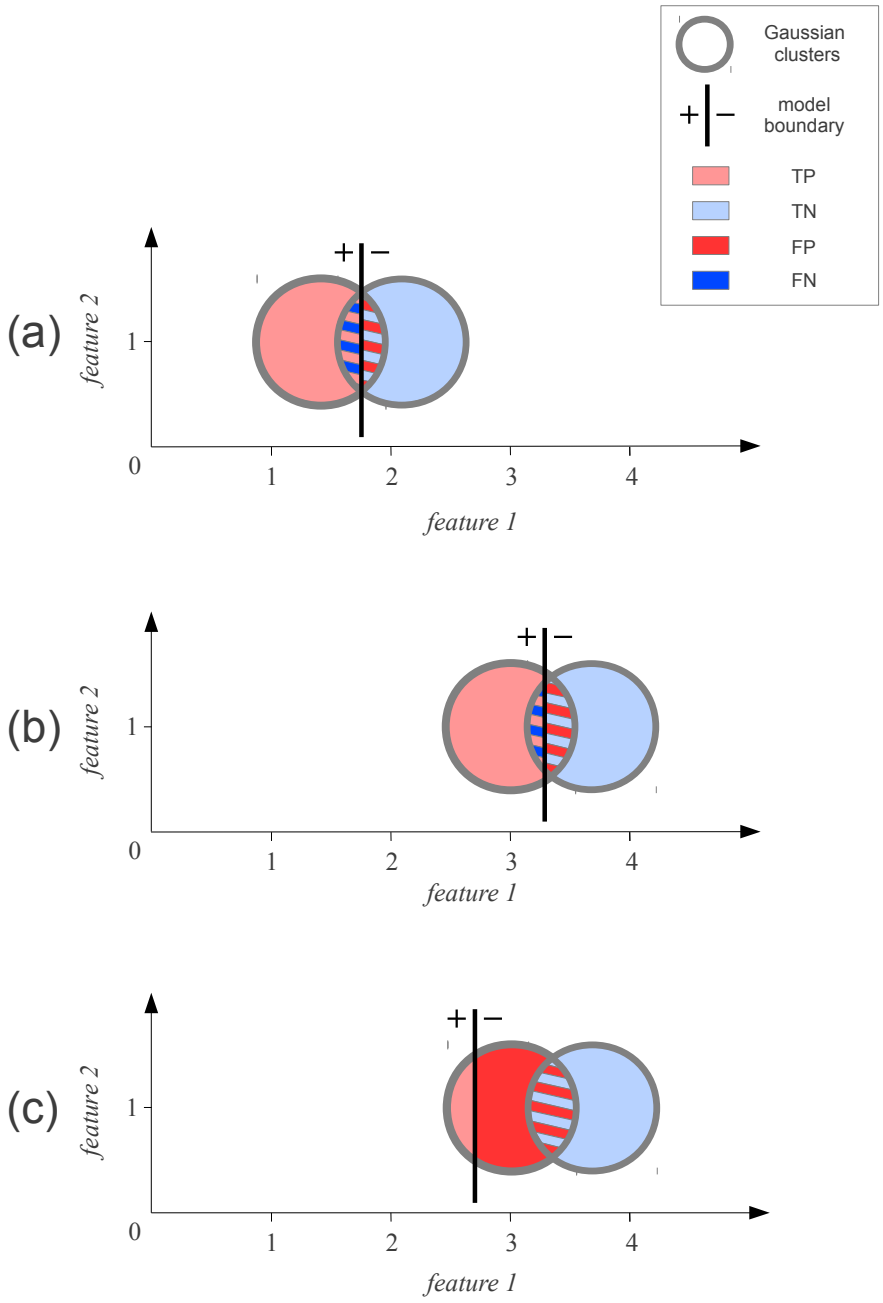


Figure 16: Representation of the model of the framework featuring the SVM at the beggining of a run(a), at the end of a successful run (b) and in case of failure (c).

The points which lie on the righthand side of the boundary are classified as positive, the points on the left are classified as negative. The boundary in this case is in an optimal position as it generates a minimal number of misclassified instances (the sum of FP and FN). Section 2.2.1 gave an explanation as to why this happens for univariate normal distributions. When concept drift is ongoing, the boundary of the model should be slightly shifted towards the left, as shown in the part (b)

of Figure 16. This might be due to a delay between the start of concept drift and this being inferred, and therefore the model being updated. This would explain the decrement of FP and the increment of FN that causes precision to increase and recall to decrease.

Part (c) of Figure 16 gives an interpretation of the performance of the framework with the second set of parameters. The figure depicts a run in which the shift of the boundary with respect to the center of the clusters is much larger than case (b). This would reduce the number of TP and increases the number of FN, while the number of TN is maximum as negative instances are all classified correctly and therefore there are no FP. This interpretation is compatible with the results of the second parametric configuration.

Some conclusions can be drawn from this analysis. The framework with the SVM is able to deal with the concept drift of this dataset if the values of the output measures do not disclose significantly with respect to their initial values. In fact, the position of the boundary of the model inbetween the two classes is optimal, which is also the initial position.

When the values of the parameters are varied, the ability of the framework to update its model changes, and this reflects on its performance. In particular, decrements of the accuracy and the recall are associated with increments of the precision, and viceversa. In addition, in this context, precision does not denote the ability of the framework to deal with concept drift as clearly as recall and accuracy do. In fact, there is little difference between the values of precision of a sample that is able to deal with drift and the values of precision of a sample that is not. This interpretation is confirmed by the plots of actual data shown in the Appendix D.

4.2.1.3 Comparison with supervised techniques

This section presents a comparison of the framework with a selection of supervised techniques. The purpose of this comparison is to highlight the consequences of not using supervision to update the model of a classifier. The supervised techniques are the Dynamic Weighted Majority (DWM) algorithm and bagging with ADWIN [8, 9, 53]. DWM was described in Section 2.4.1.4, bagging in Section 2.2.3 and ADWIN in Section 2.4.1.5.

Table 5 illustrates the parameters of the comparisons, while those of the framework and the SVM are determined by Table 2. The only parameter of bagging

with ADWIN is the size of the ensemble. In the case of DWM, β controls the penalisation that a classifier undergoes if it misclassifies an instance, γ avoids the size of the ensemble to grow. The third parameter of DWM dictates that every 1000 instances uneffective classifiers are deleted, potentially new classifiers are created and the weights of the classifiers are recomputed. The dataset is that with two Gaussian clusters illustrated in Section 4.2.1.1.

Technique	**Parameter**	**Value**
DWM	β	0.5
	γ	0.2
	period	1000
bagging + ADWIN	ensemble size	10

Table 5: Parametric configurations of DWM and bagging+ADWIN for the experiment involving SVM and Gaussian data.

This comparison reveals that there are little differences between the performances of the three techniques. Figure 17 compares the performances of the techniques. The values of accuracy are similar for the three techniques. The framework has the highest precision between the three, while bagging+ADWIN has the lowest one. Bagging with ADWIN has the best performance in terms of recall, while the recall of the framework is smaller than the other techniques. We presume that this is caused by a delay that affects the mechanism of drift inference.

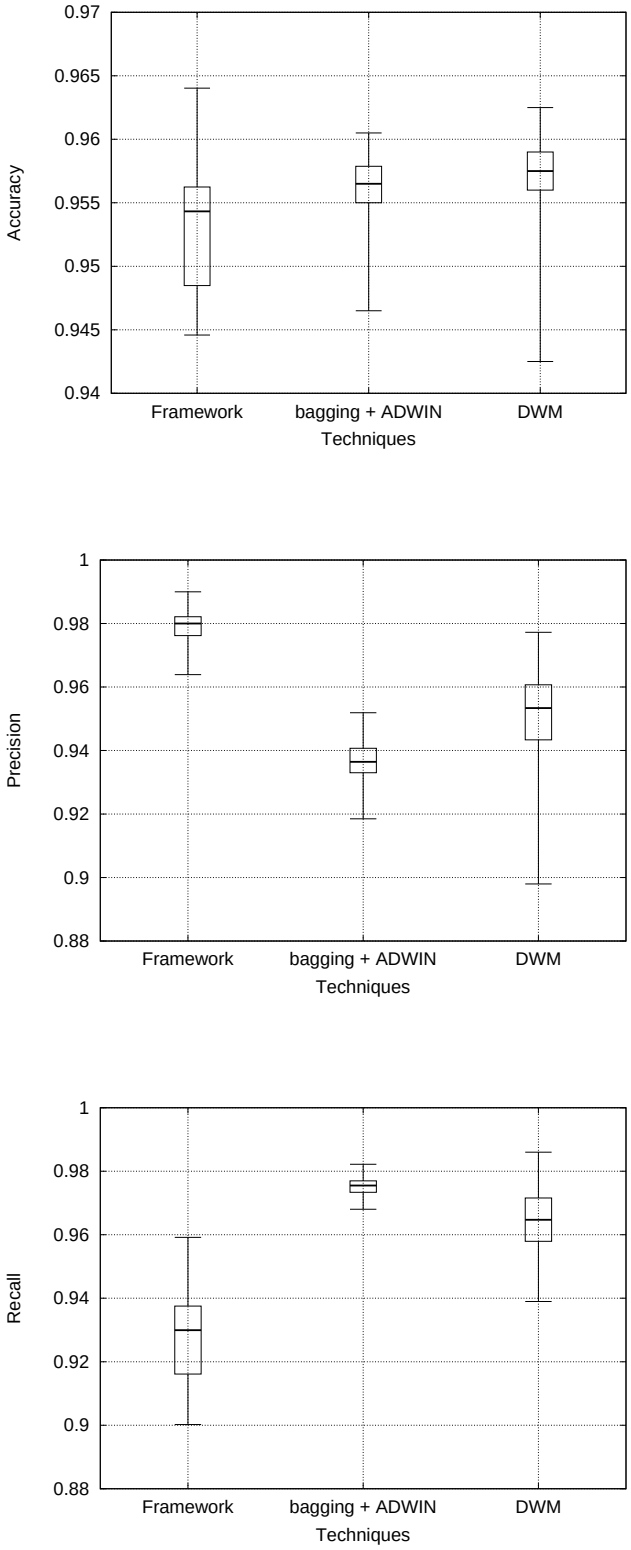


Figure 17: Distributions of the values of accuracy, precision and recall of the framework, of bagging+ADWIN, and of DWM over the Gaussian data stream. Notice that different scales are used.

4.2.1.4 Interpretation of the results

This experiment showed that it is possible to classify the Gaussian dataset without the need of supervision. However, it is to be understood what mechanism causes the framework to react to the concept drift. We propose an interpretation of the results. In order to simplify the explanation, let us suppose that the data contains only a single feature rather than two, while the number of classes and the type of distribution are left unvaried.

The explanation begins with the training of a model by means of the true classes of the data. An analysis of the outcomes of using labels from the voting to train new classifiers follows. Then, assuming that the ensemble contains three classifiers, the effect of the replacement of each of those is examined. Finally, the importance of the gradient of the distribution for the training of new classifiers is highlighted.

Training with supervised data. Figure 18 depicts $p(x, \text{“}+1\text{”})$ (red line), $p(x, \text{“}-1\text{”})$ (green line), and their sum $p(x)$ (dotted line). The supervised data that is initially provided is generated from $p(x, \text{“}+1\text{”})$ and $p(x, \text{“}-1\text{”})$. This data is used to establish the model.

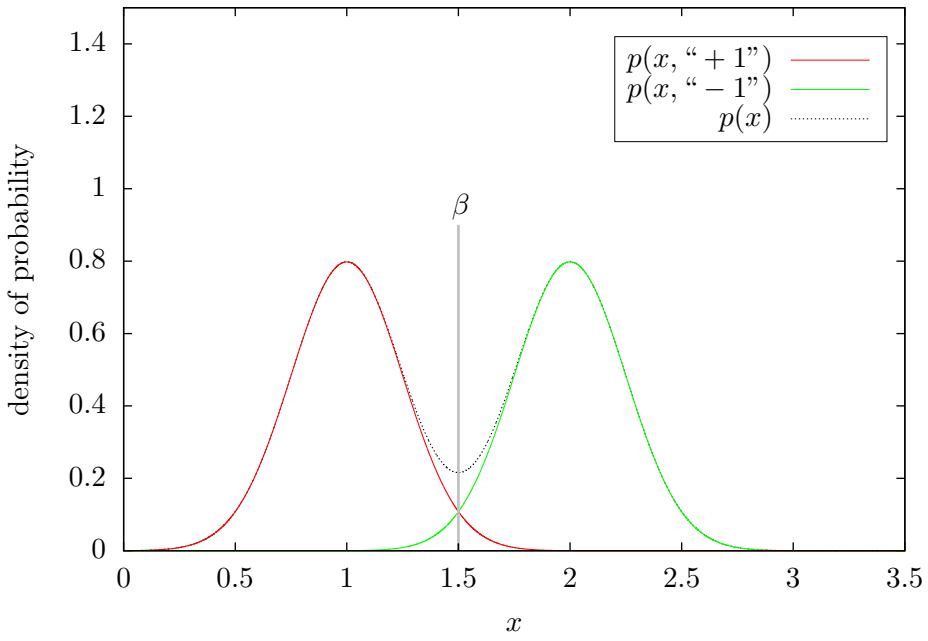


Figure 18: Data distribution of the supervised training phase. The green line and the red line represent respectively the distributions of the negative class and the positive class. Their sum, the class-unconditional distribution, is represented by the dotted line.

In Figure 18, the boundary of the model is represented as a continuous black line between the classes. The boundary is an optimal position, as it minimises the classification error according to the explanation given in Section 2.2.1.

Model updating using labels from the voting. When the online phase starts, the information conveyed by the unlabelled data only concerns the $p(x)$ of the data. In fact, since there is no supervision, information about the distributions of the single classes is no longer provided. According to the classification model, points on the left of the boundary are classified as "+1", while points on the right are classified as " $-$ 1", as shown in Figure 19. Unlabelled instances and the labels associated with them through voting are used to train a new classifier. We might consider this data as if it was generated from the distribution $p(x) = p'(x, \text{“}+1\text{”}) + p'(x, \text{“}-1\text{”})$, where $p'(x, \text{“}+1\text{”})$ and $p'(x, \text{“}-1\text{”})$" are as depicted in Figure 19.

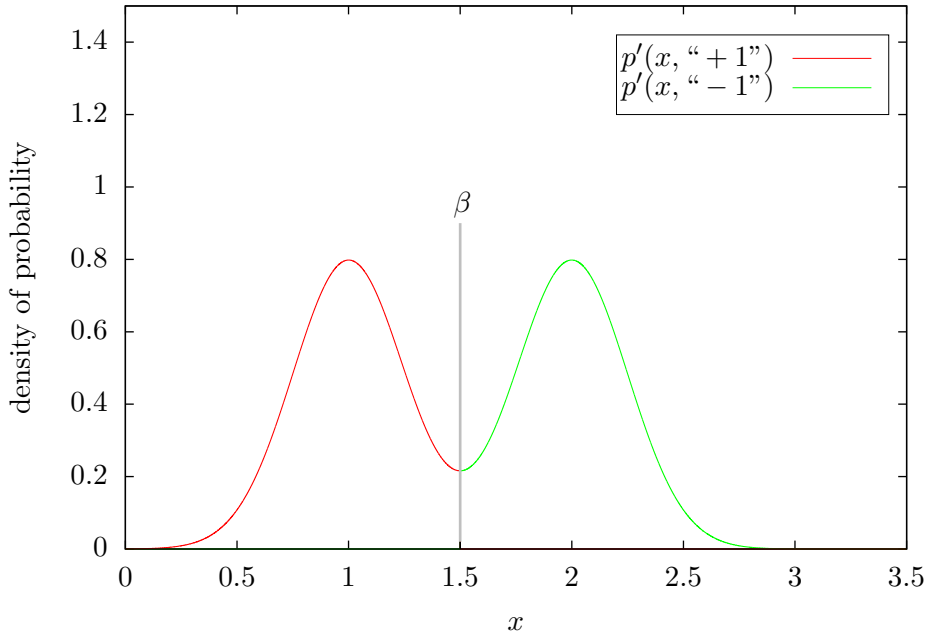


Figure 19: In the online phase, new naïve classifiers are trained using labels from the voting. Therefore, the distribution of the data and their labels can be described by $p'(x, \text{“}+1\text{”})$ and $p'(x, \text{“}-1\text{”})$".

Effect of the replacement of the different classifiers of the ensemble. Let us now suppose that, after several iterations, the distribution has drifted but the model has not changed so that its boundary is still in the same position as when the online phase started. Figure 20 depicts this scenario.

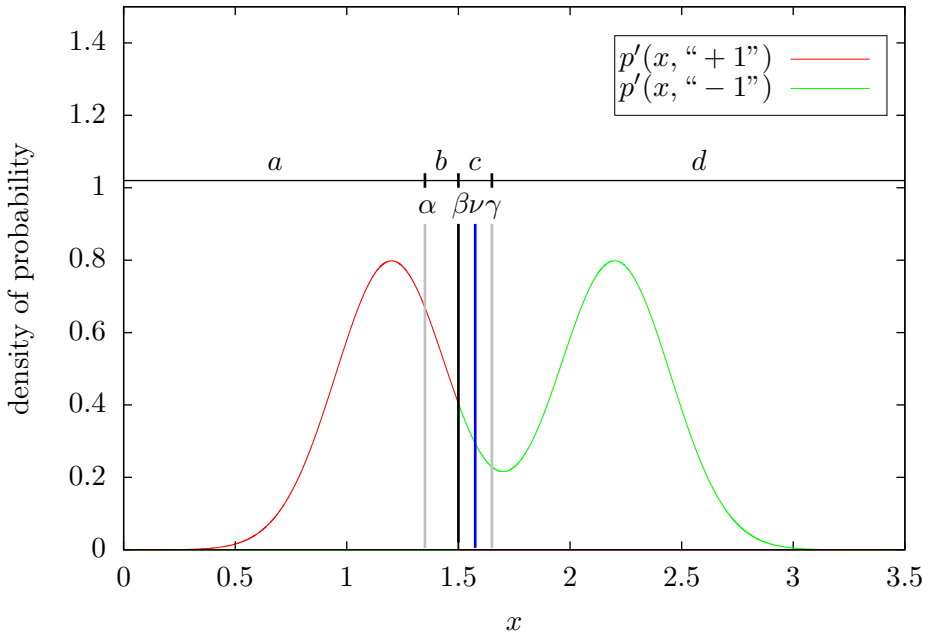


Figure 20: Probability density distributions of class “$+1$” (red line), “-1” (green line) in the online phase. The grey lines and the black line represent the boundaries of the three classifiers α, β, γ, while the blue line is the boundary of the naïve classifier ν. In particular, the boundary of β conincides with the boundary of the model of the framework.

Let us also suppose that the framework contains three mature classifiers α, β, γ (grey lines and black line), a naïve classifier ν (blue line) and the mechanism of drift inference has been triggered. The boundaries of the three classifiers divide the input space into four regions, in the diagram they are labelled as “a”, “b”, “c” and “d”.

The points lying in the region “a” are classified unanimously by the classifiers as belonging to class “$+1$”. The points that fall into the region “b” are classified as “-1” by the classifier α, while the classifiers β and γ disagree with α by assigning the class “$+1$” to those points, and so does the majority voting. Similarly, in the region “c” two classifiers out of three (α and β) label incoming points as “-1”, and all the classifiers agree that points in region “d” belong to class “-1”. Therefore, we can say that the boundary of the model of the framework corresponds to the central boundary (of classifier) β, for this reason it is highlighted using a black line.

We now analyse how the boundary of the framework model is affected when the naïve classifier ν replaces one of the mature classifiers. Figure 21 despicts these three possibilities, whether it is α, β or γ the classifier to be replaced. If the classifier ν

replaces α, its boundary will be in a central position and the boundary of the (model of the) framework will coincide with it, as shown in Figure 21 (top plot). The same result is obtained if ν replaces β, in fact, also in this case ν becomes the "central" classifier, Figure 21 (central plot). If ν replaces γ, the boundary of the model of the framework does not change with respect to the initial case, since the boundary of β continues to be in a central position (bottom plot of Figure 21). In two cases out of three the boundary of the framework has shifted to the right, which is also the direction of drift. In one case the boundary has not changed.

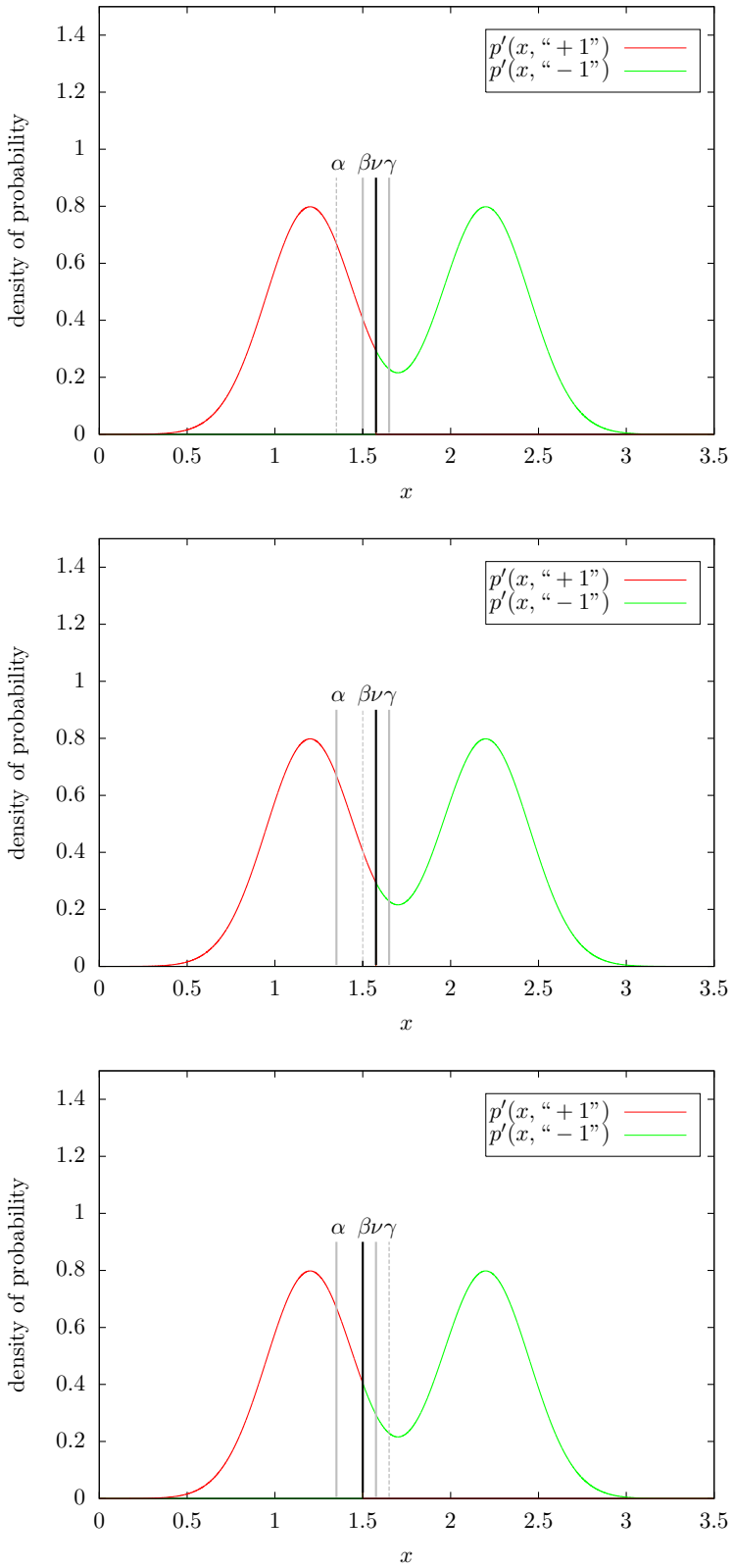


Figure 21: Effect of the replacement of the naïve classifier ν with the mature classifiers α (top), β (middle), γ (bottom) on the boundary of the model of the framework (black line). The dotted grey line indicates a classifier that has been deleted.

Effect of the gradient on the training of new classifiers. In order to describe this example, we have implicitely assumed that the boundary of the naïve model ν is on right with respect to the boundary of the model of the framework.

If the boundary of the framework model is in a position where $p(x)$ has a gradient, this is more likely to happen than the opposite case, in which the boundary of ν is on the right of the boundary β.

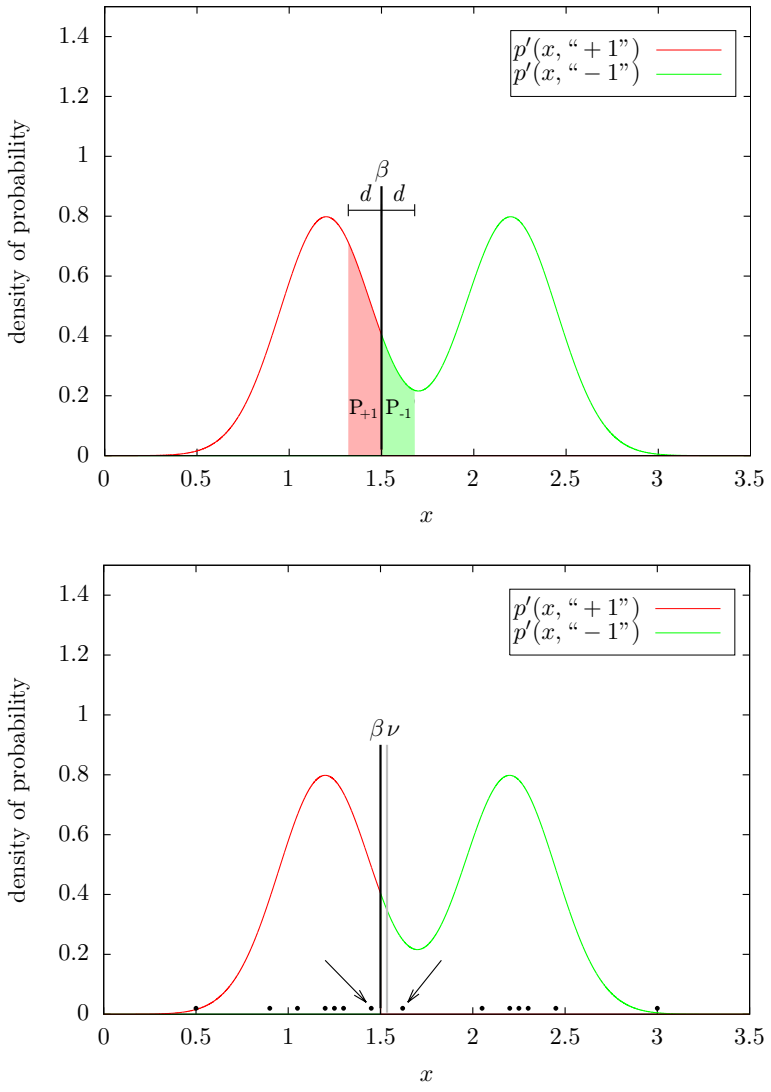


Figure 22: Probability of points to lie within a distance "d" from the boundary β is higher for points classified as "$+1$" (top). The diagram at the bottom shows that, when enough data has been classified, the same data is used to train a new naïve classifier ν', which is shifted to the left with respect to β.

In fact, if we consider an interval $[\beta - d, \beta + d]$ around the boundary of the framework β, the probability P_{+1} of a point being sampled from $[\beta - d, \beta]$ is higher

than the probability P_{-1} of a point being sampled from $[\beta, \beta + d]$. The probabilities P_{+1} and P_{-1} are the areas under the curve $p(x)$. They are highlighted in top plot Figure 22, which shows that P_{+1} is greater than P_{-1}. This means that when a batch of data is collected during the online phase, with high probability the distance of the closest point labelled as "$+1$" from the boundary of the framework β is smaller than the distance of the point labelled as "-1" that is closest to β. This is shown in Figure bottom plot 22. The same data and the labels from the voting process are used to train a new classifier ν'. The points indicated by the arrows are the support vectors, notice that the one on the "$+1$" side is closer to β than the one on the "-1" side. Since the SVM maximise the distance from the support vectors, the new classifier that is trained on these data will be shifted to the right with respect to β.

Summary. Initially the model is trained using information from $p(x, \text{“}+1\text{”})$ and $p(x, \text{“}-1\text{”})$. After that, the only information about the changing distribution is provided by $p(x)$. The replacement of a classifier of the ensemble is likely to cause the shift of the boundary of the framework towards the local minimum of $p(x)$, inbetween its two maxima. This is based on the assumption that the naïve classifier that replaces a mature classifier is closer to that region than the boundary of the model is. Although that is not always the case, it is more likely to happen than the opposite case. This is due to the gradient of the distribution $p(x)$, which causes points on the side with higher density to be closer to the boundary of the model than the points on the less dense side. Since the SVM classifier base their operation on the maximisation the distance between points of different classes, as a result new naïve classifiers will be shifted towards the local minimum of $p(x)$ with respect to the original boundary. In this context, the local minimum is also the point that provides maximum separation and therefore highest classification performance.

4.2.2 Uniformly distributed data

The experiment that is described in this section involves the use of uniformly distributed data. According to the considerations that were expressed in Section 3.2, we would not expect the framework to be able to perform well on this type of data. The distribution accounts for bivariate uniform data within the interval $[0, 4]$ of the first feature and the interval $[0, 2]$ of the second feature. Points that lie within

a circle whose center is initially in the point $(1, 1)$ and diameter equal to 1, belong to the class "$+1$". Points that lie outside of that circle belong to class "-1". Concept drift is simulated by changing the position of the circle from the initial coordinates of $(1, 1)$ to the final position of $(3, 1)$. Similar to the dataset containing Gaussian data, the number of training instances is $3,000$, while the number of unlabelled instances is $2,000,000$. A representation of the data distribution at the beginning of the experiment (a) and at the end (b) is depicted in Figure 23.

The classification technique that is used is the SVM and the parametric setting is described in Table 2.

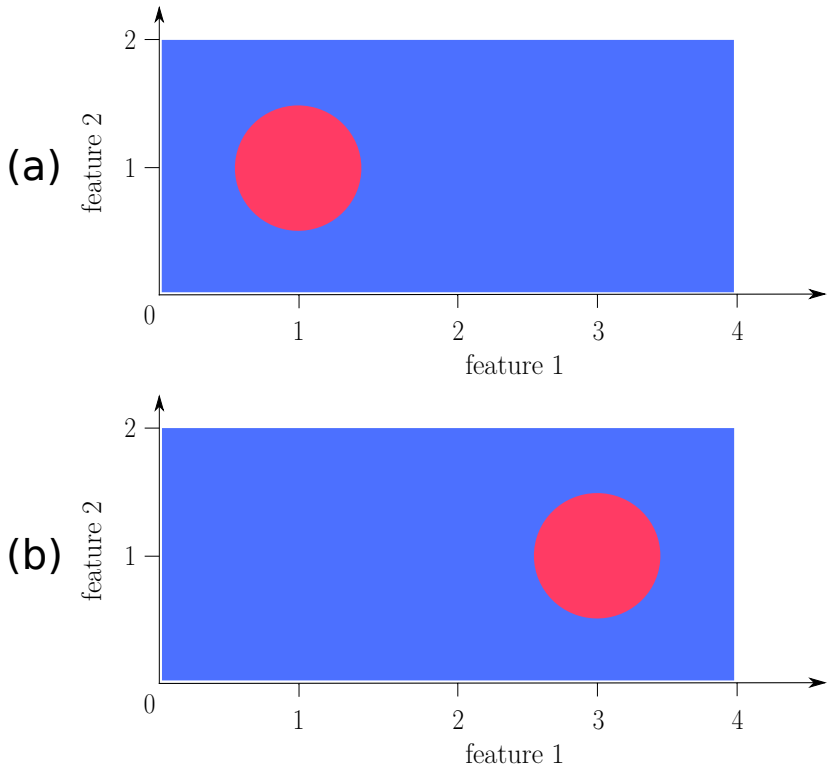


Figure 23: Representation of the uniform distribution at the beginning of the experiment (a) and at the end (b). The position of the positive class has shifted along the x axis.

The results of the A-test shows that repeating the experiment 50 times is sufficient, these are shown in Figure 24. In fact, although the A-test score for accuracy is still large for fifty runs, accuracy shows only marginally how the framework is actually performing on this distribution.

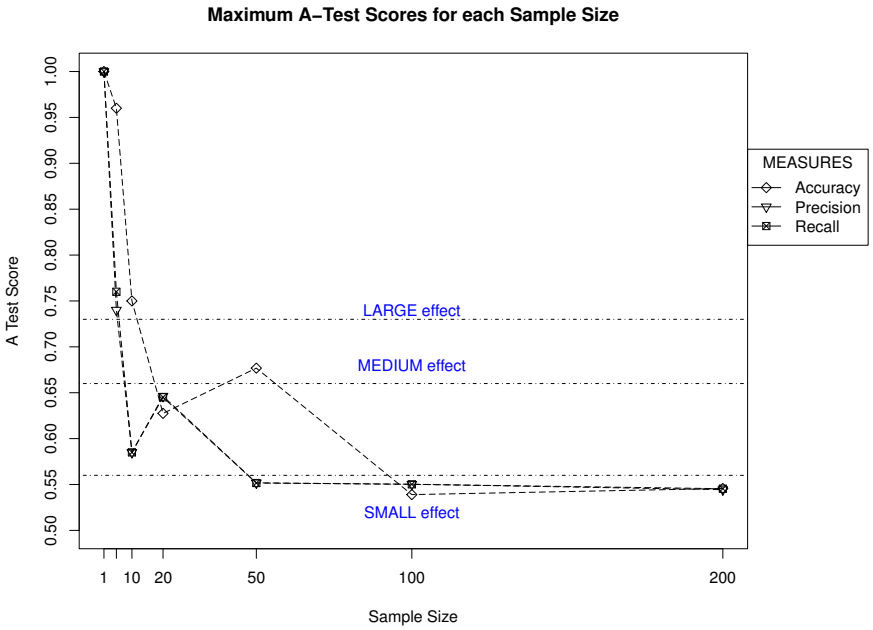


Figure 24: A-test results. Small scores are obtained for 100 runs. However precision and recall, which are more significant than accuracy in this concetxt, are sufficiently low for 50 runs.

The null hypothesis is formulated as:

> *The performance of an instance of the adaptive framework featuring the SVM, when tested on the uniform dataset, is not statistically different from the performance of an instance of the framework without adaptivity, that is, with a static model.*

The Mann-Whitney nonparametric test determined that the probabilities of the values of the adaptive instance and the values of the static instance being generated from the same distribution are: 7.803E-12 for accuracy, 1 for precision and 1 for recall. Therefore the null hypothesis cannot be rejected with confidence higher than 0.995. As a matter of fact, all the final values of precision for the adaptive and the static instance of the framework equal 0. The same is observed for the values of recall.

In particular, Figure 25 shows the plot of the values of accuracy of the adaptive instance and the static instance of the framework. Although the two distributions of values are different, the values of the adaptive framework are not higher than 0.85 for any of the runs.

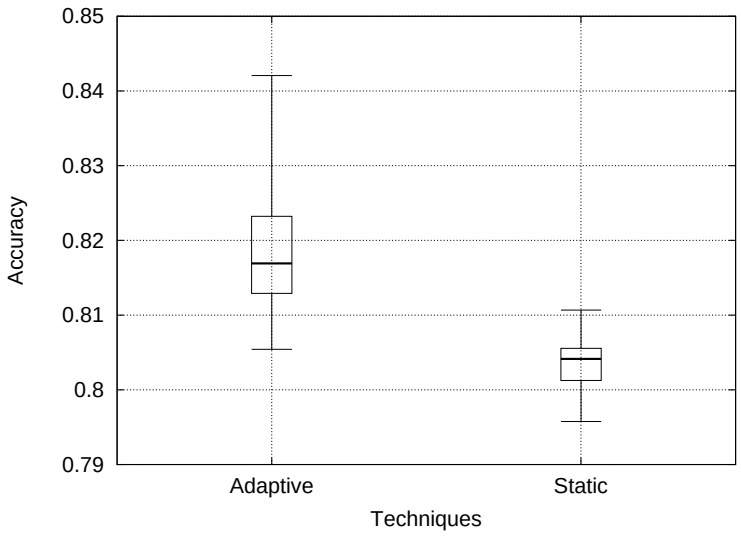


Figure 25: Comparison of the values of accuracy of the adaptive implementation and the static implentation of the framework using the SVM on the dataset with uniform distribution.

4.2.2.1 Sensitivity Analysis

Perhaps, the rejection of the hypothesis was caused by the parametric configuration that was used. A sensitivity analysis would clarify if alternative configurations allow the framework to deal with the concept drift of this dataset. The number of sampling points that were generated is 200, and 50 runs were performed for each sample. The plot of the parameter th_{online} shows that no sample combines high values of precision and recall. Therefore, we can draw the conclusion that the framework cannot deal with the concept drift of the uniformly distributed dataset. Additional plots are provided in the Appendix C.3.

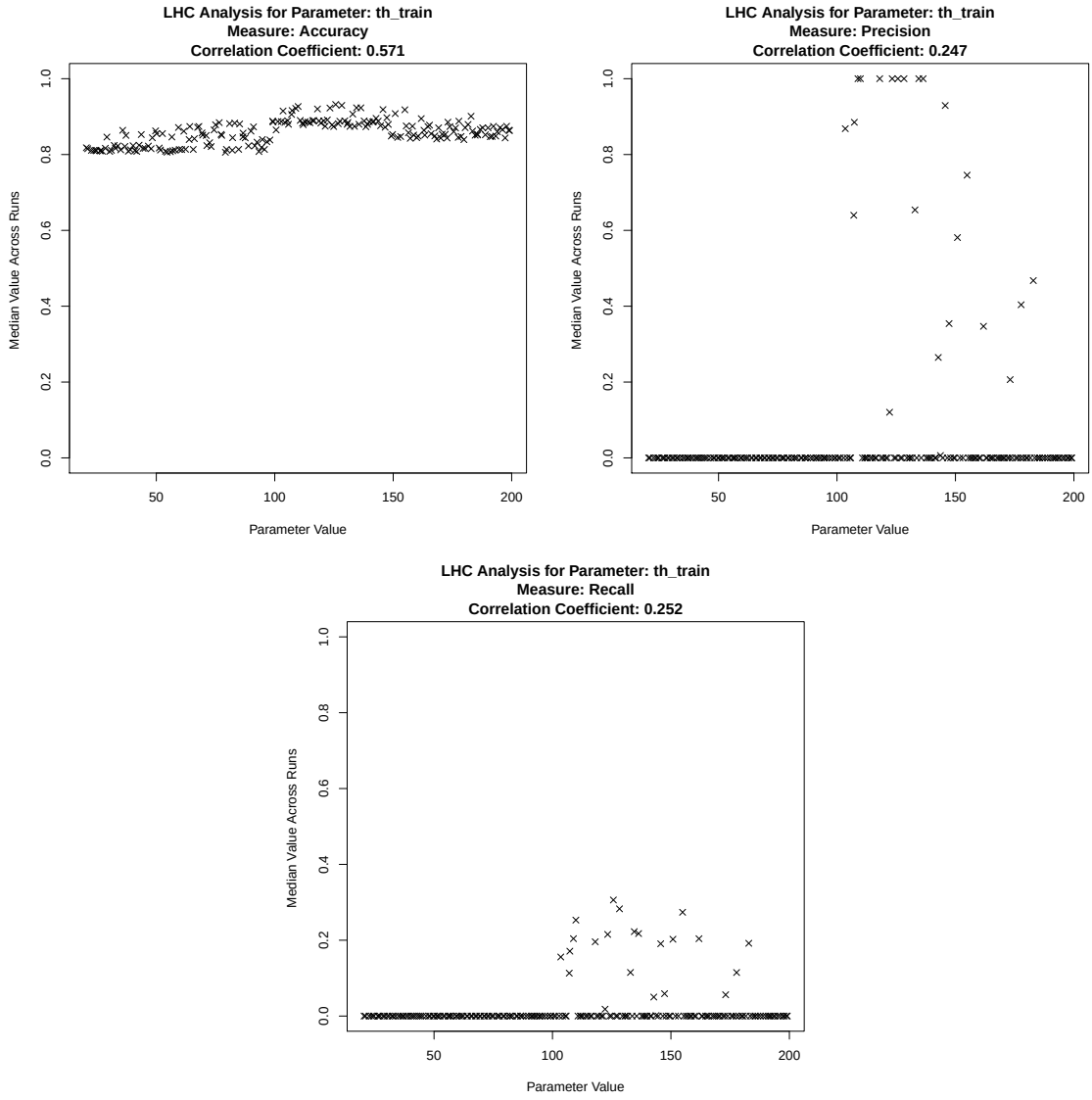


Figure 26: Latin hypercube sensitivity analysis plot of $th_{training}$ against accuracy, precision and recall.

4.2.2.2 Comparison with supervised techniques

A comparison with supervised techniques reveals the importance of supervision for this type of distributions. A selection of supervised algorithms was tested on exactly the same data that was used to test the framework. These are DWM and Bagging+ADWIN, the same techniques tested on the Gaussian data. The same data that was used to test the hypothesis is used for this comparison.

The values of the measures at the end of the runs are considered, rather than the median values. From the results, shown in the Figure 27, the difference between the performances of the supervised technique and the performance of the framework

is evident. The box and whisker plots show the distribution of the median values of accuracy, precision and recall across 50 runs. Although the accuracy of the framework may look high (around 0.85), because of the imbalance of the classes, even a classifier that assigns negative class to all the instance would perform with high accuracy on this dataset. By contrast, the three output measures of the supervised techniques of DWM and baggin+ADWIN are close to 1. In particular, baggin+ADWIN performs slightly better than DWM on this dataset.

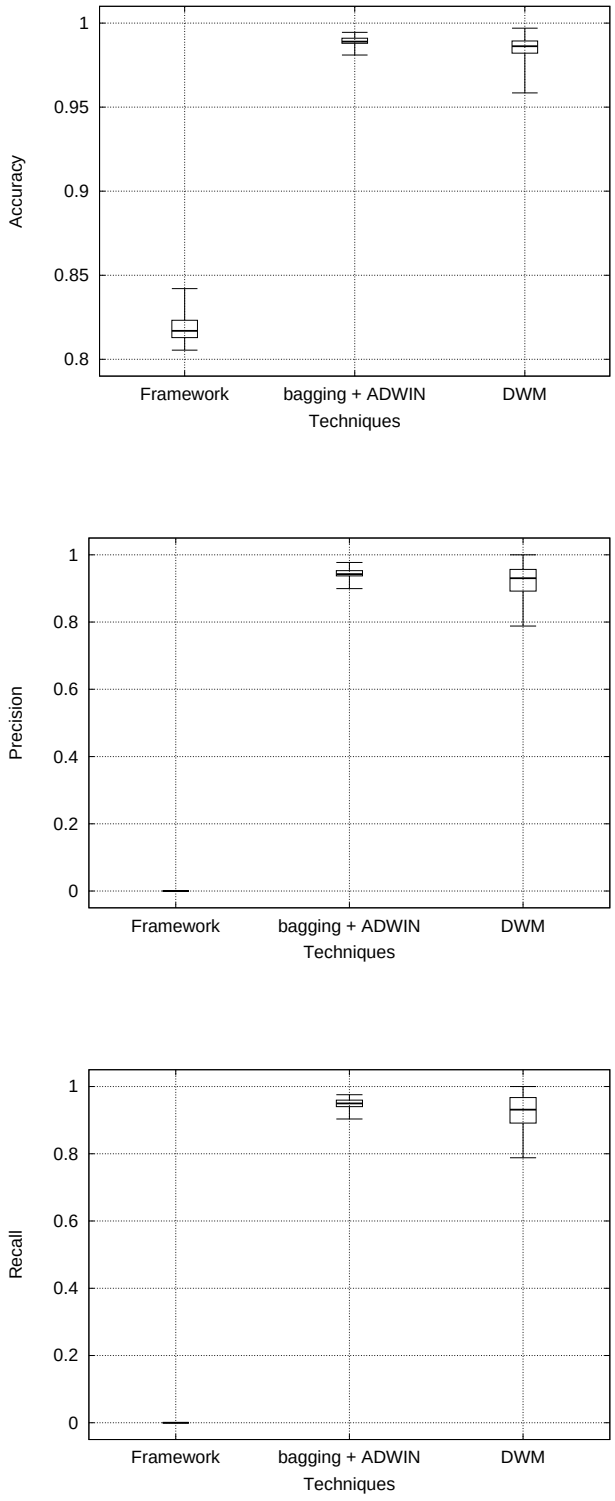


Figure 27: Distributions of the values of accuracy, precision and recall of the framework, of bagging+ADWIN, and of DWM over the data with uniform distribution. The values of recall and precision of the framework indicate that it is not able to deal with the concept drift of this dataset. Notice that different scales are used.

4.2.3 Conclusions

The objective of this investigation was to highlight potential differences in the capability of the framework of dealing with datasets with different distributions. In particular, the $p(x)$ of the dataset presented in Section 4.2.1 changes when drift starts, while the $p(x)$ of the dataset of Section 4.2.2 is not affected by concept drift. Although it is not possible to try every possible dataset, the results that were obtained seem to confirm the considerations expressed in Section 3.2. In fact, while the framework is able to classify the data with a changing $p(x)$ with good performance, it is not able to deal with the uniform distribution and fixed $p(x)$ of the second dataset.

4.3 Different classification techniques

The experiments in Section 4.2.1 showed that, under certain conditions, it is possible to classify drifting data without supervision. It is to be verified if this ability is maintained if such conditions change. For instance, it might be the case that the mechanisms of the implementation of the framework are effective only on SVM classifiers. For this reason, this section shows the results of running the framework on techniques with different characteristics from the SVM. The techniques on which the framework is tested are AISEC, Naive Bayes, the MLP and C4.5. They were described in Section 2.2.2.2. Although AISEC is a continuous learning techniques, in this context the model will not be updated after an initial training. The characteristic that makes AISEC an interesting testbed for the framework lies in in the way the model is structured. In fact, this consists of a set of detectors, a very different strategy with respect to the SVM, which uses a discriminant function. Naïve Bayes uses a generative approach to classification, different from the discriminative approach of the other techniques. The MLP is a classification technique that generates nonlinear discriminant function by combining linear functions. Our experiments utilise MLP classifiers with a single internal node, which generate linear discriminant functions. C4.5 is an algorithm that trains incremental decision trees.

4.3.1 Classification of Gaussian data with AISEC

This experiment assesses the performance of the framework with AISEC. A description of AISEC was given in Section 2.2.2.2. The data has the Gaussian distribution

described in Section 4.2.1.

The parameters of AISEC are configured according to Table 6. An explanation of the role of the parameters in the AISEC algorithm is contained in Section 2.2.2.2.

	Parameter	**Value**
framework	$th_{training}$	100
	th_{online}	400
	$ratio_{NaiveMature}$	1.0
	$FIFOsize_{ID}$	10
	th_{ID}	0.4
AISEC	CT	0.95
	AT	0.95
	CC	10.0
	MC	0.5
	NCS	125
	MCS	25
	IMC	100

Table 6: Configuration of the parameters of AISEC.

Based on the parameters of Table 6, the A-test was used to determine how many times an experiment has to be repeated in order for the distribution of the results not to vary significantly. The A-test was described in Section 4.2.1.1. However, since completing a single run takes from several hours to up to more than a day (while for the SVM the execution time was of the order of a few dozens of minutes), the test could not be extended to populations larger than 30 runs. However, for such value the A-test score seems to start converging. In fact, the scores for accuracy, precision and recall lie in the regions of medium effect and small effect, as shown in Figure 28.

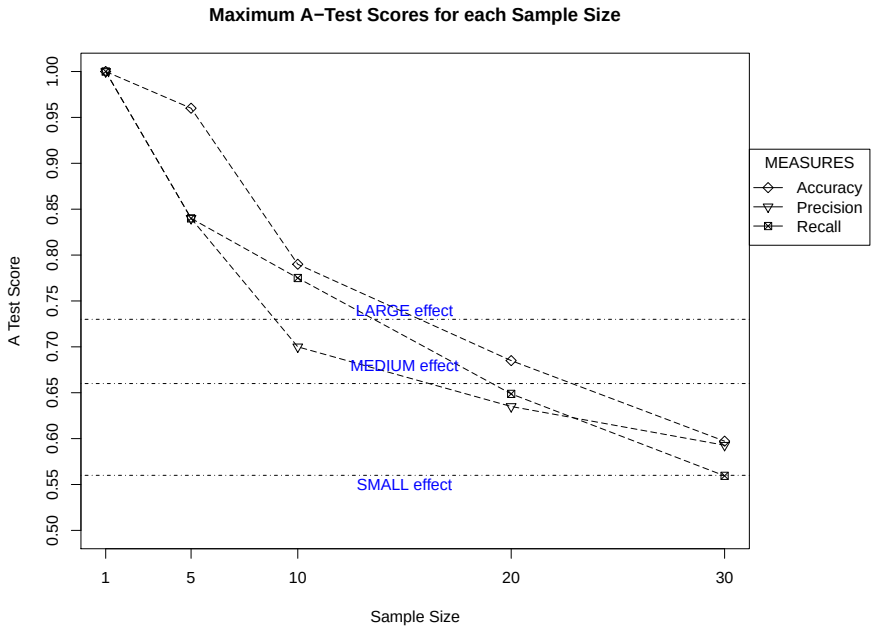


Figure 28: A-test scores for the AISEC algorithm. The graph shows that for 30 runs, the A-test score is sufficiently small.

The ability of the framework to classify concept drift is verified by comparing the values of accuracy, precision, recall generated by an adaptive instance of the framework with those of a static implementation, in which the model cannot be updated. The null hypothesis is formulated as:

> *The performance of an instance of the adaptive framework featuring AISEC, when tested on the Gaussian dataset, is not statistically different from the performance of an instance of the framework without adaptivity.*

When the median values are used, the value of accuracy and recall do not allow for the rejection of the hypothesis with a confidence level of 0.995. Figure 29 shows the distribution of the values of accuracy of the two implementations across a run. The top plot is generated using the runs of the adaptive implementation, while the bottom plot using the runs of the static implementation. The figure shows that the values of accuracy of the two implementations at 50% of completion of a run are very similar.

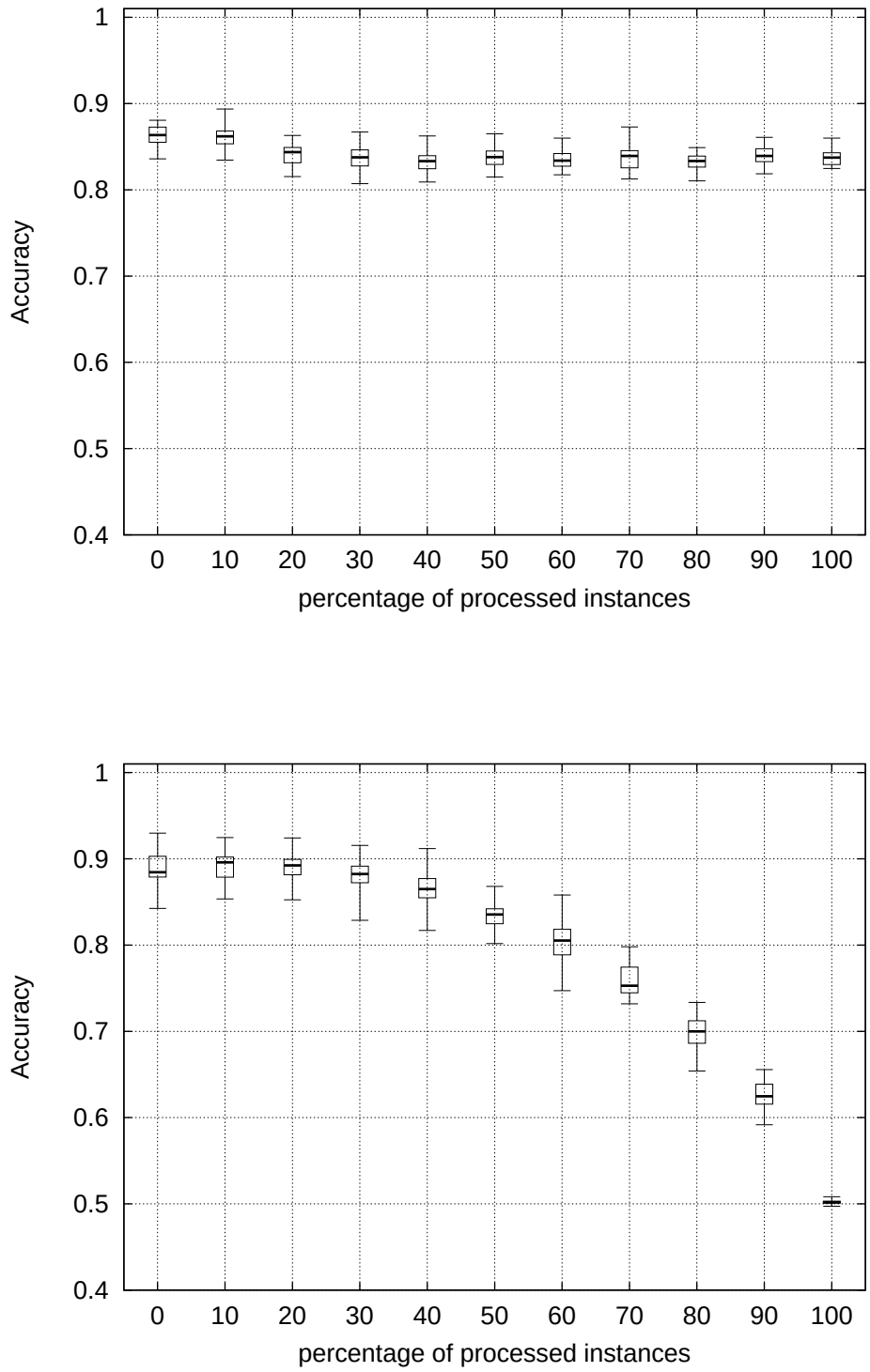


Figure 29: Values of accuracy in function of the percentage of completion of a run, for the dynamic implementation (top) and the static one (bottom). In correspondence of 50%, the distributions are very similar.

In fact, the median values of the static runs are quite high (around 0.85). However, this does not mean that the static instance is able to classify concept drift. The problem is that a static instance stops processing the data stream, before this is finished, and may be related to the parameter th_{online}, that dictates that a certain number of instances classified as "$+1$" and the same number of instaces classified as "-1" need to be collected in order to train a new classifier. Since the distribution drifts, at some point, only instances classified as "$+1$" will be available and the framework will stop updating its model. The performance degrades quite quickly, however the median remains high.

For this reason we decided to tested the hypothesis with value recorded at different stages of a run. The results are shown in Table 7.

Percentage of completion	**p-value of accuracy**	**p-value of precision**	**p-value of recall**
0	1.72941E-07	0.01998	4.61336E-10
30	5.57265E-10	1.72087E-05	6.68742E-11
50	0.44201	0.38256	0.40768
70	3.01986E-11	0.00026	3.01607E-11
100	3.01986E-11	4.81892E-09	3.01230E-11

Table 7: p-values at different stages of completion of a run.

When the initial values are considered, the probability of the two sets of values being generated from the same distribution is low for accuracy and recall, but not for precision. The three probabilities decrease at 30% of completion of a run, while at 50%, these probabilities reach their highest values high. The lowest p-values are obtained at the end of a run. As a matter of fact, in Figure 29, the accuracies of the adaptive and the static implementation clearly tend to different values. Around 0.85, for the adaptive implementation and around 0.5 for the static one. We suggest that the values at the end of a run indicate more clearly the ability of the framework to deal with drift. In correspondence of those values, the hypothesis can be rejected with a confidence level of 0.995.

4.3.1.1 Sensitivity analysis

The analysis of the sensitivity of a system requires large numbers of runs to be performed. In fact, for each sampling point, multiple runs are needed. This number

is dictated by the A-test. For example, for the sensitivity analysis of the framework with the SVM (Section 4.2.1), ten thousand runs were performed.

Running thousands or tens of thousands of runs, as it was done for the SVM, is not feasible in the case of AISEC. This encouraged us to find an alternative solution. In particular, a single run per sample was performed and a dummy variable is added to the existing parameters of the framework. Since the dummy parameter does not affect the performance of the framework, if enough sampling points are generated, we would expect the correlation between the dummy parameter and the output measures to be low. The number of points that are generated through the Latin hypercube sampling is 300, which is higher than the number of samples that were generated to test the framework with SVM classifiers. This is to compensate for the fewer runs performed for each sample. The parameters of AISEC are listed in Table 6, while the ranges of the parameters of the framework are listed in Table 8. There are some differences with respect to the ranges used for the SVM, the parameter IMC of AISEC poses in fact constraints to the values of the parameters $th_{training}$ and Th_{online} of the framework.

Parameter	**Interval**
$th_{training}$	[100, 480]
Th_{online}	[100, 500]
$ratio_{NaiveMature}$	[0, 1.0]
$FIFOsize_{ID}$	[2,20]
th_{ID}	[0, 1.0]

Table 8: Intervals of the paramaters for the sensitivity analysis.

Table 9 shows the correlation values of the framework embodying AISEC. The number of 300 samples that was chosen for this sensitivity analysis seems to be sufficient. As a matter of fact, the correlation between the dummy parameter and the outputs is sufficiently low, as it is always smaller than 0.15. Recall generates the highest correlation value. That table clearly indicates that the framework with AISEC performs in a very different way than it does when the SVM are used. A characteristic that catches the eye is that accuracy, precision and recall always have the same sign. In fact, except for th_{ID}, a variation of one of these input variables causes a similar response of the three output measures. By contrast, precision had always a different trend than accuracy and recall for the framework with the SVM.

Parameter	Accuracy	Precision	Recall
$th_{training}$	-0.593	-0.59	-0.411
Th_{online}	0.238	0.118	0.264
$ratio_{NaiveMature}$	0.645	0.651	0.464
$FIFOsize_{ID}$	-0.0507	-0.0147	-0.03
th_{ID}	0.0516	0.378	-0.222
dummy parameter	-0.0952	-0.0194	-0.144

Table 9: Correlation coefficients between parameters and output measures.

The parameters that affect most largely the performance of the framework are $th_{training}$ and $ratio_{naiveMature}$. In particular, the higher the number of naïve classifiers, the better is the performance. This contrasts with the results for the SVM classifiers, for which low numbers of naïve classifiers provided better results. In Figure 30, the plots of accuracy and recall contain clusters on the top right corners and the bottom left corners. The fact that the clusters are so neatly distinct may be related to the fact that a higher number of naïve classifiers improves the mechanism of drift inference. In fact, this is based on the measurement of the discordances between mature and naïve classifiers. The plot of recall, highlights how narrow the range of values is. In fact, while in all the accuracy and precision plots it is possible to distinguish clearly two distinct clusters, one at the bottom and one at the top, the recall plots display a single homogeneous cluster.

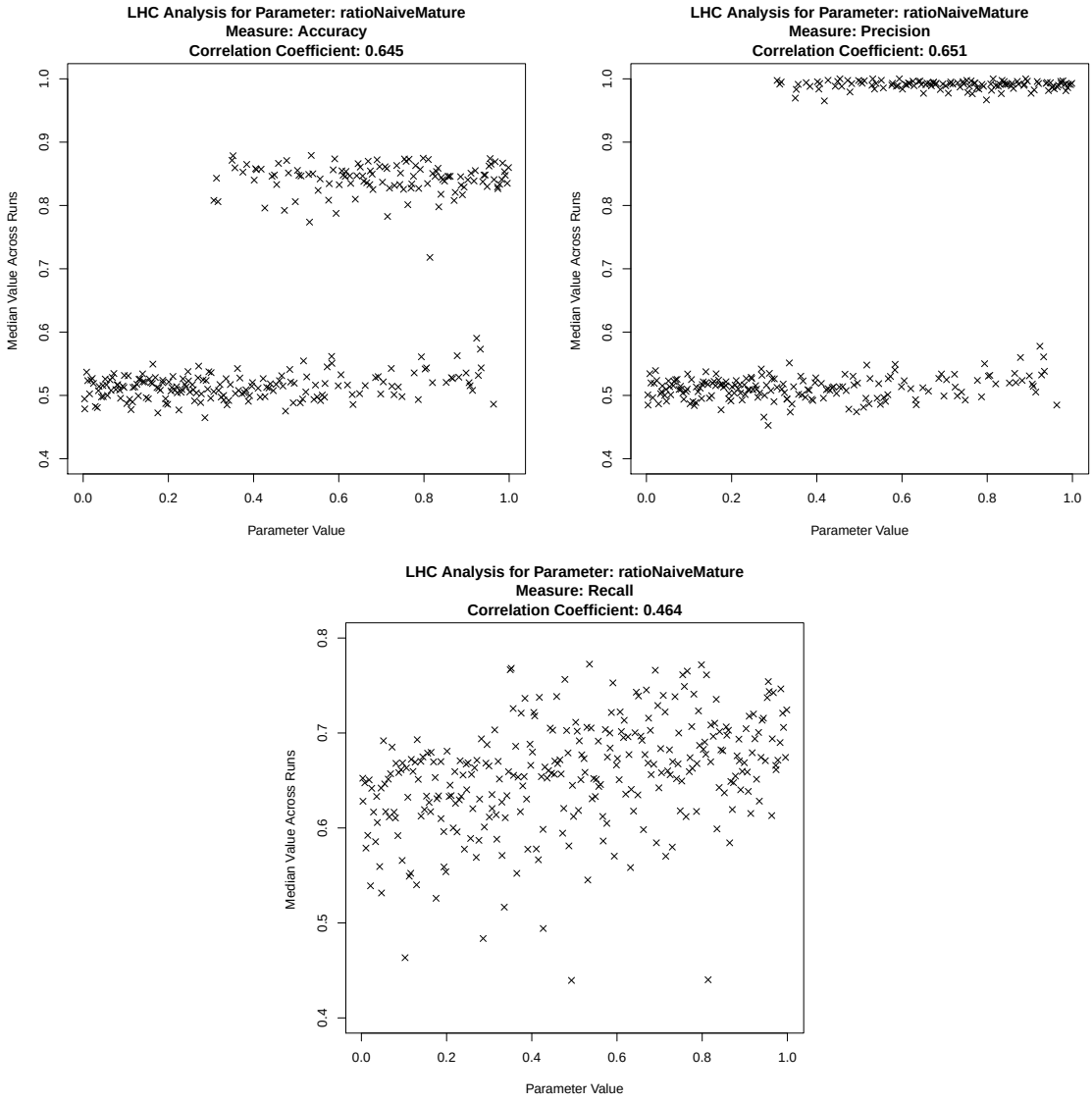


Figure 30: Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall.

Another parameter that influences the performance of the framework is $th_{training}$. Its correlation values and the plots shown of Figure 31 indicate that small values of $th_{training}$ are related to high performance. This means that large numbers of classifiers are associated with high values of accuracy, precision and recall. This contrasts with the results about the SVM classifiers, in which the low "inertia" of small ensembles was responsible for high performance. In this case, large ensembles provide a higher detection of positive instances probably because of the higher number of AISEC detectors and therefore better "coverage" of the model.

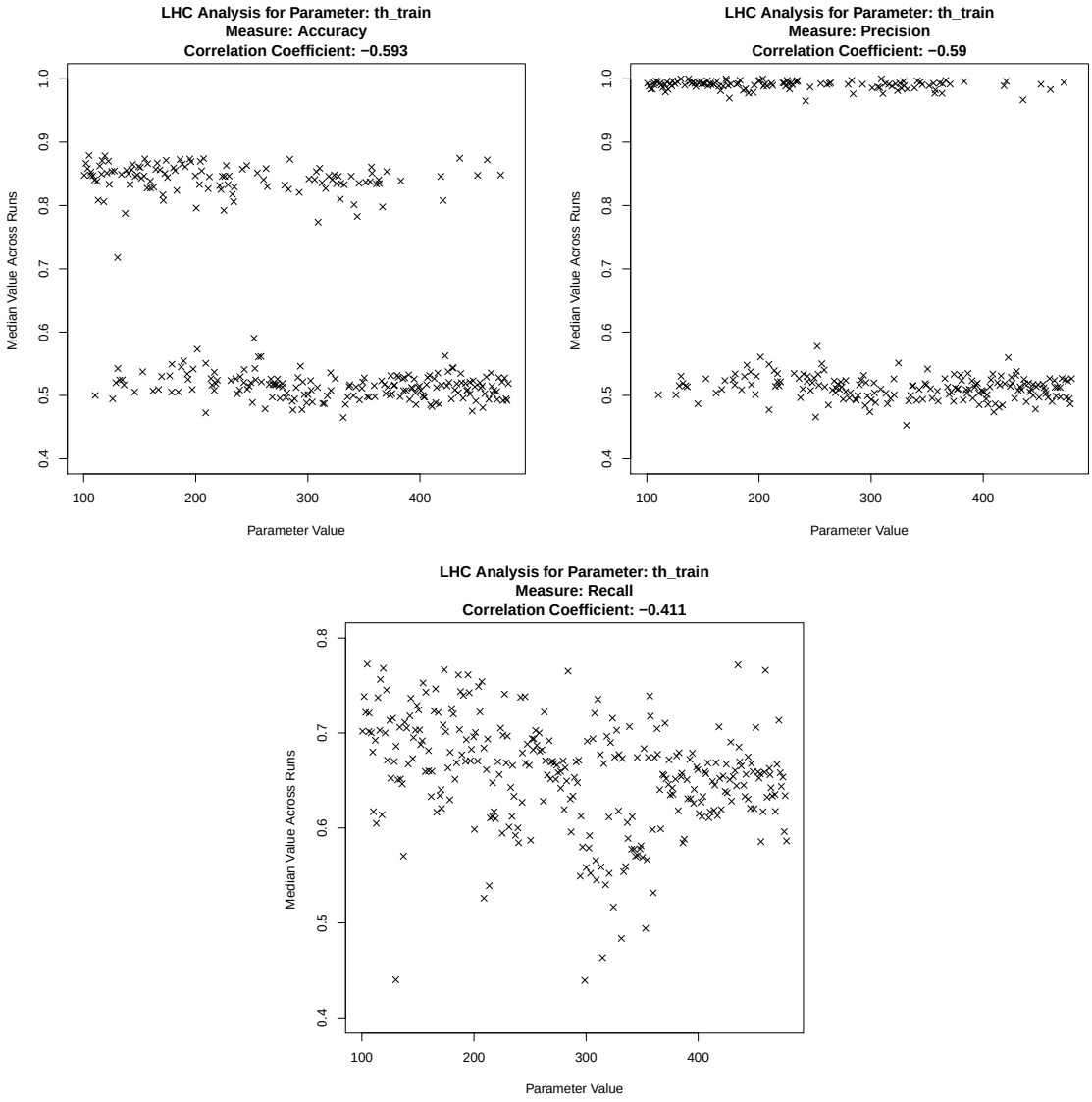


Figure 31: Latin hypercube sensitivity analysis plot of $th_{training}$ against accuracy, precision and recall.

The variable th_{ID} generated apparently contradictory results. Provided that this variable controls the sensitivity of the mechanism of concept drift inference, we would expect small values of the threshold to be associated with high performance. In fact, a low threshold should facilitate the replacement of mature classifiers with naïve ones, thus keeping the model updated. For this reason, we were expecting to observe negative values of correlation. Table 9 contradicts our expectations as the correlation of th_{ID} with accuracy does not seem to be significant, the correlation with precision is positive and that with recall is negative but small. The plots of Figure 32 confirm this counterintuitive result. The clusters in the accuracy plots are quite homogeneous. The negative trend in the top cluster of the accuracy plot

is probably generated by intermittent activation of the drift inference mechanism, which is responsible for the delay of the detectors with respect to the drifting data. This causes smaller accuray values as th_{ID} increases. Presumably, this delay is also responsible for the top right cluster in the precision plot. That plot is similar to the SVM plot of Figure **??**. This enforces our belief that both situations are caused by the low rates at which the model is updated.

Variations of the parameter $FIFOsize_{ID}$ do not seem to have effect on the performance of the framework. This in line with the results for the SVM. Although the parameters th_{online} has a limited influence on the performance of the framework, the way it affects the performance differs from the SVM case, for which it seemed to generate better performance. For AISEC, although the values of correlation are low, it seems that the higher amount of data to train a naïve classifier is collected, the higher is the performance. This result is probably due to the fact that a large amount of data is needed in order to build an effective model in AISEC. This is due to the nature of AISEC, which converts data into a model consisting of a set of detectors.

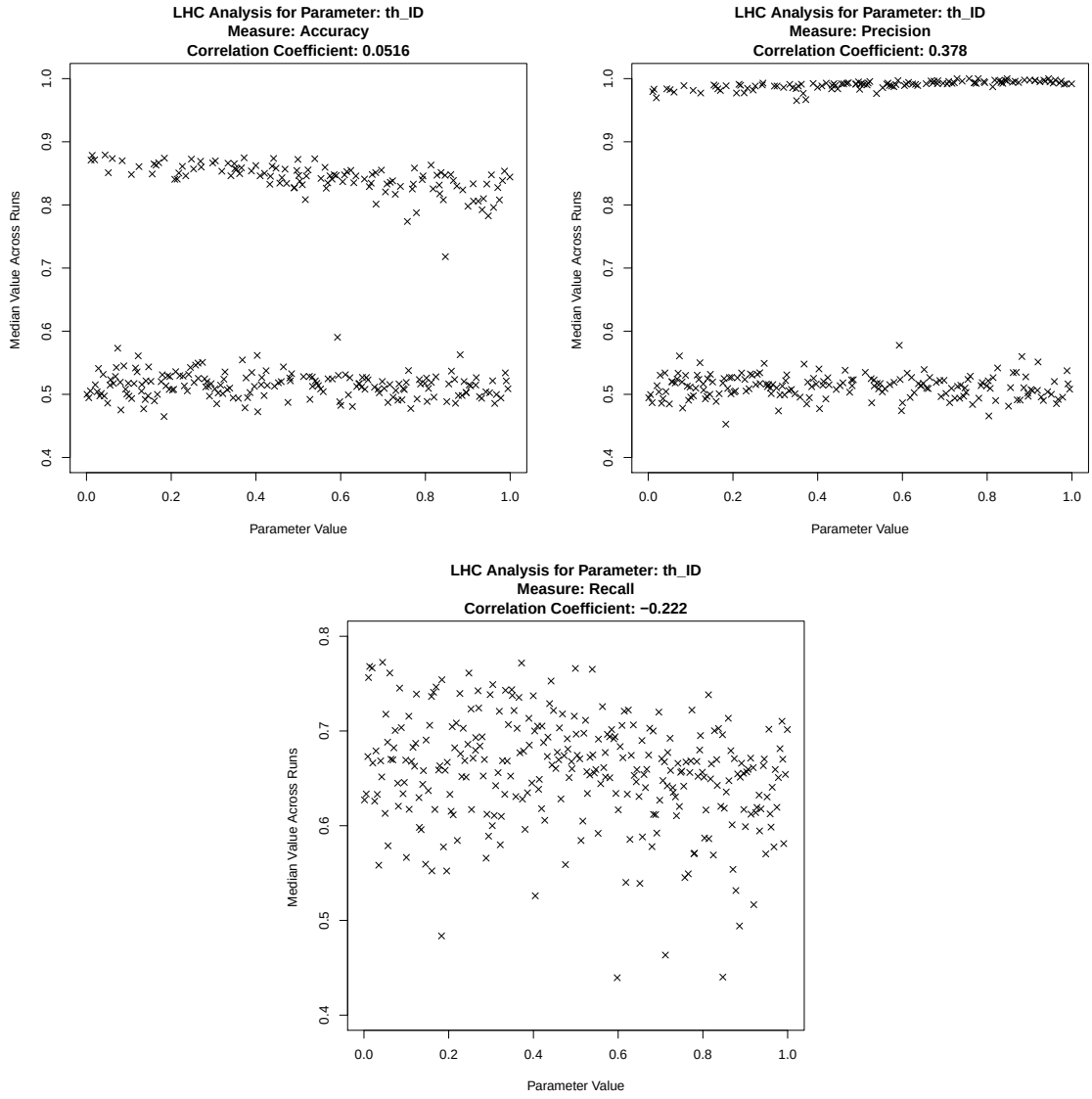


Figure 32: Latin hypercube sensitivity analysis plot of th_{ID} against accuracy, precision and recall.

4.3.1.2 Interpretation of the sensitivity analysis

Similar to the analysis that was performed for the SVM, we show the trends of TP, TN, FP and FN at different stages of a run. However, for the SVM implementation only two samples of the overall two hundred were selected.

In the case of AISEC, only one run per sample is performed because of time constraints. Showing the results of a single run may not provide sufficient information to explain why the framework reacts in a certain way. For this reason, all the samples from each cluster are used.

Firstly, the cluster of samples for which the framework shows high accuracy and

precision is considered. The behaviour of the framework is analysed by observing the magnitudes and the variations of the normalised number of TP, TP, TP and TP. The number of TP and that of TN do not vary much across drift. Their median values are respectively: 0.36815 and 0.50586. Notice that the number of TP is smaller than that of TN, while for the SVM the two measures had similar values. Initially, the number of FP is 0.01614, while the same number at the end of a run is 0.00283. This reflects on the measure of precision which increases from 0.96028 to 0.99171. The number of FN, higher than that of FP, increases from 0.10541 to 0.15157, causing a decrement of recall from 0.79065 to 0.69538. The accuracy of the framework does not undergo evident variations. The median values of accuracy across a run is 0.85967.

The second set of samples are generated from the cluster of associated with low values of accuracy and precision. The number of TP is similar to that of the other cluster. Across a run, it varies from 0.40123 to 0.32128. The TN undergo a larger decrease as soon as drift starts, from 0.47924 to 0.19606. The increase of the FP from the initial value of 0.01964 to 0.30398, causes a drop of the precision from 0.95349 to 0.51231. The variation of the number of FN is smaller than that of FP. Its median values vary from 0.09776 to 0.18049. As consequence, also the variation of recall are not as large as those of precision. This measure, from the initial value of 0.80111, reaches the final value of 0.64039. The increments of the FP and the FN and the decrements of the TP and the TN are responsible for the variation of the accuracy from 0.88017 to 0.51358.

The diagram in Figure 33 proposes an interpretation of the functioning of the framework when AISEC is used as a base learner.

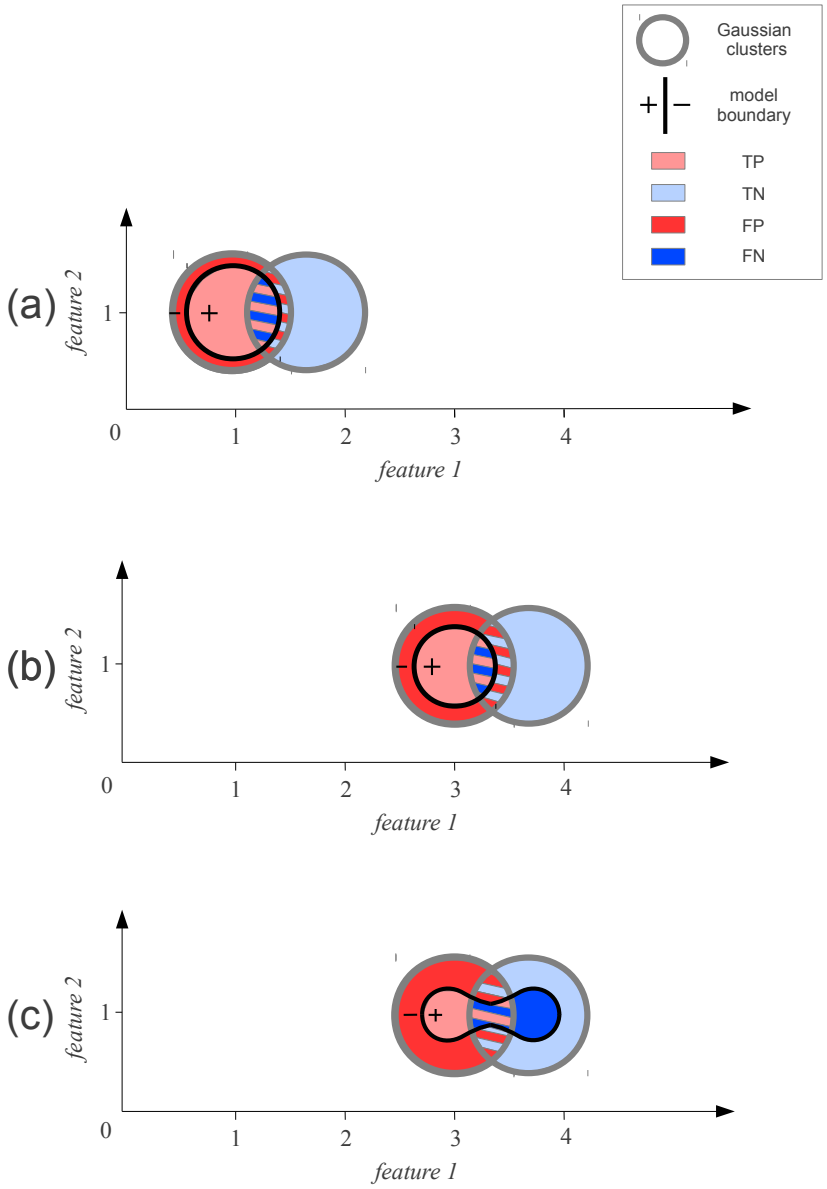


Figure 33: Representation of the model of the framework featuring AISEC at the beggining of an experiment (a), at the end of a successful experiment (b) and in case of failure (c).

It shows the boundary of the model of the framework after training (a), when a run ends with high classification performance (b) and when the framework cannot at classifying the data correctly (c). Similar to Figure 16, the grey circles indicate that the Gaussian cluster, from the initial position in (a), shift towards the final position of (b) or (c). The black line represents the boundary of the model of the framework. It has a circular shape in this case. An explanation of the reason why the boundary has this shape is given in Section 4.3.1.3. After training, the framework exhibits

the highest classification performance, however, as soon as drift starts the boundary "stretches" slightly as shown in (b). That causes a slight decrement of TP and an increment of FN (as shown in the Figures 67 and 68), as a consequence recall and accuracy decrease (Figure 69).

Concerning the samples that cannot handle the concept drift of the data stream, the boundary of the model "expands" to the "-1" Gaussian cluster as in (c), thus causing a drop of the performance. When this happens, the accuracy decreases since the number of TN drops, while the number of FP increases (Figure 71, that also affects precision. In such a way, the rates of TP and FN also vary but to a minor extent, so that recall is affected marginally.

The reason why accuracy, recall and precision seem positively correlated between each other is also explained by Figure 33. Variations of the parameter of framework affect its capability to deal with concept drift, and cause large variations of its accuracy and its precision. Precision is less affected by these variations, in fact the values of precision of an instance of the framework that deals with concept drift and the values of an instance that does not are similar. This explains the smaller magnitude of the correlation values of precision when compared to those of the other two measures.

4.3.1.3 Interpretation of the results

The rationale that was introduced to interpret the results from the SVM does not seem to applicable to explain the results from AISEC. As a matter of fact, that rationale assumes that the model maximises the separation between the classes. While this holds for the SVM, it does not for AISEC. The model in AISEC is in fact made of a set of detectors for positive instances. A detector can be generated by a training instance with positive class or by clonation of an existing detectors. The regions of the input space with high density of positive training points are covered by a higher number of detectors than sparse regions. When supervision ceases and labels from the voting, rather than the true classes are used, what causes the model to adapt seems to be the density of points. After the training phase, the model classifies a certain number of points as positives. Even though the true class of part of these points is negative, in the following iteration, these points will generate new detectors. The information following from the initial supervision about the true classes is partially lost, and from this point on density becomes the driving force

for adaptation. This means that detectors will proliferate only in regions where the density of points is high. As the distribution is Gaussian, detectors will tend to concentrate in the center of the postive cluster, this causes the boundary of the model to stretch starting from the initial iterations as shown in Figure 33 (b). The same mechanism is also reponsible for performance drop. This occurs when detectors, attracted by density, migrate towards the negative cluster thus causing the boundary of the model to appear as in Figure 33 (c).

4.3.2 Classification of Gaussian data with naïve Bayes

Although the SVM represent a very effective technique for many applications, one of the drawbacks of the SVM is the long training time [54]. Section 4.3.1 highlighted that, with AISEC, the framework proved to be even slower than when the SVM are used. In order to increase the execution time of the experiments, the framework should be tested on quicker techniques. Naïve Bayes seems to fit to our requirements. In fact, according to [54], this technique combines very fast training with short classification time. A description of this techniques is given in Section 2.2.2.2.

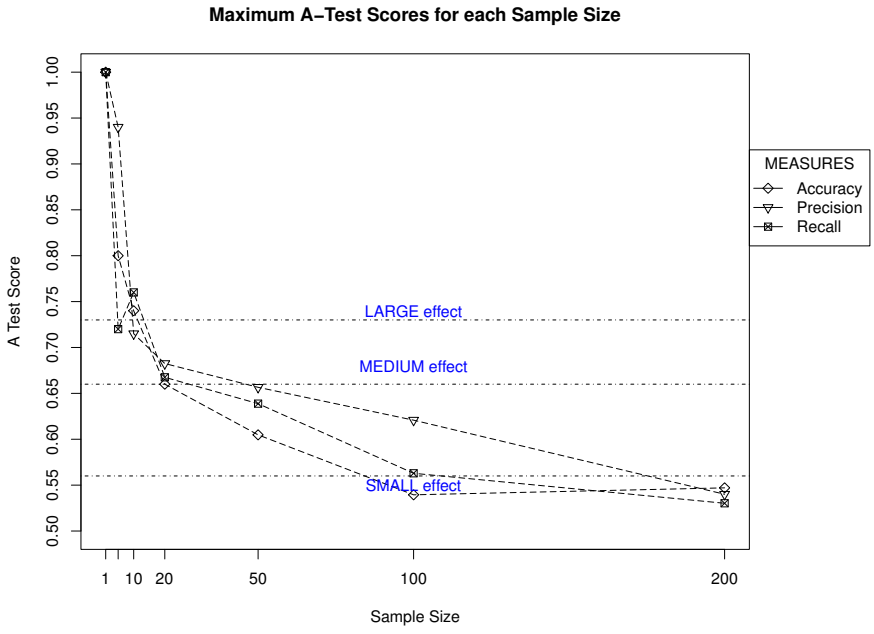


Figure 34: A-test results for the framework with naïve Bayes. The test determines that low difference between two different sets of output values is reached when 200 runs are performed.

Similar to the experiments of Section 4.3.1 and Section 4.2.1, we want to test

whether the use of adaptivity of the framework, with naïve Bayes in this case, brings any advantage in terms of performance with respect to a framework that does not update its model. The distribution of the data is the same as that used to test the framework with the SVM and AISEC, and it is described in Section 4.2.1.

The version of naïve Bayes being employed in this experiment, makes use of a Gaussian model to fit the data (as described in Section 2.2.2.2). While this version does not require parameters to be set, the parametric configuration of the framework is the same as that of the experiment with the SVM (Table 2). The null hypothesis, that assumes that there is no difference between the performance of the adaptive framework and that of the static framework, is tested by performing 200 runs with different datasets generated from a common distribution. The number of runs was determined by means of the A-test (Section 4.2.1.1), which results are shown in Figure 34. The null hypothesis is formulated as:

> *The performance of an instance of the adaptive framework featuring naïve Bayes, when tested on the Gaussian dataset, is not statistically different from the performance of an instance of the framework with a static model.*

In order to compare the static and the adaptive framework, the performance at the end of a run are used rather than the median values. This decision was taken on the base of the considerations of Section 4.3.1, which assert that the performance measured at the end of a run denotes more clearly the ability to deal with the drift of this dataset. On those values, the Mann-Whitney test reveals that the all probabilities of the values of accuracy, precision and recall of the static and the dynamic framework being generated from the same distribution equal $2.2E-16$. Therefore the null hypothesis can be rejected with a confidence level of 0.995. the same distribution. In particular, the time required to complete a run with naïve Bayes is about an order of magnitude smaller than the time required by the SVM. Figure 35 shows the values of accuracy, precision and recall of the adaptive framework.

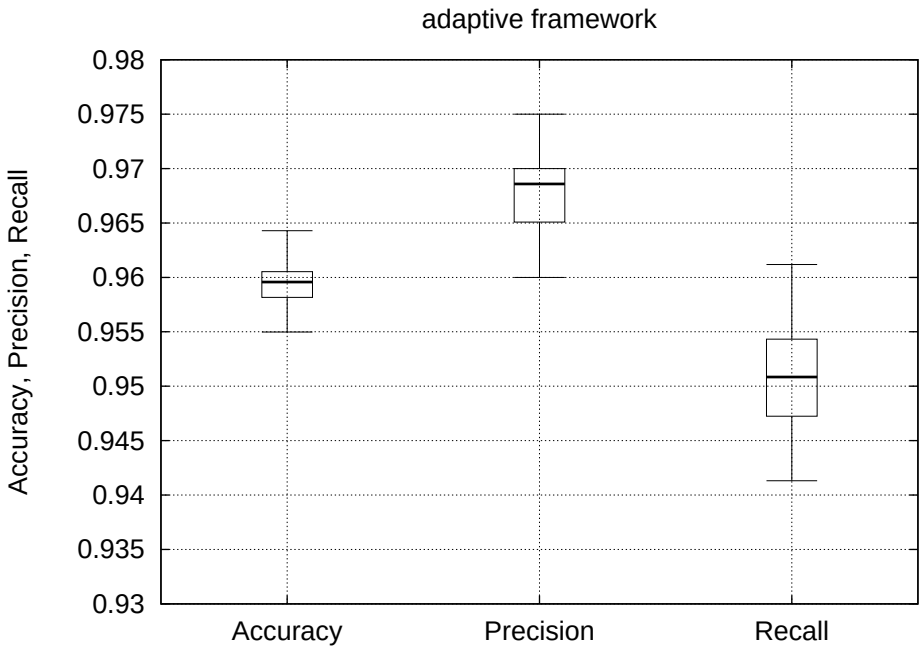


Figure 35: Values of accuracy, precision and recall of the adaptive framework with naïve Bayes.

4.3.2.1 Sensitivity analysis

An analysis of the sensitivity was performed to determine the effect of the parameters of the framework on its functioning. The ranges of the parameters are shown in Table 10.

Parameter	**Interval**
$th_{training}$	[20, 200]
Th_{online}	[20, 500]
$ratio_{NaiveMature}$	[0, 1.0]
$FIFOsize_{ID}$	[2,30]
th_{ID}	[0, 1.0]

Table 10: Intervals of the parameters for the sensitivity analysis of the Framework with naïve Bayes as a base learner.

Table 11 displays the values of correlation between input parameters and output measures. From that table it is evident that, apart from th_{ID}, the parameters seem

to have little or no influence on the performance. This indicates that the framework with naïve Bayes performs well with different numbers of mature classifiers, large or small testing windows, different values of $FIFOsize_{ID}$.

Parameter	Accuracy	Precision	Recall
$th_{training}$	0.337	-0.506	0.363
th_{online}	-0.207	0.0128	-0.202
$ratio_{NaiveMature}$	0.21	0.398	0.19
$FIFOsize_{ID}$	0.121	0.102	-0.184
th_{ID}	-0.643	0.633	-0.644

Table 11: Correlation coefficients between parameters and output measures.

The parameter $ratio_{NaiveMature}$, however, has some effect on the performance. Figure 36 suggests that when the number of naïve classifiers is notably smaller than the number of mature classifiers, accuracy, precision and recall drop. In fact, a cluster of dense points is identifiable in the bottom left corner of each plot.

A characteristic of these plots caught our attention. For each plot, the cluster at the top is much denser than the cluster at the bottom. This means that the number of points, and therefore parametric configurations, for which the framework has high performance is much larger than the number of points associated with low performance. An explanation to this observation may be related to the fact that the technique of naïve Bayes suits particularly well to the dataset being used. In fact, the data has Gaussian distribution and the implementation of naïve Bayes that is employed fits data to a Gaussian model. Therefore, despite the different parametric configurations of the framework, in this context, the technique of naïve Bayes may be responsible for the high performance.

4.3.3 Classification of Gaussian data with the multilayer perceptron algorithm

This section shows the performance of the framework with the multilayer perceptron algorithm (MLP). This classification technique, described in Section 2.2.2.2, generates a discriminant function in order to classify instances. For our experiments, we use MLP classifiers with a single internal node to reduce the time required to complete a run. In this way, the models that are generated have linear models. The

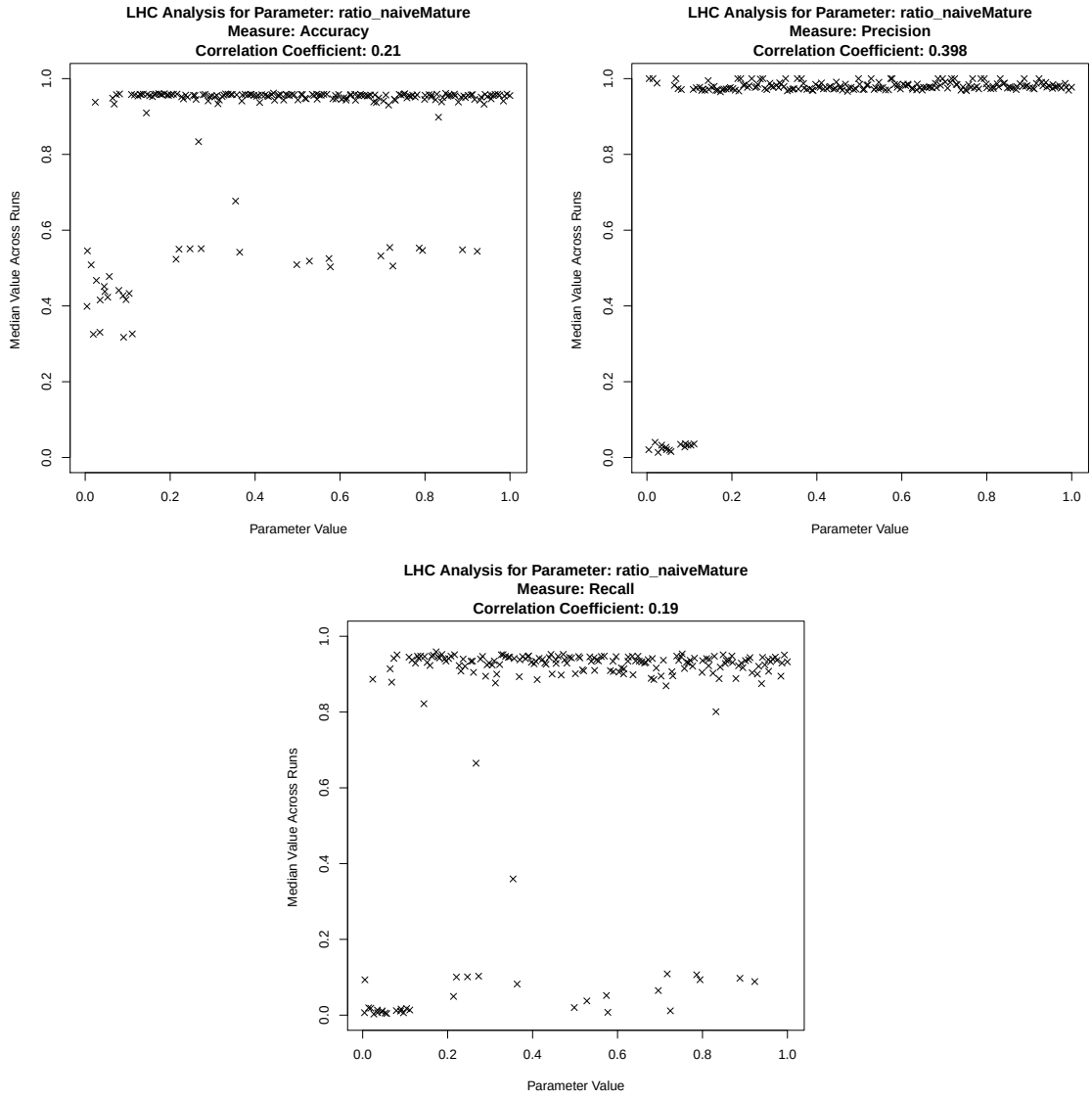


Figure 36: Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall.

learning rate is 0.7, the momentum is 0.1 and the number of epochs is 30. The framework uses the same parameters of experiment with the SVM of Section 4.2.1, they are listed in Table 2. The distribution is the two-Gaussians dataset defined in Section 4.2.1. Similar to the previous experiments, the null hypothesis is formulated as:

> *The performance of an instance of the adaptive framework featuring the MLP algorithm, when tested on the Gaussian dataset, is not statistically different from the performance of an instance of the framework with a static model.*

Following from the results of the A-test of Figure 37, the null hypothesis is tested on 200 runs. Different runs use different datasets generated from The Mann-Whitney test determines that the probabilities that the values of accuracy, precision and recall of the two instances of the framework are generated from the same distribution are respectively: 2.2E-6, 1 and 2.2E-6. Therefore, the null hypothesis cannot be rejected.

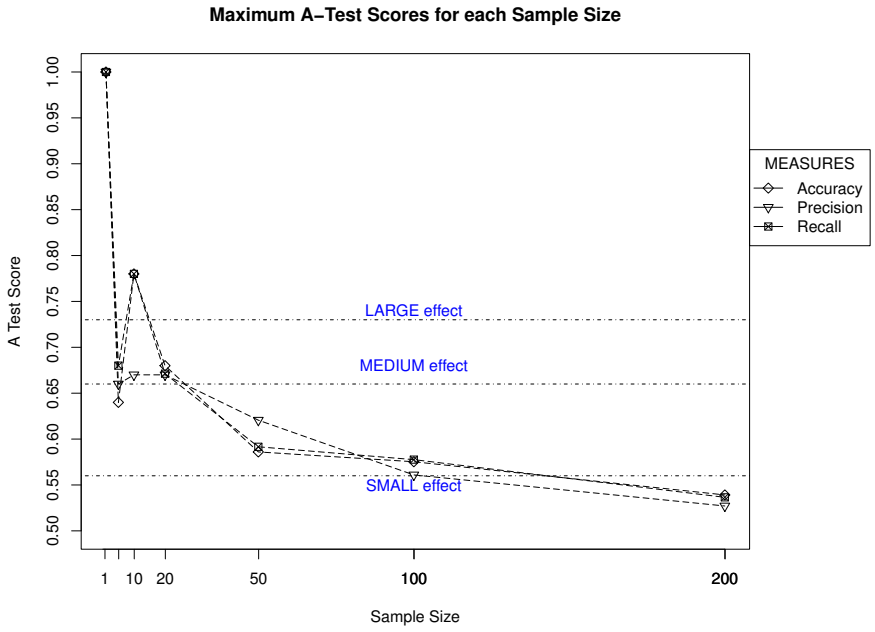


Figure 37: A-test results for the framework with MLP. The test determines performing 200 runs is sufficient.

Although the distributions of the values of accuracy and precision are different according to the Mann-Whitney test, the values of these measures lie in a small subset of the input space, as shown in Figure 38. The low values of accuracy and recall entail that the framework with this technique and this configuration is not able to deal with the concept drift of the Gaussian distribution.

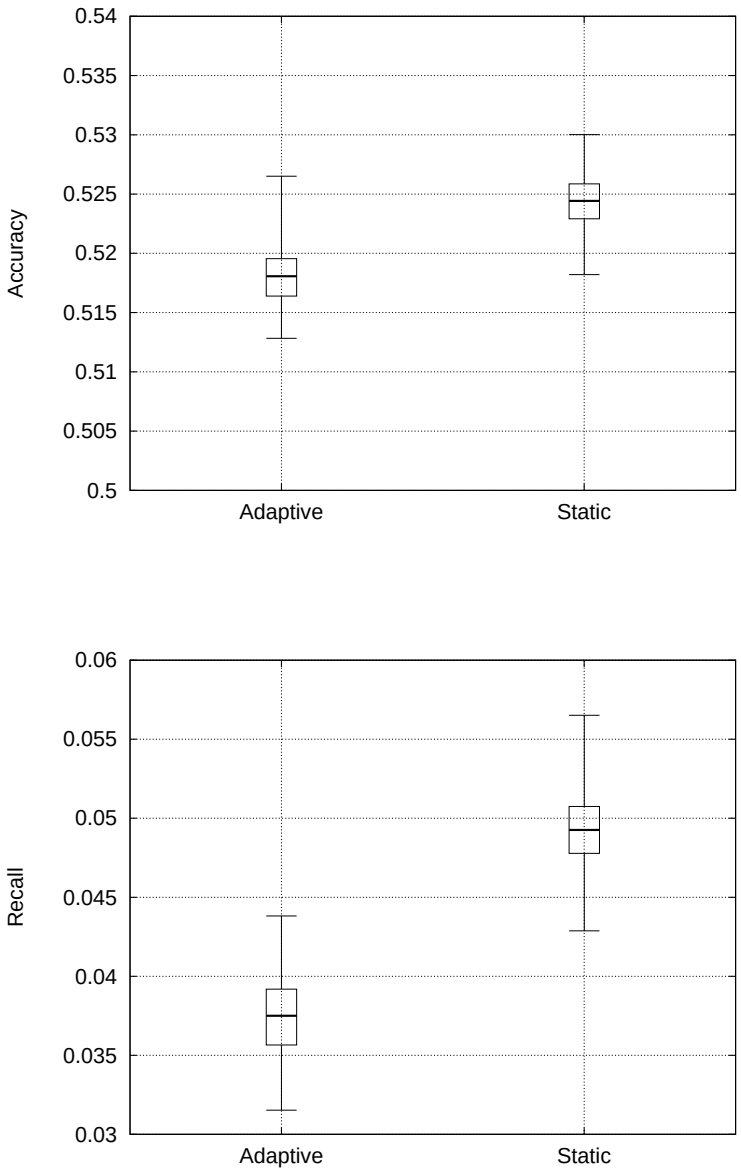


Figure 38: The overlap between the values of accuracy (top plot) and precision (bottom plot) of the adaptive of the static and the adaptive framework is limited. However the different sets of values lie is small intervals of the spaces of the parameters.

4.3.4 Sensitivity analysis

A sensitivity analysis would clarify if there are sets of parameters for which the framework is able to classify the drift of the data. Table 12 shows the correlations between input parameters and output measures. The table highlights that accuracy and recall are negatively correlated to precision, as for the framework with the SVM.

Parameter	**Accuracy**	**Precision**	**Recall**
$th_{training}$	0.21	0.333	-0.0857
th_{online}	0.557	0.197	0.463
$ratio_{NaiveMature}$	0.038	-0.0624	0.0391
$FIFOsize_{ID}$	-0.0485	0.0912	-0.147
th_{ID}	0.0706	-0.0468	0.117

Table 12: Correlation coefficients between parameters and output measures.

This may be explained by the fact that both the SVM and the MLP use discriminant functions. The values of correlation are low for all the parameters, except th_{online}. However, Figure 39 shows that the samples of the dense clusters of the plots of accuracy and recall have an increasing trend that could affect the values of correlation. The plot of the accuracy of the framework are slightly higher than 0.6. That indicates that there are no settings of the parameters for which the framework is able to deal with the drift of this dataset.

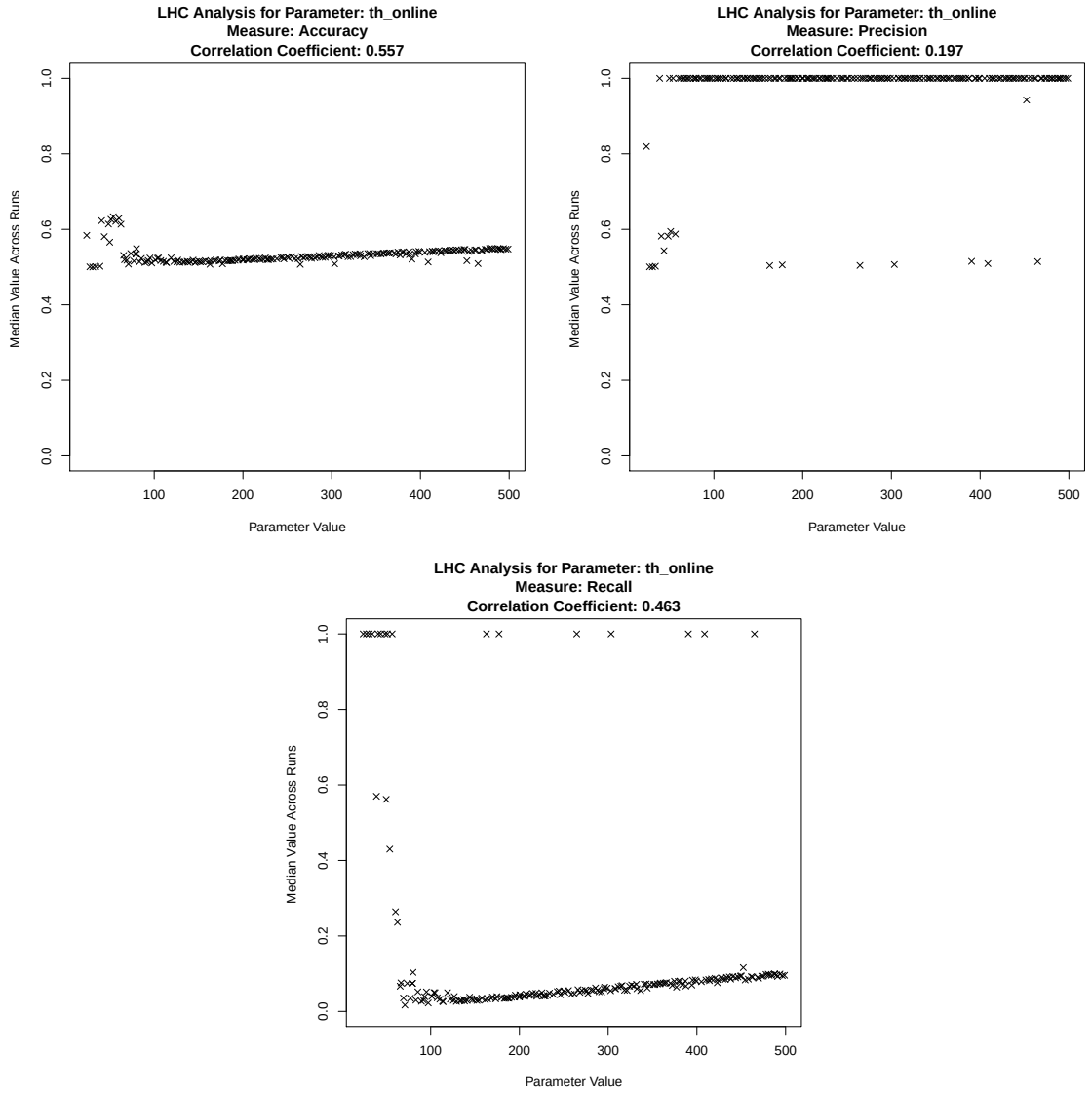


Figure 39: Plot of the sensitivity analysis of th_{online} against accuracy, precision and recall when the MLP is used.

4.3.5 Classification of Gaussian data with C4.5

This section analyses the performance of the framework with the algorithm C4.5, described in Section 2.2.2.2. This algorithm generates decision trees. We intend to determine whether the use of C4.5 allows the framework to deal with concept drift. The parameters of the framework for this experiment are the same of of those used for the SVM, naïve Bayes and MLP, and they are defined in Table 2. The distribution is the two-Gaussians dataset used for the other experiments. The null hypothesis, is tested over 200 datasets having the same distribution. It states that:

The performance of an instance of the adaptive framework featuring

the C4.5 algorithm, when tested on the Gaussian dataset, is not statistically different from the performance of an instance of the framework with a static model.

The number of runs is determined using the A-test, whose results are plotted in Figure 40. The A-test score of accuracy has value 0.55835, and therefore is below the threshold of the lowest region, which has value 0.56.

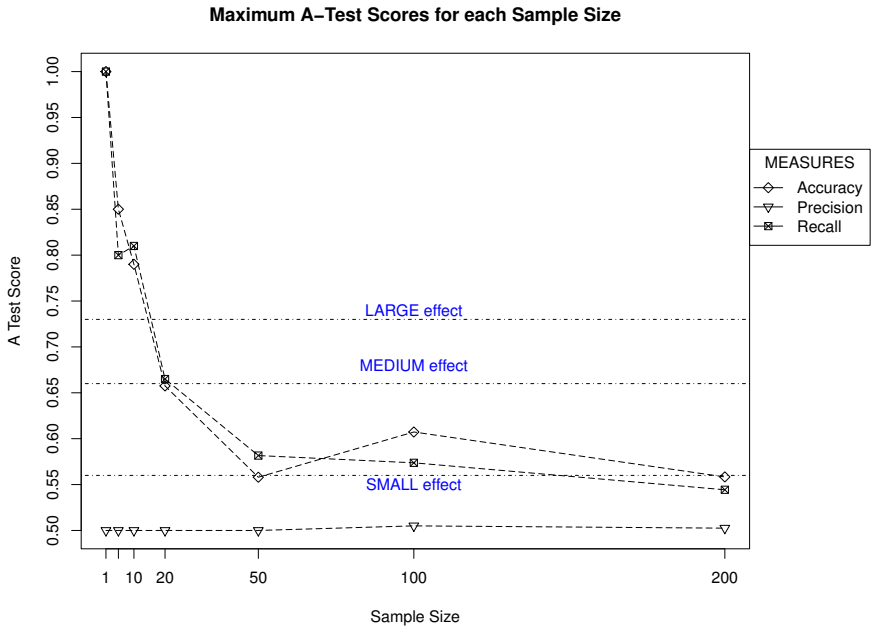


Figure 40: A-test results for the framework with the algorithm C4.5. The test determines performing 200 runs is sufficient.

The p-values of he measures of accuracy, precision and recall are respectively: 0.8039, 1 and 0.02247. These values cause the rejection of the null hypothesis with a confidence level higher than 0.995.

4.3.6 Sensitivity analysis

Table 13 shows that varying the parameters of the framework does not seem to produce any effect on its performance. An explanation of the high correlation values observed for th_{online} was given in Section 4.3.3. Figure 41 displays the plots of the sensitivity analysis of the parameter $th_{training}$. Those plots clarify that there is no setting of the parameters that allows the framework to deal with the drift of this dataset.

Parameter	Accuracy	Precision	Recall
$th_{training}$	-0.127	0	-0.165
th_{online}	1	0	1
$ratio_{NaiveMature}$	0.0249	0	-0.041
$FIFOsize_{ID}$	-0.0642	0	-0.14
th_{ID}	-0.0829	-0	0.264

Table 13: Correlation coefficients between parameters and output measures.

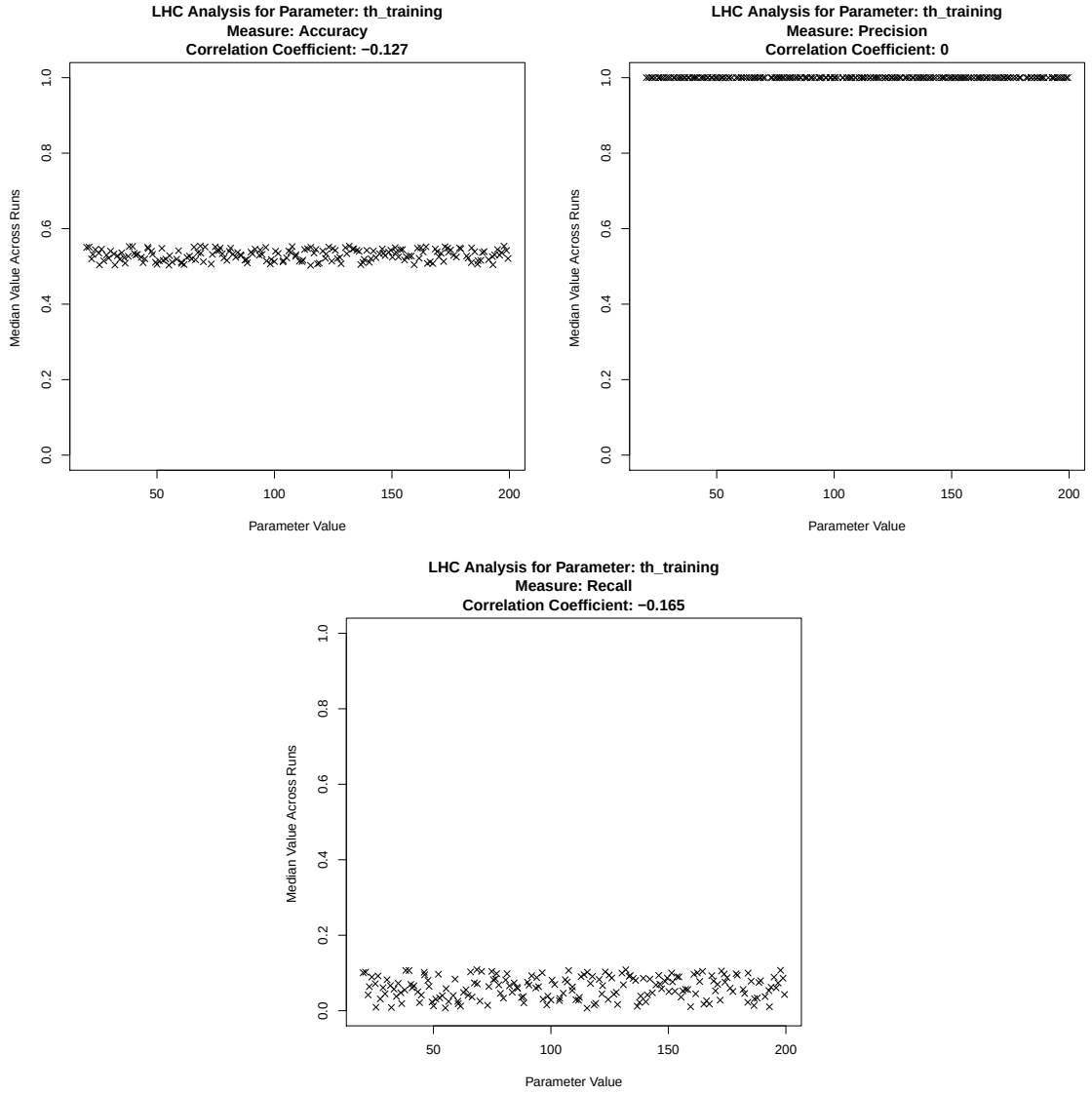


Figure 41: Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall.

4.4 Comparison of the framework with unsupervised model updating techniques

This section compares the framework with a set of techniques that are able to update their models without supervision. They are: the single-classifier approach of Section 2.4.3, and the online clustering algorithms CluStream and DenStream, presented in Section 2.5.2. The aim of this comparison is to highlight the differences between these approaches.

The single-classifier technique (SC) uses a moving window of fixed size. Two variants of this algorithm are also used. The first variant (SC-NO) uses a nonoverlapping window in order to reduce the amount of computation. In this way, if the window has size 1000, the model is updated every 1000 instances rather than for every instance that is presented. The second variant (SC-DAF) uses the density-adaptive forgetting mechanism described in Section 2.4.1.4. Density-based forgetting proposes that an instance should be forgotten if a high number of more recent instances lie in its neighborhood. The classification techniques that are used by the variants of the single-classifier techniques and by the framework are naïve Bayes and the SVM. CluStream (CS) and DenStream (DS) use a two-level clustering approach. The online level updates a set of micro-clusters. Micro-clusters reduce the memory consumption and the amount of computation by maintaining statistics about incoming instances, that are discarded. High-level clustering is applied to micro-clusters, rather than raw data. The technique of "seeding", which is illustrated in Section 2.5.2, is used to associate clusters with the classes. It uses training data to initialise two centers. Each of these represents an estimate of the center of the distribution of a class. Initially, a center is associated with the centroid of the training instances with positive class, the other center is initialised with the centroid of the training instances with negative class.

The integration between seeding and CluStream is straightforward for this data distribution. Initially, the set of centers generated by the seeding (from training data) serve to initialise the high-level k-means of CluStream. Since the number of centers is two, the k-means implementation for high-level clustering is initialised with $k = 2$. When CluStream has collected at least *horizon* unlabelled instances and it has identified the first two clusters, each cluster is labelled with the class of the center that is closest to its centroid. The coordinates of the centroids of the

clusters become the new centers. This process is repeated until streaming data is provided. In order to label an instance, the closest micro-cluster to the instance is identified. Then, the instance being processed is labelled with the label of the cluster that micro-cluster belongs to. Only for the first *horizon* instances, since the first set of clusters has not been generated yet, random labels are assigned to those instances.

Technique	**Parameter**	**Value**
single-classifier	window size	1000
single-classifier (nonoverlapping window)	window size	1000
single-classifier (adaptive forgetting)	window size	1000
	θ	0.2
	m	5
	τ	0.7
CluStream	horizon	1000
	# of micro-clusters	100
	micro-cluster radius	2
	k	2
DenStream	λ	0.006643856
	μ	1
	β	0.001
	initial # of micro-clusters	1000
	DBSCAN ϵ	0.01
	DBSCAN θ	11
	high-level DBSCAN ϵ	0.07
	high-level DBSCAN θ	6

Table 14: Parametric configurations of the of the unsupervised comparisons of the framework.

The classification process is similar for DenStream. A difference is that DenStream does not generate a fixed number of clusters. Therefore, it may identify more or less than two clusters. For that reason, the clusters formed by DenStream are associated with the classes (and the centers are updated) only when the number of clusters equals the number of centers.

The values of the parameters of these techniques are listed in Table 14. The effect of these parameters on their respective techniques are described in Section 2.4.1.4, Section 2.4.3 and Section 2.5.2. The parameters of the framework and the parameters of the SVM have the values of Table 2.

The dataset with two Gaussian classes is used for this comparison. In fact, after having proved that the framework is able to deal with this distribution, it would be interesting to observe how the other unsupervised techniques perform on the same data. In order to provide a fair comparison, the smae classification technique is used in order to compare the framework and the methods with a single classifier. We assume that the framework does not bring any advantage in terms of performance with respect to the comparative methods. When the technique of naïve Bayes (NB) is used, the null hypothesis is stated as:

> *The performance of an instance of the adaptive framework using naïve Bayes, when tested on the dataset with Gaussian distribution, is not statistically different from the performances of the comparative methods.*

The Mann-Whitney test applied to the values of accuracy, precision and recall of the techniques generates the the p-values of Table 15.

	SC + NB	SC-NO + NB	SC-DAF + NB	CS	DS
Accuracy	0.0673	**0.00158**	**2.2E-16**	**2.2E-16**	**2.2E-16**
Precision	**2.2E-16**	**2.94E-12**	**2.2E-16**	**2.2E-16**	**2.2E-16**
Recall	**2.2E-16**	**0.00019**	**2.2E-16**	**2.2E-16**	**7.5E-9**

Table 15: p-values of the framework with naïve Bayes and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use naïve Bayes). The p-values that are responsible for the rejection of the hypothesis are highlighted in bold type.

The rejection of the hypothesis with a confidence level alpha of 0.995 denotes that there are differences between the framework and the other methods. Concerning precision and recall, the differences are evident. Concerning accuracy, the rejection occurs for the clustering methods CS and DS, for SC-DAF and for SC-NO. However, the p-value of SC-NO is only marginally below the threshold. The framework and SC have similar values of accuracy.

The same hypothesis, with the difference that the SVM are used instead of naïve Bayes is also tested. In this case, the Mann-Whitney test generates the p-values of Table 16. The hypothesis can be rejected for all the techniques that are compared with the framework, with some differences. In particular, the performance of the

framework is different from those of CS and DS for all the measures. The accuracy of the framework and the three single-classifier methods have similar values of accuracy, but not precision and recall.

	SC + SVM	SC-NO + SVM	SC-DAF + NB	CS	DS
Accuracy	0.0925	0.0578	0.1374	**2.2E-16**	**2.2E-16**
Precision	**2.2E-16**	**2.2E-16**	**2.2E-16**	**2.2E-16**	**2.2E-16**
Recall	**2.2E-16**	**2.2E-16**	**2.2E-16**	**2.2E-16**	**2.2E-16**

Table 16: p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use the SVM). The p-values that are responsible for the rejection of the hypothesis are highlighted in bold type.

The plots of Figure 42 show the distributions of the accuracy values of the unsupervised techniques across the 200 runs. They indicate that, apart from the single-classifier with naive Bayes (SC-DAF+NB) and DenStream (DS), all the other techniques have similar performances. However, the accuracy of CluStream (CS) is slightly smaller than the others. Moreover, the instance of the framework with naïve Bayes and that with the SVM perform similarly on this data. The higher values of precision of the framework with respect to the comparative techniques, that can be observed in Figure 43, may be caused by small delays of the mechanism of drift inference. In fact, the framework is the only technique of this comparison that is triggered by such a mechanism, while the other techniques evolve over time. That could also be the reason for the values of recall of the Framework being marginally smaller than the values of recall of the other techniques, as displayed by Figure 44.

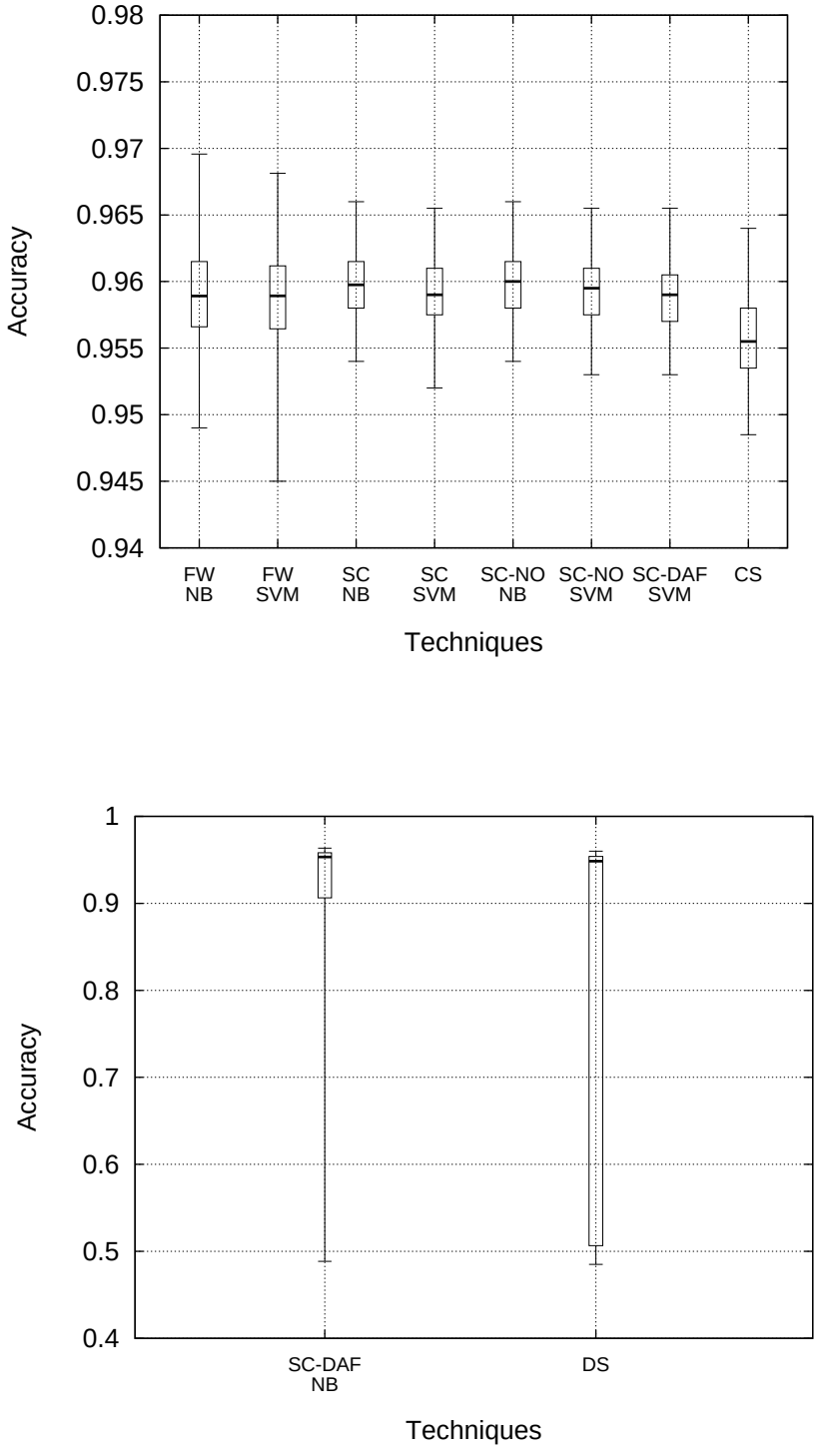


Figure 42: Box and whisker plots of the values of accuracy of the techniques for unsupervised model updating.

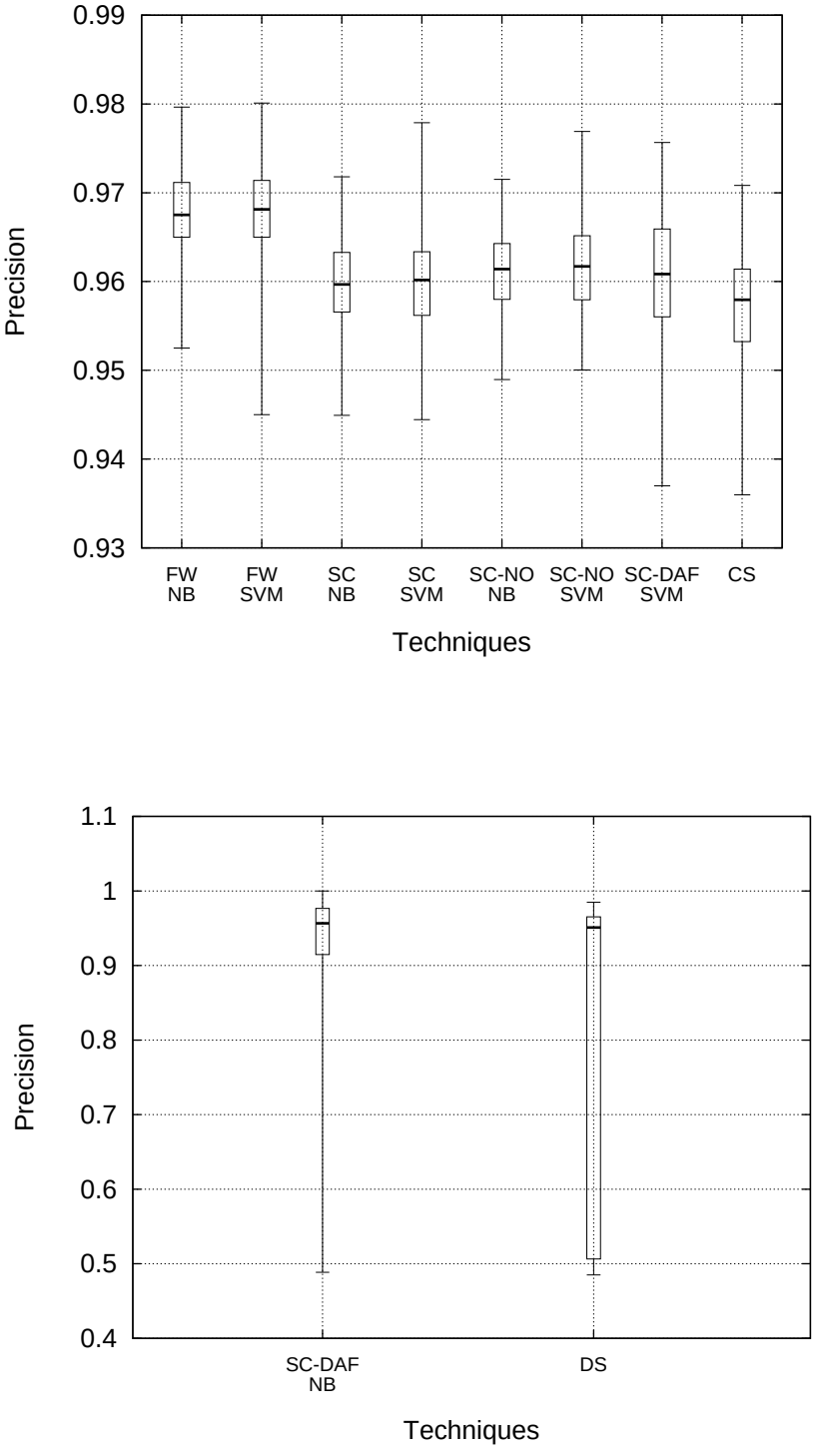


Figure 43: Box and whisker plots of the values of precision of the techniques for unsupervised model updating.

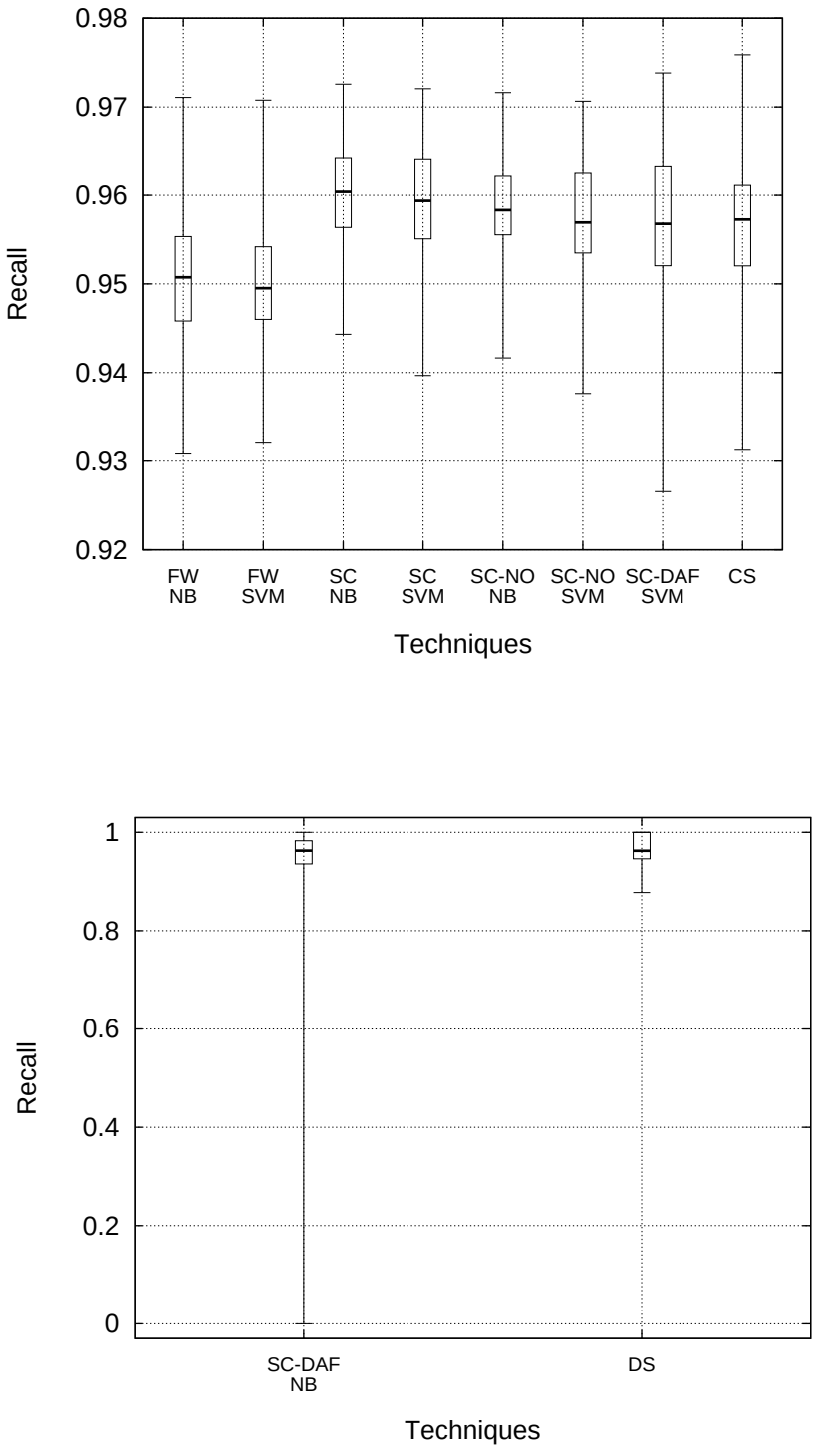


Figure 44: Box and whisker plots of the values of recall of the techniques for unsupervised model updating.

4.5 Data with multiple Gaussians with changing positions and characteristics

The experiment presented in this section has the purpose of evaluating the performance of the framework on a dataset with different characteristics. In particular, a modified version of the four-Gaussian dataset described in Section 2.3.2.1 is used. The class imbalance has been reduced with respect to the original dataset. To be more precise, the prior probabilities of the classes now have the values $P(\text{“}+1\text{”}) = 0.25$ and $P(\text{“}-1\text{”}) = 0.75$. The distribution generates 500 training instances without with fixed distribution and 10^6 unlabelled instances with concept drift.

Similar to the previous experiments, we test the ability of the framework to classify this drifting distribution by comparing its performance against a static version of the framework. The techniques on which the framework is tested are naïve Bayes and the SVM. On one side, naïve Bayes affords very fast classification, on the other side, this techniques may perform poorly on some distributions. For that reason, the SVM are also used. The parameters of the SVM are described in Table 2. For both classification techniques, the same parameters of the framework are used, they are illustrated in Table 17.

Parameter	**Value**
$th_{training}$	23
th_{online}	25
$ratio_{NaiveMature}$	0.4
$FIFOsize_{ID}$	10
th_{ID}	0.1

Table 17: Parametric configuration of the framework for the experiment involving the SVM and Gaussian data.

These parameters generate, on average, instances of the framework with five mature classifiers and two naïve classifiers. The inference of drift mechanism is set up to be sensitive, in order to provide quick detection and an high rate of replacement of mature classifiers. The value of the parameter th_{online} determines that naïve classifiers are trained with a relatively low number of instances.

The performance of an instance of the adaptive framework, when tested on the dataset with four clusters with Gaussian distributions, is

not statistically different from the performance of an instance of the framework without adaptivity, that is, with a static model.

The Atest, performed using naïve Bayes determined that performing 200 runs is sufficient, as shown in Figure 45. Concerning the instance of the framework with

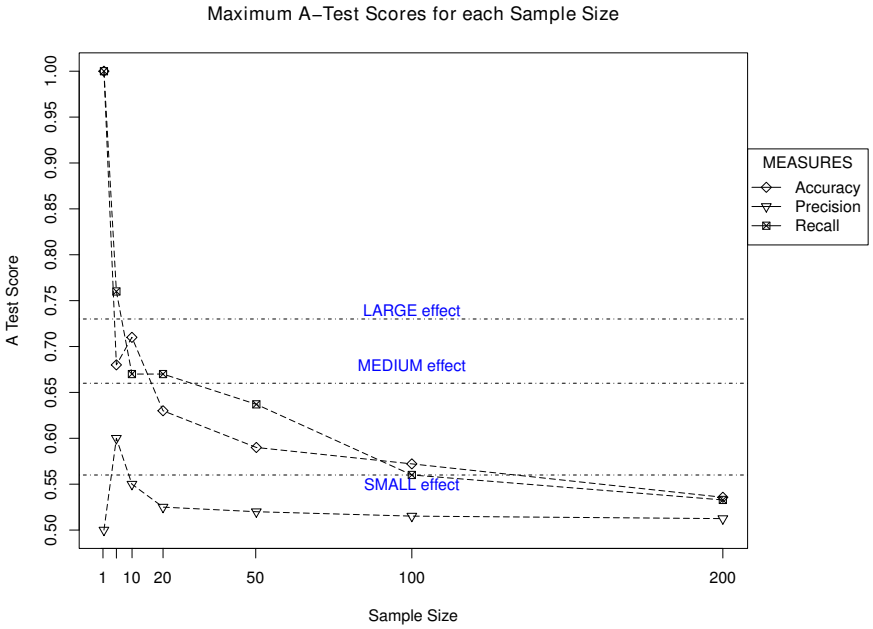


Figure 45: A-test results of the framework using naïve Bayes.

naïve Bayes, the results of the Mann-Whitney test determines that the probabilities of the values of accuracy, precision and recall of the two different instances of the framework being generated from the same distributions are respectively: 2.2E-12, 3.7008E-08 and 3.534E-08. Therefore, the null hypothesis is rejected with a confidence level of 0.995. Concerning the instance of the framework that uses the SVM, the same probabilities have the value 2.2E-12. As a consequence, also in this case the null hypothesis is rejected with the same level of confidence.

The rejection of the null hypothesis implies that, for both classification techniques, the adaptive instance of the framework generates values of accuracy, precision and recall that are different from the same values of the static instance of the framework. However, this does not necessarily mean that the adaptive instances of the framework are able to deal with the drift of this dataset. Figure 46 compares the accuracy of the adaptive instance and that of the static instance when naïve Bayes is used. Even if the two distributions of values are different, all the values of

the adaptive instance are around 0.75. Given that 25% of the instances have class "+1" and the remaning instances have class "−1", a classifier that assigns negative class to every instance would have the same performance as the adaptive framework with naïve Bayes. Figure 47 shows the values of accuracy of the instances of the

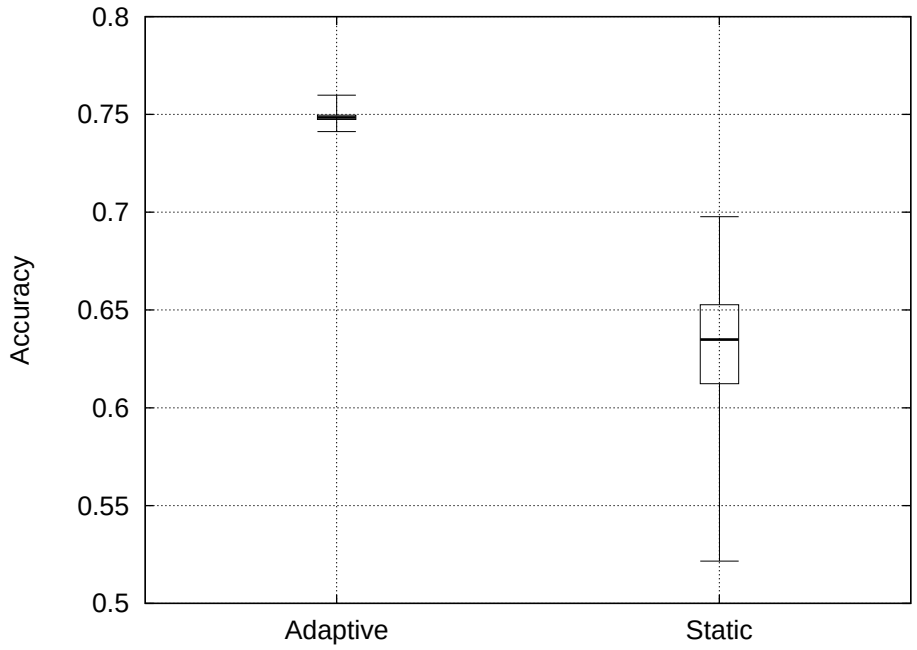


Figure 46: Distributions of the values of accuracy of the static instance and the dynamic instance of the framework using naïve Bayes.

framework that use the SVM. The values of the static framework are smaller than those of the adaptive counterpart. The values generated by the adaptive framework form two separate groups. A group has values around between 0.7 and 0.9, while the other group contains values around 1 or slightly smaller. The neat distinction between the two groups suggests that the framework is able to deal with the concept drift of this dataset only for the runs of the group with higher values. That group contains 93 runs, therefore the other group contains the remaining 107 runs.

Finally, although the hypothesis is rejected for both instances of the framework, only the instance with the SVM is actually able to classify this distribution with high performance. That occurs for part of the runs. The instance with naïve Bayes is not able to deal with the concept drift of this distribution.

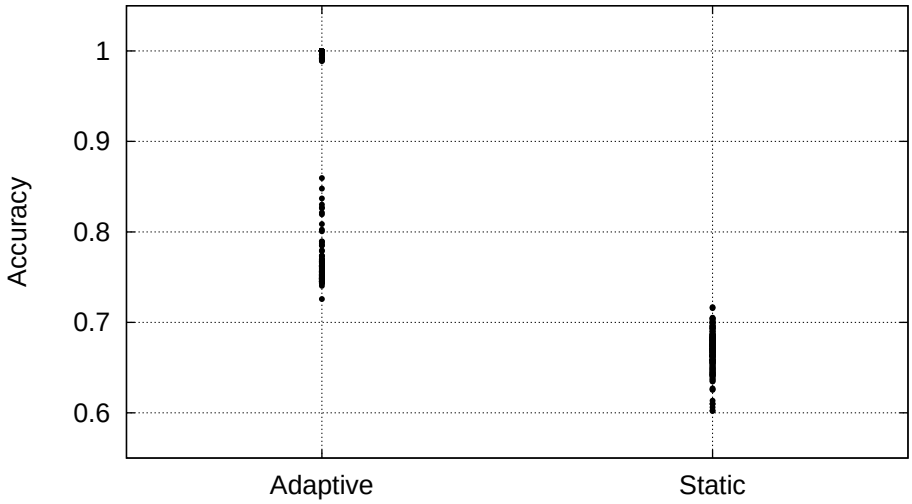


Figure 47: Values of accuracy of the static instance and the dynamic instance of the framework using the SVM.

4.5.1 Comparison with unsupervised techniques

The performance of the framework is measured against a selection of the unsupervised techniques analysed in Section 4.4. The goal of this experiment is to check the performances of the different techniques over this dataset.

For this experiment, the implementation of DenStream and that of CluStream are different with respect to their implementations of Section 4.4. At each iteration, the entire clusters of micro-clusters of the previous iteration are maintained, instead of the estimates of the centers. The label of a cluster is "transferred" to the newly generated cluster whose micro-clusters have the closest mean distance from the older micro-clusters.

Also the implementation of CluStream is slightly different. In particular, the initialisation of the centers of CluStream is performed by using k-means. To be more precise, k-means with $k = 4$ clusters the set of training instances, in order to identify four centroids and associate labels to them based on the majority labels of their respective clusters.

Among the three variants of the single-classifier algorithm defined in Section 4.4, only the one with nonoverlapping windows is maintained. This decision is motivated by the good performance of that variant and by its higher speed of classification with respect to the other two variants.

Table 18 presents the values of the parameters of the techniques. The configuration of the framework was performed via a global sensitivity analysis. Similarly, the length of the window of the single classifier is the result of a large set of trials with different values. Concerning the clustering algorithms, firstly, the parameters that control the number, the forgetting rate and the coverage of the micro-clusters were set. Secondly, the parameters of the higher-level clusterers were set.

Technique	**Parameter**	**Value**
single-classifier	window size	500
CluStream	horizon	1000
	# of micro-clusters	1000
	micro-cluster radius	2
	k	4
DenStream	λ	0.004643856
	μ	4
	β	0.03
	initial # of micro-clusters	500
	DBSCAN ϵ	0.5
	DBSCAN θ	3
	high-level DBSCAN ϵ	1.2
	high-level DBSCAN θ	1

Table 18: Parametric configurations of the unsupervised comparisons of the framework.

In order to evaluate the similarity between the techniques being analysed, the following hypothesis is stated:

> *The performance of an instance of the framework using naïve Bayes, when tested on the dataset with multiple Gaussian clusters, is not statistically different from the performances of the comparative methods.*

Table presents the p-values generated by the Mann-Whitney test over the values of accuracy, precision and recall of the techniques, when naïve Bayes is used. The values in the table show that the hypothesis can be rejected.

	SC + NB	CS	DS
Accuracy	0.01341	**2.2E-16**	**2.2E-16**
Precision	**3.7E-8**	0.18	**2.603E-8**
Recall	**3.7E-8**	**0.3965**	**0.0025**

Table 19: p-values of the framework with naïve Bayes and the comparative for the measures of accuracy, precision and recall (the single-classifier variant also uses naïve Bayes).

The results of the Mann-Whitney test of the hypothesis related to the naïve Bayes implementations, shows that the framework and the single classifier perform similary in terms of accuracy and recall, but their values of precision are different. The accuracy of the single classifier (SC-NO) is similar to that of the the framework, as well as the precision of CluStream. Concerning the clustering techniques, the hypothesis can be rejected for all the performance measures.

	SC + SVM	CS	DS
Accuracy	0.03534	**2.2E-16**	**2.2E-16**
Precision	**7.82E-9**	**2.2E-16**	**2.2E-16**
Recall	0.6207	**2.2E-16**	**2.2E-16**

Table 20: p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variant also uses the SVM).

Figure 48 compares the performances of the techniques being analysed. The plot of the accuracy shows that the techniques that are able to classify the drifting data with high performance are the framework with the SVM (FW SVM), the single-classifier with the SVM (SC-NO SVM) and CluStream (CS). In fact, the 100^{th} percentiles of the distributions of these techniques equal 1. The number of runs of CluStream that end "successfully" are 13 out of 200. The framework and the single-classifier approach both show high accuracy in 93 runs. DenStream (DS), the framework (FW NB) and the single classifier that use naïve Bayes (SC NB) are not able to deal with the drift of this dataset in any of the runs. The plots of precision and recall highlight that the framework has higher precision, but a slightly lower recall than the single classifier. The other techniques have low performance for these measures.

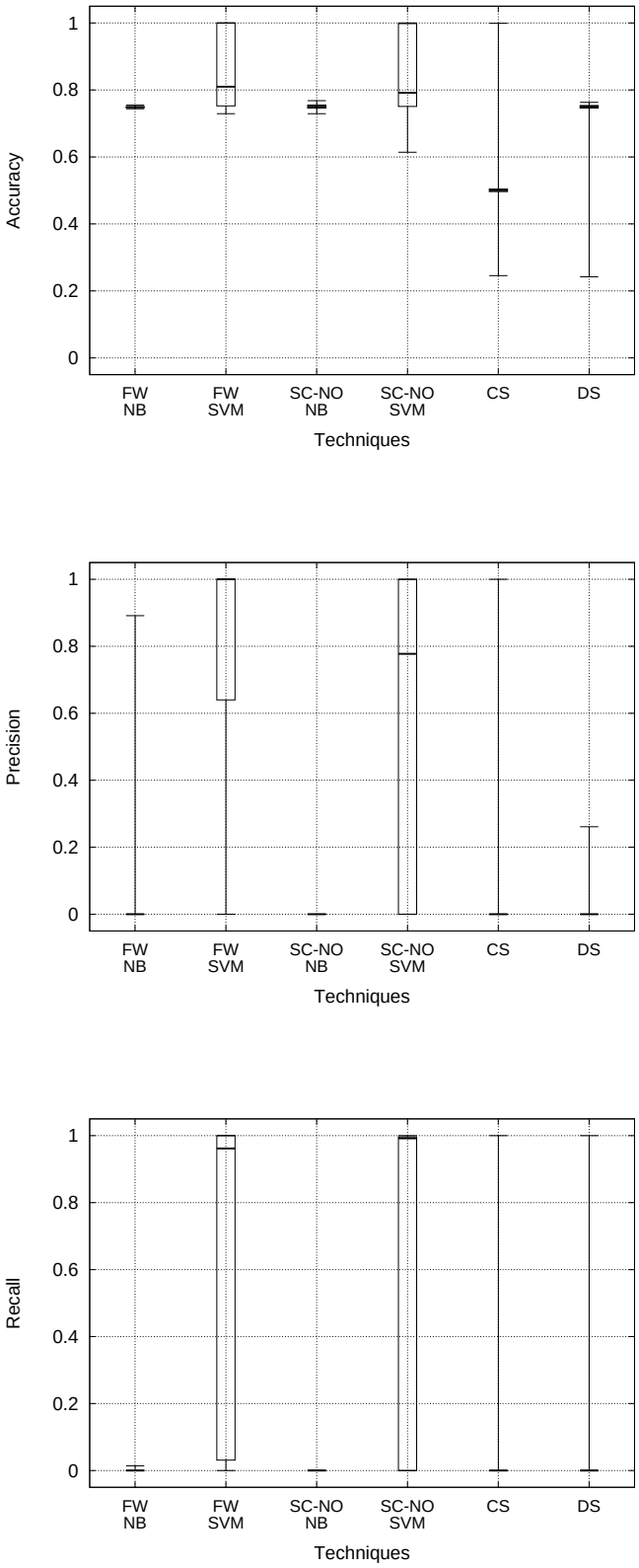


Figure 48: Distributions of the values of accuracy, precision and recall of the unsupervised techniques.

4.5.2 Preliminary investigation on the numbers of instances that are maintained in memory

This section presents a preliminary analysis of the memory requirements of the framework in terms of the number of instances it uses. In particular, the instance of the framework with the parameters of Table 17 is compared against an instance of the single-classifier technique. The analysis of the runs that were performed to test the hypothesis of Section 4.5 revealed that the framework maintains in memory, on average, 103 instances and the maximum number of instances that are stored is 310 (over the 200 runs). In fact, the number of instances processed by the framework varies across a run (it is determined by the parameter th_{online}). We want to verify if the single classifier can operate on the dataset with four Gaussian clusters without storing more than 310 instances for iteration. The comparison of the value of accuracy of the framework and those the single classifier with a window of size of 310 shows that the null hypothesis that these values of the two techniques are generated from the same distribution can be rejected with a confidence level of 0.995, with a p-value of 2.2E-16. Figure 49 shows the comparison of the performance in terms of accuracy of the single-classifier approach (using windows of different sizes) and the performance of the framework. That figure shows that single classifier technique cannot maintain high performance for window sizes smaller than 400. By contrast, the framework uses a smaller number of instances. This result could be explained by the fact that ensemble classifiers are able to learn a dataset incrementally. In fact, they can be used to learn large datasets, as described in Section 2.2.3). However, it should be noticed that the framework maintains multiple classifiers in memory, rather than a single one.

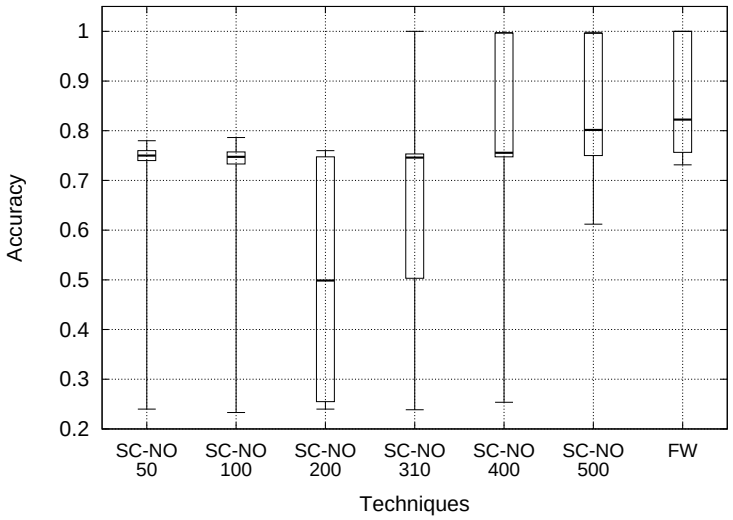


Figure 49: Distributions of the values of accuracy of several instances of the single-classifier technique (with different window sizes) and the framework.

4.6 Analysis of the inference of drift

This section presents a comparison between the mechanism of inference of drift being proposed, which is illustrated in Section 3.3.3, and two of the unsupervised methods presented in Section 2.4.1.5. Those are statistical tools that compare the similarity between two pools of data. They are: the Friedman-Rafsky test and a technique that makes use of supervised classification to detect change by Hido et al. [30, 37].

In order to provide a fair comparison, the two statistical methods have been integrated within the framework. In this context, the framework still uses the queue ($FIFO_{ID}$) to detect concept drift (the description of the drift inference scheme is provided in Section 3.3.3). In particular, a 1 is added to the ($FIFO_{ID}$) queue only if these techniques detect a change between the distribution of two pools of data. Of these pools, one is not updated unless drift is detected, the other is refreshed with new data at each iteration. When the computation starts, both pools contain the same instances from the training phase. At the end of the first iteration, the new data that is collected is copied into the second pool. At each iteration, the similarity of the two pools is tested. If they are different, according to the test being used, the first pool is updated with the instances of the second pool and a 1 is added to the FIFO queue. If the rate of 1s is higher than th_{ID}, the mature classifiers of the

framework are replaced with the naïve classifiers. If the pools are not different, the first pool does not change, while the other pool awaits for new data to replace the data it contains.

We measure the sensitivity and the specificity of the drift detection techniques being compared. A true positive (TP) is recorded when drift is ongoing and it is detected. If drift is not detected, a false negative (FN) is generated. In absence of drift, if the mechanism of detection is not triggered, a true negative (TN) is recorded. Otherwise, a false positive (FP) is recorded. We want to measure how sensitive and how specific the three detection mechanisms are. Sensitivity measures the ability to detect drift. It is expressed by the following formula:

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.4}$$

The denominator represents the total number of positives (drift is ongoing), expressed as the sum of those that are detected (TP) and those that are not (FN). Notice that the measure of sensitivity is equivalent to the measure of recall, defined in the Expression 4.3.

The specificity of a detection technique measures the number of false alarms in absence of drift. It is expressed as the fraction between the number of TN and the total number of negatives:

$$Specificity = \frac{TN}{TN + FP} \tag{4.5}$$

The delay between the start of a concept drift and its detection is also measured. It is expressed as the number of instances that are processed before drift is detected, divided by the total number of instances.

Separate tests with the techniques of naïve Bayes and the SVM are performed to evaluate the detection performances of the mechanism. In fact, different combinations of detection techniques and learners may affect the detection capabilities, as well as, the performance of the framework and the comparisons.

4.6.1 Dataset with Gaussian data

We test the framework on a variant of the two-Gaussians dataset of Section 4.2.1. For this experiment, the stream generates $1,000$ training instances, followed by $1,000,000$ unlabelled instances. Of these, only the second half (from instance $500,001$ to instance $1,000,000$) is affected by concept drift. At the end of a run,

the centers of the clusters are shifted by 0.5 units along the x-axis with respect to their initial positions, defined in Section 4.2.1. Table 21 shows the parameters of the framework for the different combinations of detection techniques and classification techniques. In fact, different techniques may require different configurations of the parameters of the framework in order to improve their detection performances.

Parameter	**Framework**		**Friedman-Rafsky**		**Hido et al.**	
	NB	SVM	NB	SVM	NB	SVM
$th_{training}$	90	65	90	90	90	90
th_{online}	100	100	200	100	200	100
$ratio_{NaiveMature}$	1.0	1.0	1.0	1.0	1.0	1.0
$FIFOsize_{ID}$	10	10	60	60	60	60
th_{ID}	0.5	0.4	0	0	0	0

Table 21: Parameters of the configurations of the framework for the different mechanisms of drift detection.

The inference of drift of the framework is tested against a framework that is not able to detect drift. The null hypothesis, in this case, is stated as:

> *The values of sensitivity, specificity and delay of detection of the mechanism of drift inference of the framework, when tested on the Gaussian dataset, are not statistically differerent from the values of the same measures generated by a framework that is not able to detect drift.*

The A-test, performed using naïve Bayes determined that performing 200 runs is sufficient for the measures of sensitivity and specificity, but not for delay. However, as shown in Figure 50 the A-test score of delay is in the medium region. Also because of time reasons, 200 runs are performed to test the hypothesis. The probabilities of the values of sensitivity, specificity and the detection delay of the mechanism of drift inference and those of its comparison being generated from the same distribution are respectively: 2.2E-16, 0.00293 and 2.2E-16. The null hypothesis can be rejected with a confidence level of 0.995. This indicates that drift is inferred.

4.6.2 Comparison of the mechanism of drift inference with alternative drift detection techniques

This Section evaluates the detection capabilites of the framework against the comparative methods. The parameters of the framework for the different techniques are

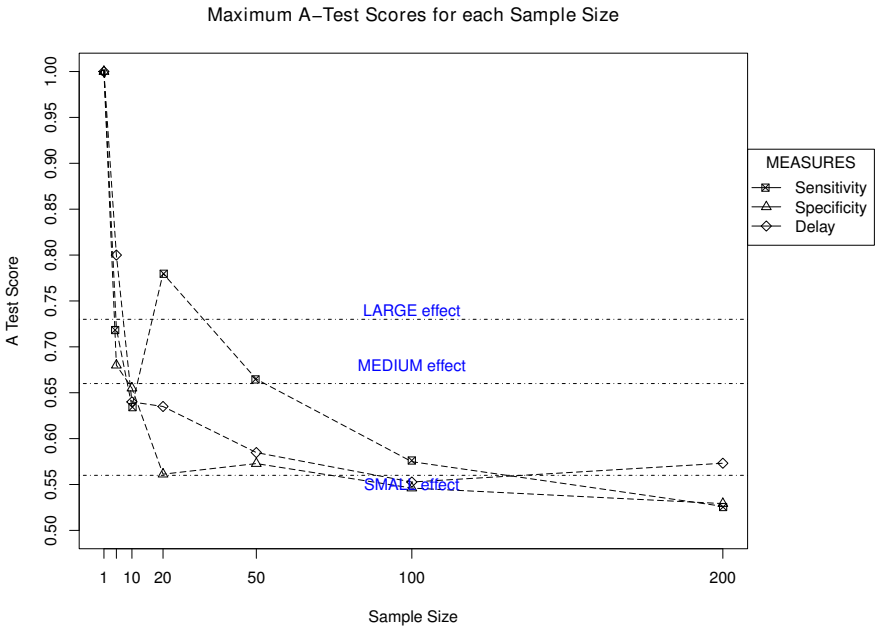


Figure 50: A-test results of the framework using naïve Bayes.

presented in Table 21. The parameter α of the method Friedman-Riefsky has value 0.998, while the parameter α of the method by Hido et al. has value 0.95. In order to evaluate the similarity between the techniques, the following hypothesis is stated:

> *The values of sensitivity, specificity and delay of detection of the mechanism of drift inference of the framework that uses naïve Bayes, when tested on the dataset with Gaussian distribution, are not statistically different from the values of the same measures generated by the instances of the framework that use the comparative detection techniques.*

Table 22 presents the p-values of the frameworks that use naïve Bayes, while Table 23 shows those of the instances that use the SVM:

	H + NB	FR + NB
Sensitivity	**2.2E-16**	**2.2E-16**
Specificity	0.0946	**2.2E-16**
Delay	**2.2E-16**	**7.196E-16**

Table 22: p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use naïve Bayes).

	H + SVM	**FR + SVM**
Sensitivity	**2.2E-16**	**2.2E-16**
Specificity	**4.76E-5**	**2.2E-16**
Delay	**2.2E-16**	**8.939E-16**

Table 23: p-values of the framework and the comparative with the SVM for the measures of accuracy, precision and recall.

The tables show that, apart from the similarity of the values of specificity generated by the inference of drift and the the method Friedman-Rafsky, all the other p-values are under the level of significance that was set. Figure 51 depicts the performances of the techniques being tested. The plot of the sensitivity shows that the framework with the SVM (FW SVM) and the technique by Hido et al. with naïve Bayes (H NB) have lower values of sensitivity than the other techniques. A low sensitivity may be acceptable as long as it guarantees a sufficient rate of updating of the model. The specificity plot indicates that the framework with the SVM, Hido with the SVM and Friedman-Rafsky with NB (FR NB) are the techniques that generate the lowest numbers of false positives. Moreover, the inference of drift of the framework is the mechanism with the lowest median delays of activation. However, the 100^{th} percentile of the framwork with the SVM shows that the drift is not detected in a few cases.

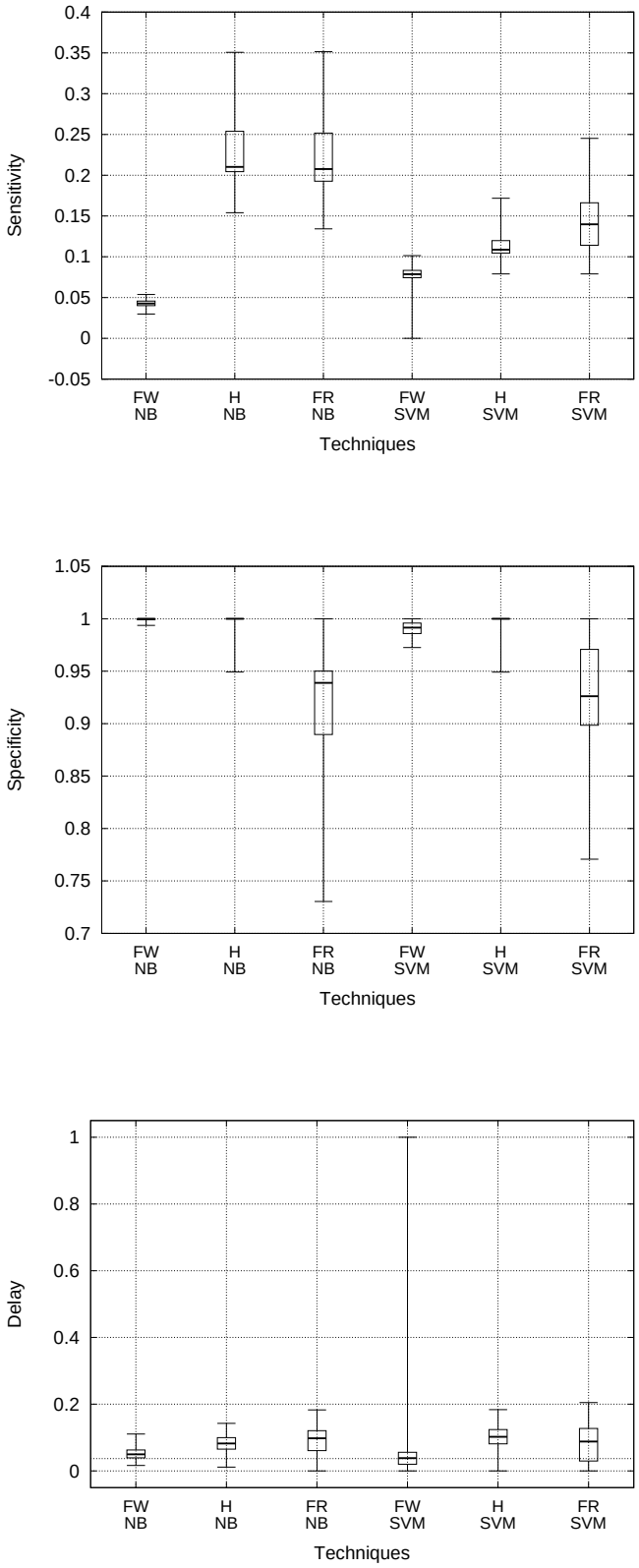


Figure 51: Distributions of the values of sensitivity, specificity and delay of detection of the unsupervised techniques.

Figure 52 shows the classification performances of the techniques. When Naive Bayes is used, the instances of the framework have high performance on this dataset. This, may be related to that particular classification technique, as hypothesised in Section 4.3.2. Concerning the instances of the framework with the SVM, the inference of drift provides better classification accuracy than the other frameworks, on average. However, the distributions of the performance measures indicate that all the techniques using the SVM are not able to deal with the concept drift of this dataset for part of their runs (two out of two hundred runs).

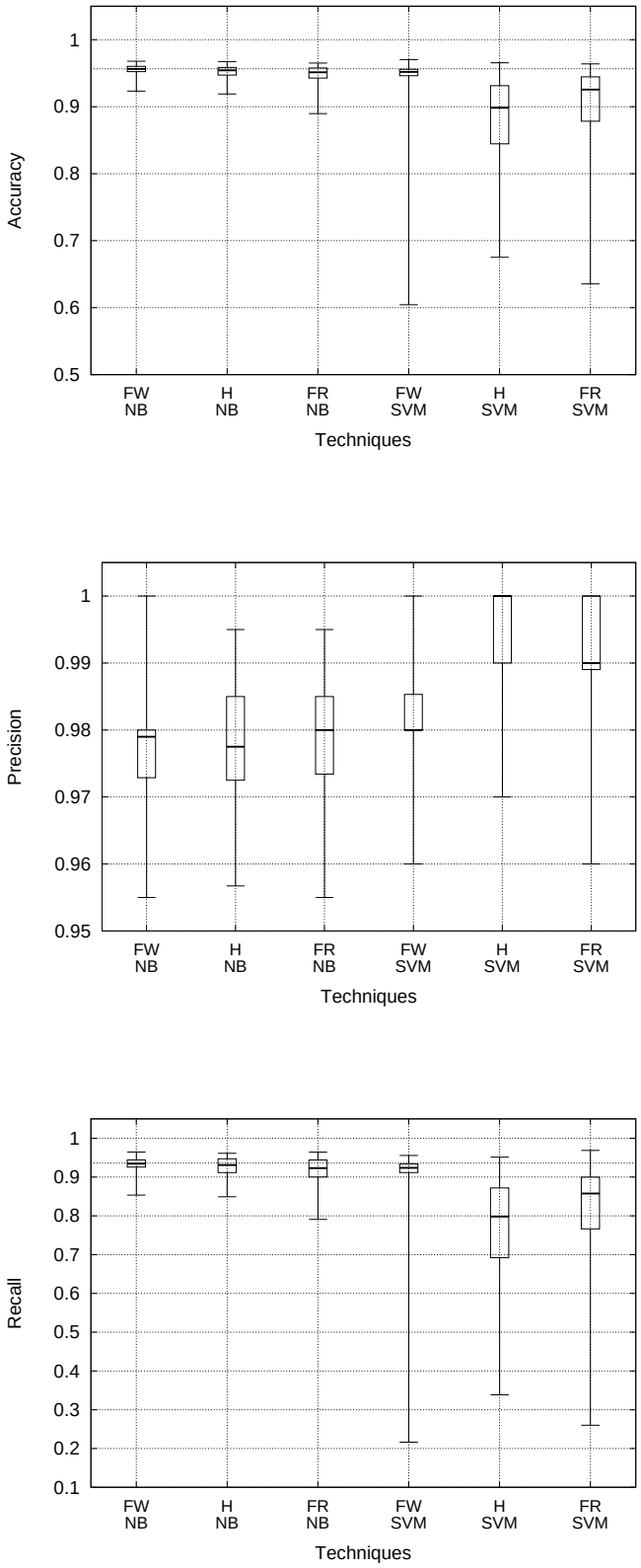


Figure 52: Distributions of the values of accuracy, precision and recall of the unsupervised techniques.

4.6.3 Dataset with separable classes and uniform distributions

This experiment tests the framework on data consisting of two classes with uniform distributions. In particular, each class has the shape of a square with an edge of 1.0. The number of training instances is 500, while the number of unlabelled instances is 1,000,000. The center of the distribution of class " + 1", which is fixed, has coordinates $(2, 1)$. The initial position of the center of the distribution of the other class is determined by the coordinates $(3, 1)$. The first quarter of the dataset (250,000 instances) is not affected by concept drift. The second quarter of the dataset (from instances 250,001 to 500,000) is affected by a concept drift that moves the center of the distribution of class " − 1" to the position $(4, 1)$. Starting from instance 500,001 the same distribution moves in the opposite direction, and it reaches the final position of $(2, 1)$ at the end of a run. Figure 53 illustrates the stages of the concept drift of this dataset. The values of the parameters of the framework are shown in Table 24.

Parameter	**Framework**		**Friedman-Rafsky**		**Hido et al.**	
	NB	SVM	NB	SVM	NB	SVM
$th_{training}$	45	45	45	45	45	45
th_{online}	400	100	100	100	100	100
$ratio_{NaiveMature}$	1.0	1.0	1.0	1.0	1.0	1.0
$FIFOsize_{ID}$	40	40	60	60	60	60
th_{ID}	0	0	0	0	0	0

Table 24: Parameters of the configurations of the framework for the different mechanisms of drift detection.

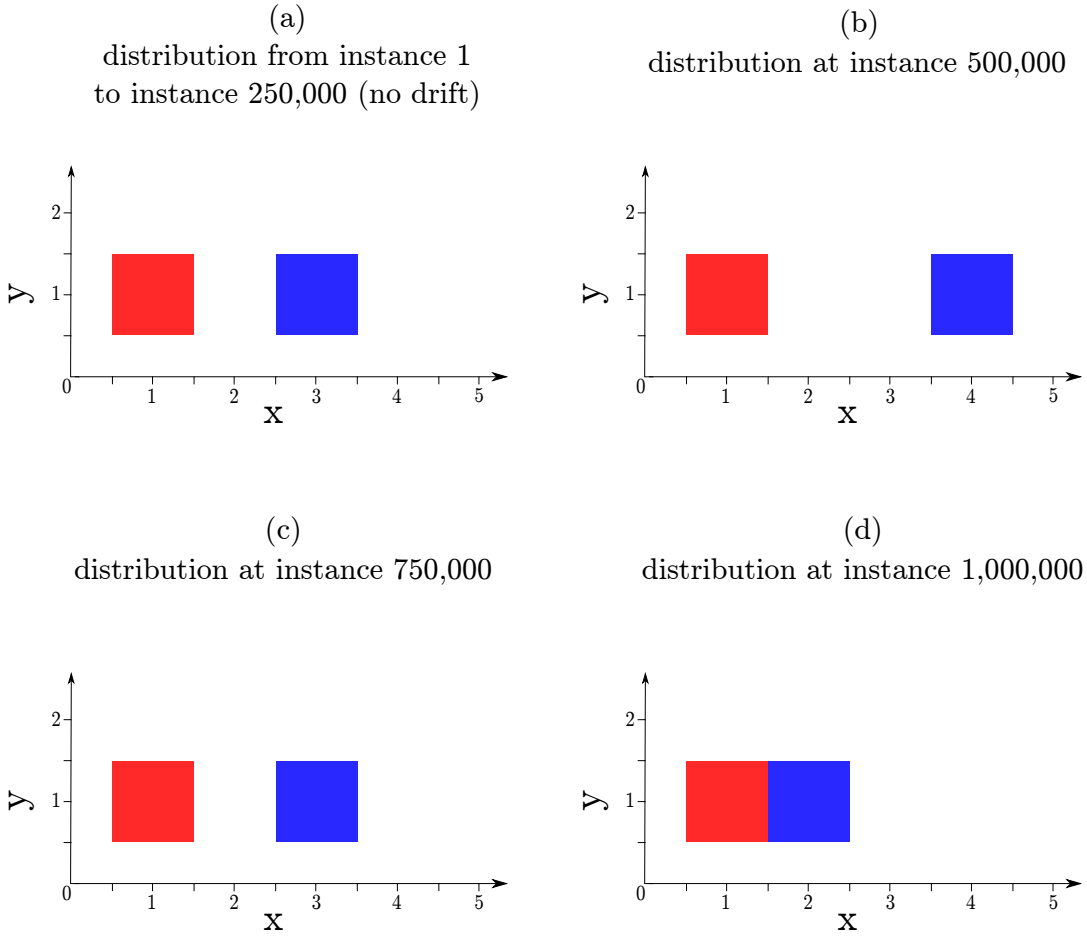


Figure 53: Concept drift of the dataset with uniformly distributed classes. Red is used to indicate the distribution of the class of positive instances, while blue is used for the negative class. The data distribution does not change for the first quarter of the number of instances, as shown in the diagram (a). After that, the center of the negative distribution moves to position (4,1), as shown in diagram (b). The center of negative instances changes direction in the second half of the concept drift (diagram (c)), and reaches the final position depicted by diagram (d).

The inference of drift of the framework is tested against an instance of the framework that does not infer concept drift. The null hypothesis, in this case, is stated as:

> *The performance of the mechanism of drift inference, when tested on the Gaussian dataset, is not statistically differerent from the performance of an instance of the framework that is not able to detect drift.*

The A-test, performed using naïve Bayes determined that performing 200 runs is sufficient for all the measures, as shown in Figure 54. The probabilities of the values of sensitivity, specificity and detection delay of the mechanism of drift inference and

those of its comparison being generated from a common distribution are respectively: 2.2E-16, 2.2E-16 and 2.2E-16. The null hypothesis can be rejected with a confidence level of 0.995. This shows that the performance of the inference of drift differs from the performance of a framework that cannot detect drift.

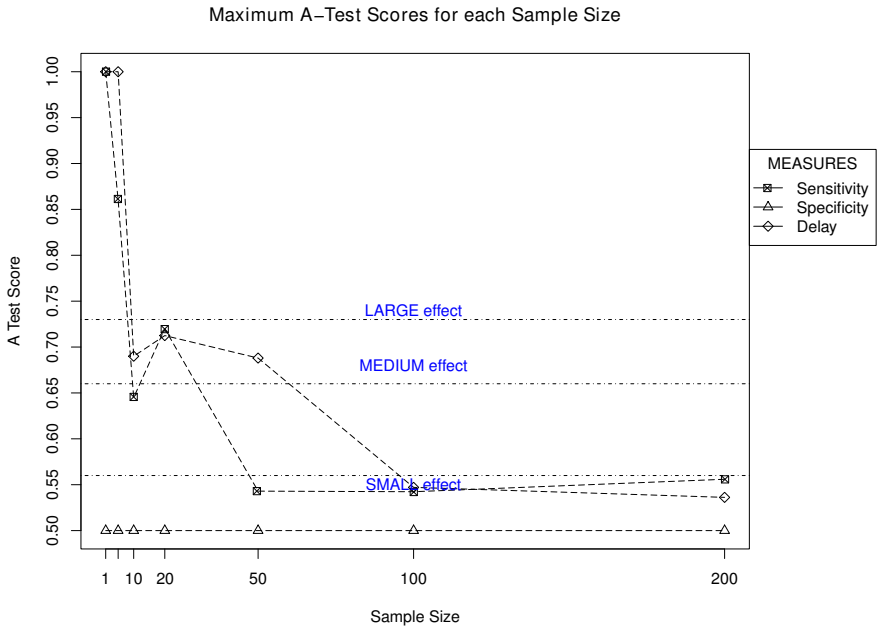


Figure 54: A-test results of the framework using naïve Bayes.

4.6.4 Comparison of the mechanism of drift inference with the unsupervised drift detection techniques

The performance of the inference of drift is compared against the performances of the other unsupervised methods. In order to evaluate the similarity between the techniques, the hypothesis is stated as:

> *The values of sensitivity, specificity and delay of detection of the mechanism of drift inference of the framework that uses naïve Bayes, when tested on the dataset with separable classes, are not statistically different from the same values generated by the instances of the framework that use the comparative detection techniques.*

The values of the parameters of the framework and the comparative methods are shown in Table 24. Table 25 shows the p-values measuring the similarity between the performance values generated by the techniques when naïve Bayes is used. The

same values when the SVM are used are displayed in Table 26. According to the p-values of both tables, the null hypothesis can be rejected with a confidence level of 0.995 for all the comparative methods and for both classification techniques. In addition, the magnitudes of the p-values denote that the way in which the inference of drift operates differs considerably from the other techniques.

	H + NB	FR + NB
Sensitivity	**2.2E-16**	**2.2E-16**
Specificity	**1.41E-12**	**5.709E-6**
Delay	**2.2E-16**	**2.2E-16**

Table 25: p-values of the framework with the SVM and the comparative for the measures of accuracy, precision and recall (the single-classifier variants also use naïve Bayes).

	H + SVM	FR + SVM
Sensitivity	**2.2E-16**	**2.2E-16**
Specificity	**5.709E-6**	**5.709E-6**
Delay	**2.2E-16**	**2.2E-16**

Table 26: p-values of the framework and the comparative with the SVM for the measures of accuracy, precision and recall.

Figure 55 shows the performance of the techniques being tested. The plot at the top of the figure highlights that sensitivity of the inference of drift of the framework (FW NB and FW SVM) is lower that those of the other drift detection methods. That is caused by the fact that the mechanism of drift inference starts detecting drift after that 85% of the instances have been processed. However, drift starts at 25% of the total number of instances. The other techniques detect drift at about 30% of the number of instances. Therefore, for a good part of the duration of the concept drift, the inference of drift is not activated. This causes the low sensitivity observed in the plot at the top of Figure 55. By contrast, the specificity of the same mechanism is higher than the values of the specificity of the other techniques, as it equals 1 for both classification techniques. That means that mechanism of drift inference does not generate false detections.

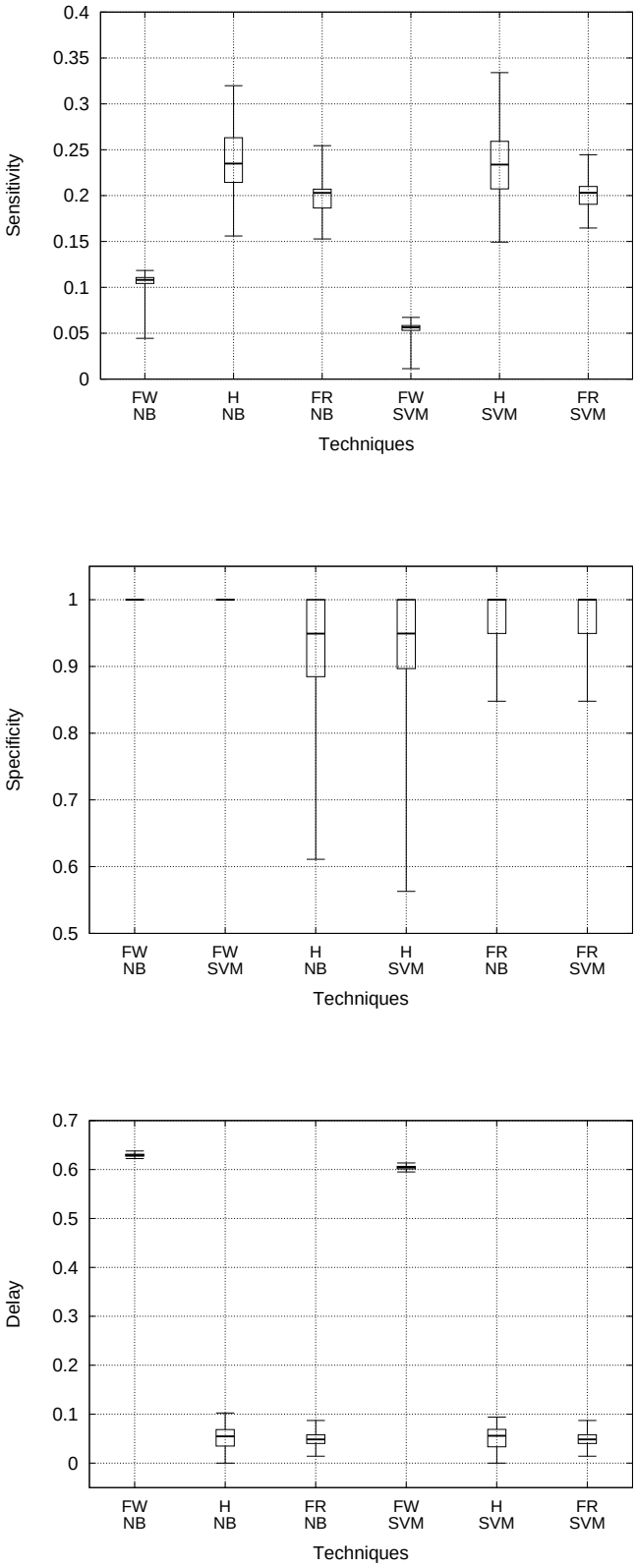


Figure 55: Distributions of the values of sensitivity, specificity and delay of detection of the unsupervised techniques.

Figure 56 shows the classification performances of the techniques. In particular, the distributions of values depicted by the plots of Figure 56, highlights that the instances of the framework with naïve Bayes and the SVM perform better than the other techniques for the measures of accuracy and precision. The majority of the values of accuracy generated by the techniques equal 1, as shown in Figure 56 (top plot).

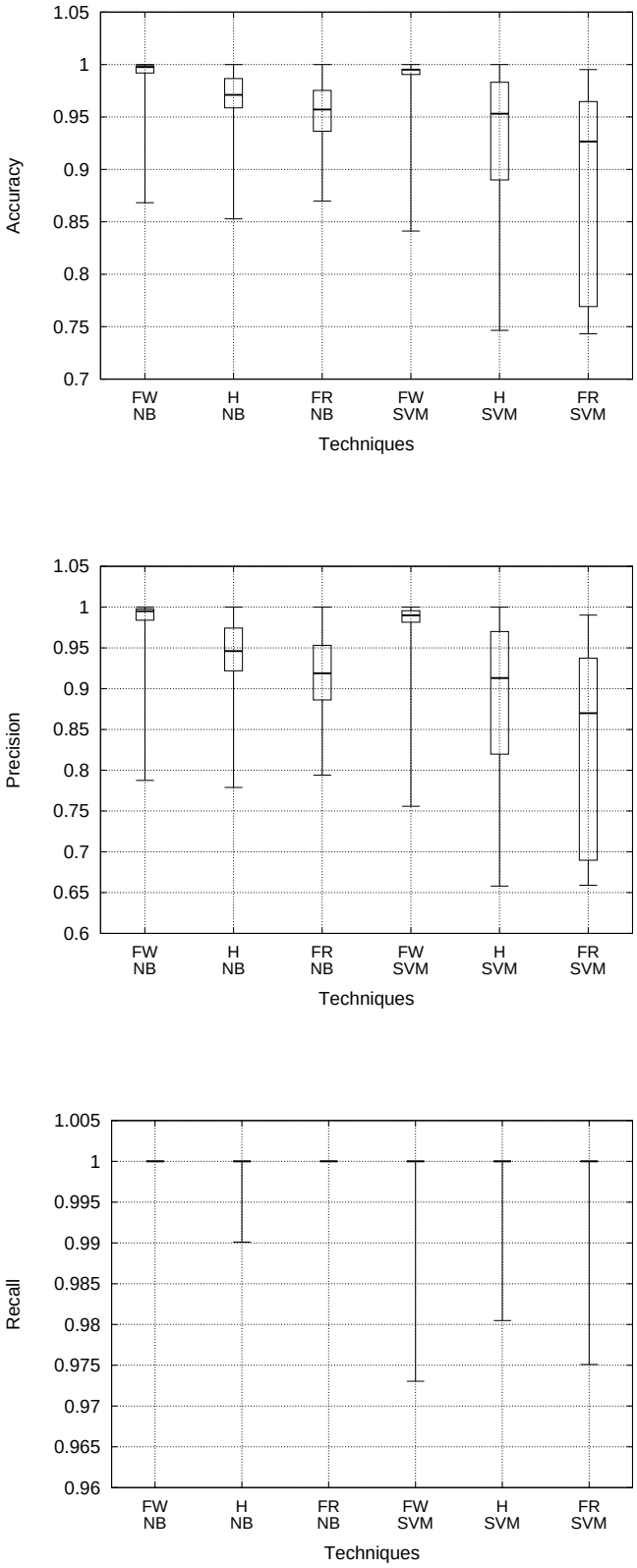


Figure 56: Distributions of the values of accuracy, precision and recall of the unsupervised techniques.

The plots of Figure 57 show the values of accuracy and the detection of drift of the three techniques for one of the runs that uses the SVM. Drift starts at 25% of completion of a run. The method Friedman-Rafsky detects drift at about 35% of the dataset. At that point mature classifiers are replaced with naïve classifiers. However, the accuracy of the framework with the Friedman-Rafsky technique does not change. In fact, the updated model is able to classify the instances correctly, as well as the older model did before it was replaced. The method by Hido et al. performs similarly. However, for this run, it is triggered before drift starts, thus generating a value of specificity smaller than 1 (it equals 0.82233). By contrast, the inference of drift of the standard framework is not activated. In fact, in the part without drift, the boundaries of its models lie in a region with a null density of instances, as shown in Figure 58, case (a). In that figure, the boundaries of the mature models are represented using red lines, while the green lines represent the boundaries of the naïve classifiers. The classes are separable and, as a consequence, the classifiers generate the same decisions. Since the are no discordancies among the classifiers, the Hamming distances between their decisions d_M, d_N and d_{MN} should equal 0. When 50% of the instances have been processed, the distribution of the negative class has drifted to the position (1,3). At that point, the boundaries of the mature classifiers have not changed, while the boundaries of the naïve classifiers lie between the two classes, as shown in the part (b) of Figure 58. This implies that, given an instance, all the classifiers generate the same decisions. Then, the distribution of the negative class starts moving in the opposite direction, until it reaches the position that is depicted in part (c) of the same figure. Some points of class " -1" cause discordancies between the mature classifiers, as highlighted by the plot at the bottom of Figure 59. As a consequence, the distances d_M and d_{MN} increase, while d_N does not change. Notice that, at that point, the accuracy of the framework has not started decreasing yet. The arrow points at a spike of d_{MN} that is higher than both d_M and d_N. That triggers the replacement of mature classifiers with naïve classifiers, which is followed by an increase of the accuracy. Figure 58 shows the boundaries of the models and the data at the end of the run.

Finally, the framework processes information generated by the models (their decisions) to infer the presence of drift. The framework is able to detect quickly a change in that information and take actions to avoid performance degradation, as shown in Figure 59. That generates higher classification performances for this

dataset according to Figure 56. On the contrary, the two comparisons use the input vector, representative of $p(x)$, to detect concept drift. In this comparison, suffered worse performance degradations with respect to the framework with the inference of drift, according to Figure 57. The intermittent replacement of classifiers that can be observed in that figure, is responsible for not optimal classification performance (Figure 56). However, the two statistical tests are able to detect changes in the distribution that the mechanism of drift inference is not able to detect. To be more precise, drift cannot be inferred by the framework if the distributions of the classes are separable. However, for this dataset, those type of concept drift do not cause performance degradation of the framework, as it can be observed in Figure 57.

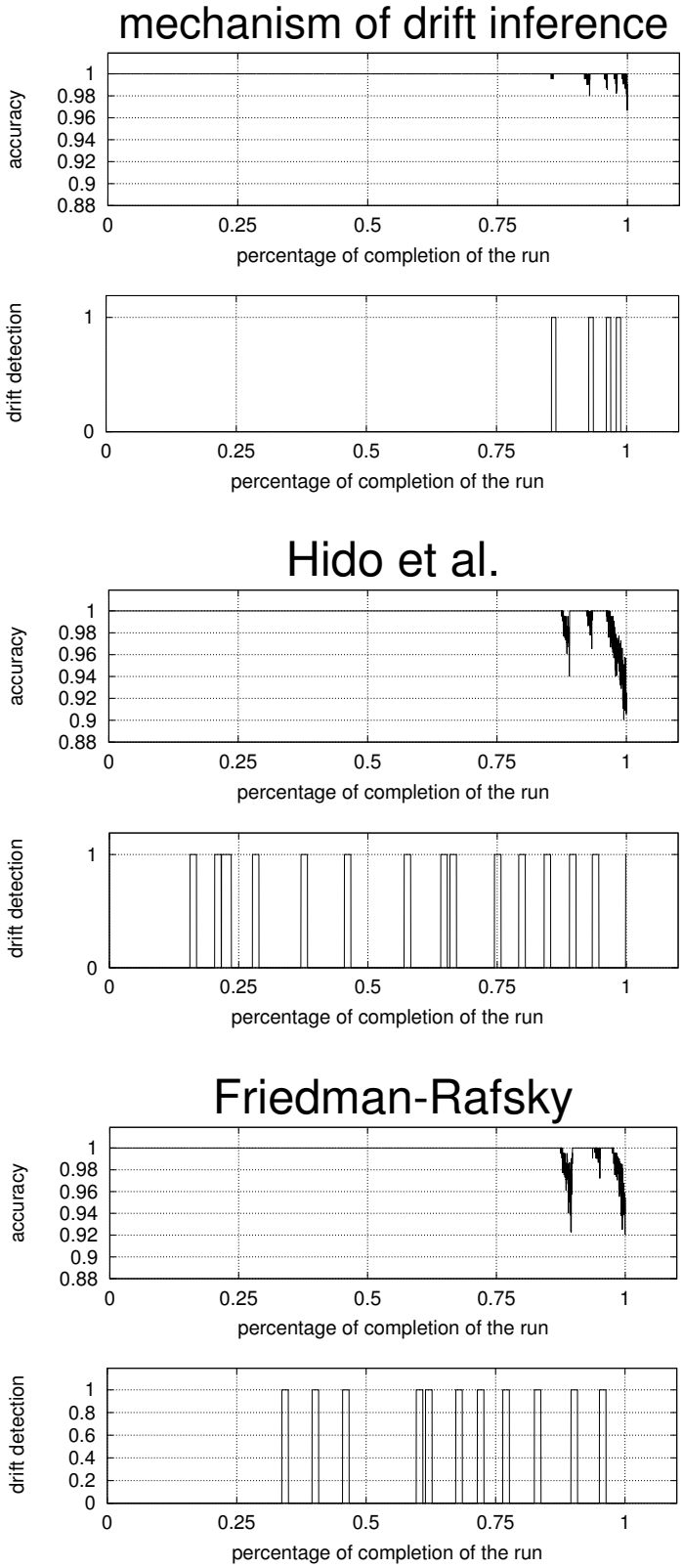


Figure 57: Plots of the accuracy and plots of the detection of drift of the version of the framework with the mechanism of drift inference (top), of the framework with the test by Hido et al. (center) and of the framework with the Friedman-Rafsky test (bottom).

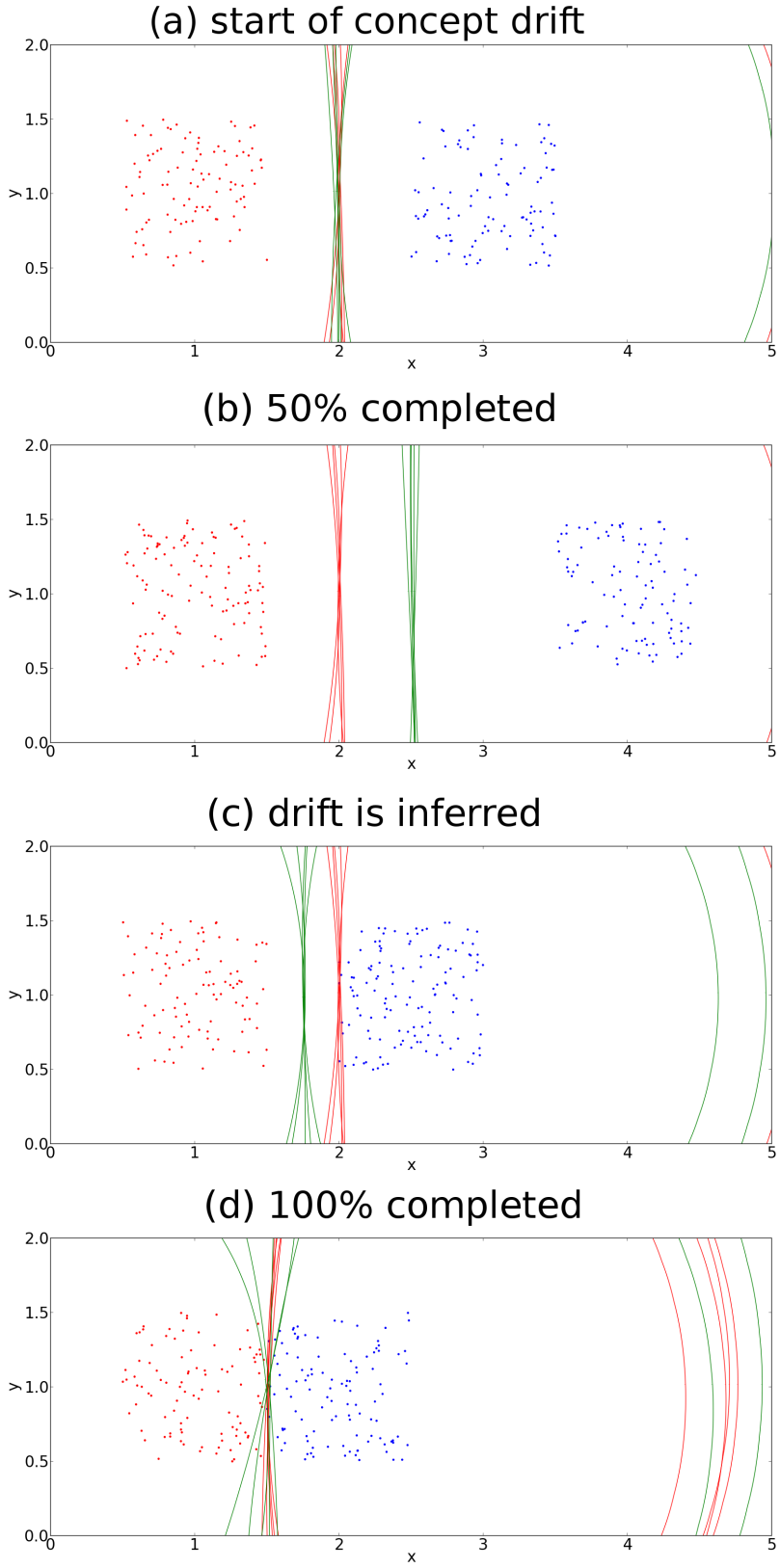


Figure 58: Plots of actual data and boundaries of the classifiers. The red points represent the positive instances, blue is used for the negative instance. The red lines represents the boundaries of the mature classifiers and the green lines represent the naïve classifiers.

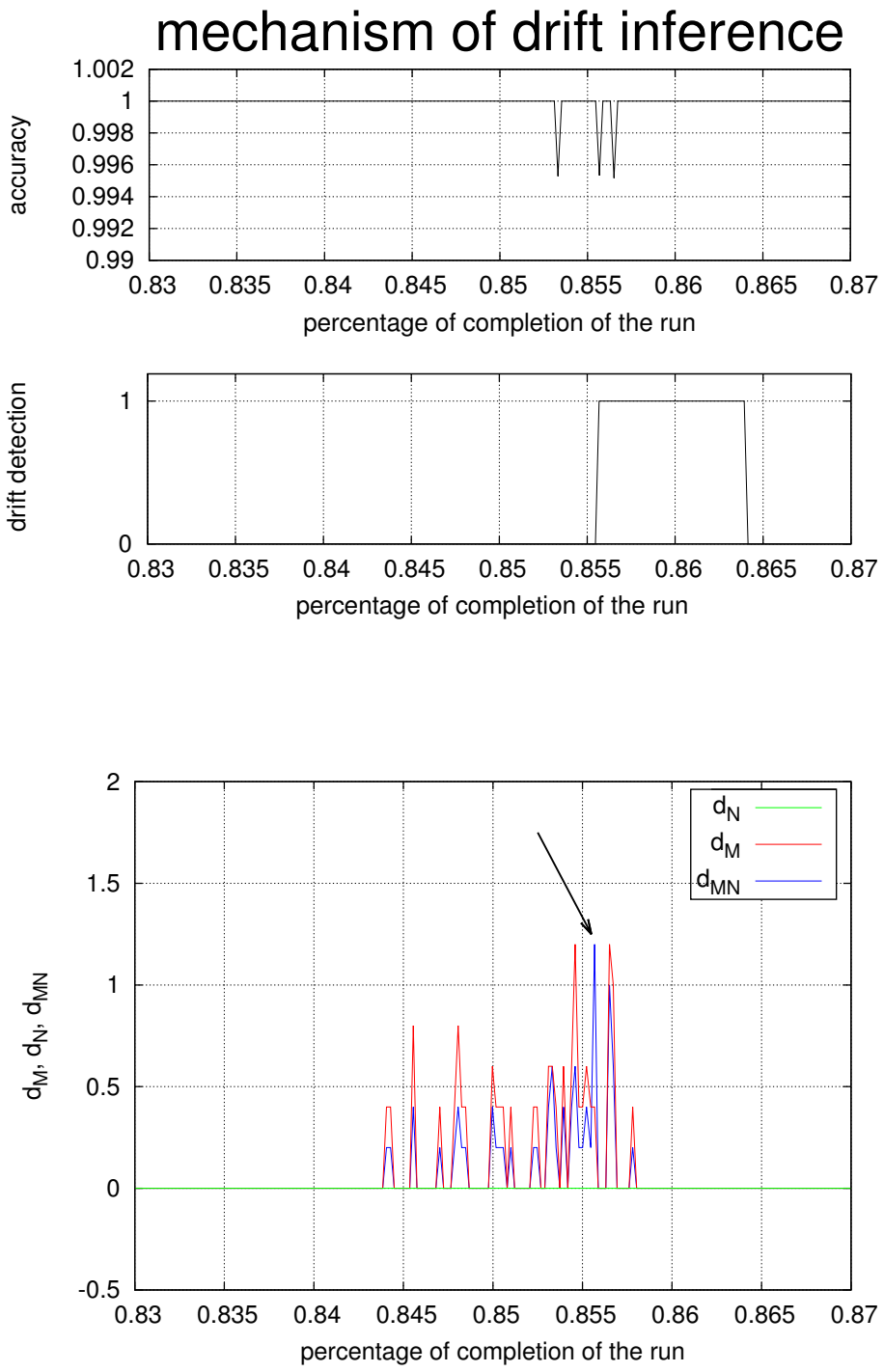


Figure 59: First detection of drift. The two plots at the top are enlargements of the plots at the top of Figure 57. The plot at the bottom shows the mean distances between the mature classifiers d_M, between the na"ive classifiers d_M, and those between the mature and the naïve classifiers. The arrow highlights a spike of d_{MN} that is higher than d_M and d_N. That triggers the updating of the classification model, and the subsequent increment of the accuracy.

4.7 Summary

This chapter presented the experiments that have been performed to evaluate the capability of the framework of dealing with different data distribution, different classification techniques.

The experiments of Section 4.2 revealed that the framework is able to maintain high classification performance on data affected by concept drift. However, as it was hypothesised in Section 3.2, the same results were not observed when a dataset with concept drift but fixed $p(x)$ was used.

Further experiments highlighted that the capability of dealing with drift is maintained when different classification techniques are used. They were presented in Section 4.3. In particular, the framework is able to operate with naïve Bayes, the SVM and AISEC. However, the results are different when it uses the algorithm C4.5 and the MLP with a single internal node.

The framework was tested on a large selection of classification techniques. The experiment involving naïve Bayes (Section 4.3.2) and AISEC (Section 4.3.1) showed that the framework is able to operate with classification techniques with very different characteristics. The interpretation of the results and the sensitivity analysis highlighted that the mechanisms that allow the framework to classify drifting data depend on the classification technique that is used. In particular, the interpretation of Section 4.2.1.2 presumes that the gradient of the data plays an important role in the updating of the model of the framework when the SVM are used. Concerning AISEC, its detectors tend to concentrate in the regions with high density of instances. Further experiments revealed that, with naïve Bayes, the framework shows high performance regardless of the parameters that are used. That is probably due to this classification technique being particularly suited to the distribution of this dataset. Further experiments showed that the framework is not able to deal with the algorithm C4.5 (Section 4.3.5), or with the algorithm MLP with a single internal node (Section 4.3.3).

Section 4.4 compared the framework with a selection of unsupervised techniques on a dataset with Gaussian distribution. The comparative techniques are a single classifier that retrains its model and two clustering methods. The techniques showed similar performances on that dataset, apart from the clustering algorithm DenStream and the single classifier with density adaptive-forgetting did not manage

to provide high performance on all the runs.

The experiment of Section 4.5 analyses the framework on a data with multiple patterns of concept drift. In particular, this is a hard test for unsupervised model updating because of the high overlap and the changing characteristics of the classes. The framework and the single-classifier algorithm are the only techniques that are able to classify correctly the data of this dataset. That happens in about half of the runs and only when the SVM are used. In addition, the framework stores and processes a quantity of instances that is smaller than that of the algorithm with a single classifier, that was shown in Section 4.5.2. However, the computational cost associated with the framework is higher since it uses multiple classifiers. Both clustering algorithms have low performance on this dataset.

The experiments of Section 4.6 evaluated the mechanism of inference of drift of the framework, and compared it against two techniques that use different principles of detection. The first experiment employed nonseparable data, while the second dataset used separable classes. Concerning the first experiment (Section 4.6.1), the framework shows a smaller detection time and, when the SVM are used, also a higher specificity then its comparisons. In addition, on average, the use of the framework leads to higher classification performance with respect to the other techniques. However, for two of the runs involving the SVM, the framework was not able to reveal the presence of drift. That may be related to the set of parameters that were used. Concerning the experiment with separable classes (Section 4.6.3), the mechanism of drift inference was not triggered for part of the concept drift of that dataset, while the comparatives methods were able to detect it. However, that did not cause a decrease of the classification performance of the framework. On the contrary, the general classification performance of the framework over that distribution are higher than those provided by the other detection techniques. In particular, since the framework compares the output decisions of its classifiers, if there is a change in the patterns of information that are extracted, it is able to detect it and that allows for quick adaptation and, as a consequence, higher performance. The comparative methods do not use information about the output as they operate on the input features.

Chapter 5
Conclusions

5.1 Introduction

This chapter provides a summary of the thesis, by giving an outline of the problem and our contribution to the problem, an adaptive framework. It briefly describes the two novelties of the implementation of the framework we propose, feedback of the decisions and the mechanism of drift inference. Moreover, it proposes an overview and an interpretation of the results of the experiments.

This chapter also suggests some ideas on how to develop the framework. These regard parametric adaptivity, which is noted as future work, and the extension of structural adaptivity to the other components of the framework, especially towards the feature extraction level. Moreover, it highlights the need to revisit the inference of drift mechanism, investigate the co-existence of different techniques and extend the concept of feedback also to the other components of the framework.

5.2 Review of the research problem and the solution

The problem of classification consists of establishing a model by combining data potentially from multiple and different sources and labels that state the classes the data belong to. A model is a function that associates labels to unlabelled data, and is used to classify instances for which the true classes are unknown.

A problem that affects many applications, in particular real-world ones, is that the distribution of the data can vary. This might be caused by a change in the system being observed, or for example by a change in the data collection process. This problem is known as concept drift and requires the model to be updated to

avoid performance degradation. The classification algorithms that deal with concept drift employ the same mechanism to classify drifting data. They use information about the true class of an instance, which is provided after this has been classified, to modify the model and minimise performance loss.

Due to the large amounts of data that are collected nowadays, the problem of generating the correct labels is becoming a bottleneck for many applications [59]. In fact, labelling is generally costly and time consuming as, often, it is done manually. In many cases, providing labels at the rate at which data is presented is not feasible and researchers have started facing the problem by proposing algorithms that require a limited amount of labelling [47, 59, 103] or no labelled data altogether [78].

We suggest that classification algorithms should become more autonomous, and the amount of supervision they require should decrease. With that pupose we analysed the conditions that are required in order to perform unsupervised model updating. As part of our research, we investigated the ideas of the adaptive framework [51]. That framework proposes a modular structure which can be instantiated at different levels: feature extraction, classification or decision fusion. It consists of a set of stovepipes and a data fusion module. A stovepipe contains a technique, depending on the level, that can be a feature extraction technique, a classifier or a technique to fuse decisions. Multiple stovepipes process the input data in parallel. The function of a data fusion module is to combine the data generated from the stovepipes to produce a single output. For instance, data fusion may involve the selection of the output of a particular stovepipe, the combination of the outputs of the stovepipes (e.g., through voting or averaging). When instantiated, stovepipes and a data fusion module form a box. This framework also accounts for the reuse of the output of a box as an input to the box itself or another box at a lower level. This idea of feedback as well as the whole framework had not been implemented at that time.

Starting from the ideas outlined in [51], we analysed the forms of adaptivity the framework should perform. This lead to the identification of two different types of adaptivity: structural and parametric. Parametric adaptivity concerns the optimisation of the parameters of a stovepipe of a data fusion module, while structural adaptivity is concerned with the creation, the deletion and the replacement of a component of the framework.

In particular, our investigation of these concepts is limited to the structural

adaptivity of the classification level of the framework. More precisely, we developed our own implementation of a classification box in order to handle concept drift. This has a fixed number of classifiers and majority voting is the technique used to fuse their decisions. This is not different from any other ensemble classifier for concept drift. However, in our implementation the classifiers are divided into mature and naive, where only mature classifiers participate in the voting, while naïve classifiers are candidate to become mature. The box represents the basis on which the two novelties of this thesis were developed. They are: a mechanism of feedback of the decisions of the box and a mechanism of drift inference.

Our implementation of feedback consists of combining unlabelled testing instances and labels from the voting to train a new naïve stovepipe. Notice that, thanks to this technique, supervision is not required as the labels to update the model are generated by the voting.

Many drift detection mechanisms monitor the error rate, which requires the true classes of the data to be known. They could not be used in our case, as the framework does not make use of supervision. Other techniques use the input data without labels to detect drift. We pursued a different approach to drift detection. In fact, the mechanism of drift inference compares the similarity between the decisions taken by the mature classifiers and those by the naïve ones. In fact, if the distribution is stationary, as naïve classifiers are trained from labels generated by mature classifiers, the decisions of the two different types of classifiers tend to be similar. If a change of the distribution modifies the class-unconditional pdf, it will affect the model of naïve classifiers. Therefore, these will differ from mature ones. This is the signal that triggers the inference of drift.

5.3 Evaluation of the work

Several experiments were designed to test the capability of the implementation of the framework in dealing with concept drift. In particular, the experiments were aimed at identifying the distributions and the classification techniques on which the framework can operate. Furthermore, the performance of the framework was measured against those of a set of comparative methods in order to highlight potential benefits and limitations of the framework. Although a small amount of supervised data is provided to establish the model of the framework, no supervision

is used to update its model.

The experiments revealed that the framework is able to maintain high classification performance on data affected by concept drift. This capability is observed for datasets with concept drift and changing class-unconditional distributions, even though for some datasets the framework did not show high performance on all runs. According to our observations, this ability is lost when the class-unconditional distribution is not affected by concept drift. On the contrary, supervised techniques are able to classify the same data. Although, more experiments should be performed, the results we have obtained provide a positive answer to the question *"is it possible to maintain high classification performance on different types of data distributions?"*.

Further experiments highlighted that the capability of dealing with drift is maintained when different classification techniques are used. In particular, the framework is able to operate with naïve Bayes, SVM and AISEC. However, different results were noted for the technique C4.5 and the MLP with a single internal node. The analysis and the interpretation of the results seem to reveal that the principle of operation of the framework is related to the classification technique that is used. As a matter of fact, we were unable to develop a rationale that was valid for different classification techniques. This provides a positive answers to the research question *"is the framework able to operate with different types of classification techniques?"*. However, this ability is not maintained for all techniques that were used.

Unsupervised comparative methods were employed with the purpose of determining if the framework offers advantages in respect of those techniques. On the tests that were performed, the framework and a method that uses a single classifier provided higher performances than the other techniques. Although, it is not possible to give a positive answer to the research question *"does the framework offer any advantage in terms of performance with respect to comparative techniques?"*, the results of our experiments showed that its performance is equivalent to the performance of the single-classifier technique. However, preliminary results revealed that the framework seem to use less memory that the comparative techniques.

The mechanism of drift inference was evaluated on datasets with different characteristics and different learners, in order provide an answer to the research questions: *"is the mechanism of drift inference of the framework able to reveal the presence of drift?"*, *"is the mechanism of drift inference of the framework capable of operating with different classification techniques and on different types of distributions?"* and

"does the mechanism of inference of drift offer any advantage in terms of performance with respect to comparative techniques?". Based on the experiments that were performed, a distinction should be made. When the classes were nonseparable, the framework was able to detect drift with different classification techniques. However, when the distributions of the classes are separable, the mechanism of drift inference was not able to detect the presence of drift. However, it reacted to react to performance degradation. This gives a positive answer to the first question, but the ability of the mechanism of drift inference to reveal the presence of drift depends on the characteristics of the data that is used. Moreover, a positive answer can be given to the second question concerning the types of classification techniques it can operate with. Concerning the datasets, more tests should be performed. The inference of drift was compared with a set of unsupervised techniques for drift detection. The results show that, when the data is nonseparable, the framework is able to detect drift more quickly and with lower false detections than the comparative methods. When the classes are separable, although the inference of drift is not able to detect the start of concept drift, it is able to reveal the start of performance degradation quickly and with high sensitivity. That allows prompt update of the model of a classifier, thus leading to higher classification performance than the performances observed for the comparative methods. This gives a positive answer to the third question. However, it also suffers from the limitation of not being able to operate on separable classes.

5.4 Future directions

Although the implementation of the proposed framework shows that supervision is not a prerequisite for the classification of drifting data, we believe that the true potential of the architecture of the framework has not been expressed yet. We hope that the solutions proposed so far represent only an early stage of the development of the framework. For this reason, we suggest some research directions that would be interesting to investigate.

5.4.1 Inference of drift

The mechanism of drift inference monitors the discordancies between each pair of classifiers. It does so by maintaining a matrix M, whose rows and columns represent

classifiers. The cell M_{ij} contains the Hamming distance between the decision of the classifiers i and j in the current iteration. The inference of drift mechanism uses the matrix to calculate the average distance between naïve and mature classifiers, the average distance between mature classifiers and the average distance between naïve classifiers. If the first distance is higher than the other two for a certain number of nonconsecutive iterations then drift is inferred. We believe that much more information can be extracted from the matrix in order to infer drift.

The information in the matrix can be represented as a network, in which nodes are associated with classifiers and the edges that connect them are labelled with the Hamming distance between the decisions of the classifiers. From this perspective, by analysing characteristics of the network such as dispersion and internal connectivity it should be possible to gain a better understanding of the inference of drift. In such a way, notions from network theory could be used [68], for example, to distinguish between the different types of drift, or reduce the delay between when the start of a drift and its inference.

The triggering nature of the mechanism, could also be reconsidered. We believe in fact that the mechanism should not behave like an "on-off" switch for updates of the model. Instead, the updating of the model should be a continuous function of the rate at which drift is occurring.

The distinction between naïve and mature classifiers which is required by the mechanism of drift inference may as well be reconsidered. Instead of maintaing two pools with diverse functions, the framework could have a single pool, perhaps of variable size. Drift would be inferred by comparing the behaviour of recent stovepipes with older ones. If recent stovepipes tend to form a cluster within the network, this could be interpreted as ongoing concept drift. In fact, if the distance between recent classifiers decreases and their distance from the other stovepipes increases, this could potentially mean that they are embodying information about concept drift. Moreover, every stovepipe would contribute to the output, even the newest ones, but with different weights.

5.4.2 Parametric adaptivity

The current implementation of the framework is not capable of parametric adaptivity. In fact, the parameters are set at training time and cannot be changed afterwards. However, replacing an ineffective classifier (or stovepipe) is useless if

the new one uses the same parameters. We envisage two solutions. The simplest one consists of selecting a set of values from a look-up table, according to the actual scenario. Another solution consists of applying dynamic optimisation to the parameters [14, 101]. Among the available techniques there are genetic algorithms and particle swarm optimisation [99, 100]. In this way, if the current parameters are no longer effective, it should be possible to modify them and therefore deal with situations not foreseeable at design time. Parametric adaptivity, however, should bebased on the estimation of the performance of its components. We believe the mechanism of drift inference, and perhaps the ideas about network theory previously introduced, represents a solid basis on which to develop an estimator for the performance of a component of the framework.

Adaptivity should consider the parameters of components such as stovepipes and data fusion modules. For example, classifiers or feature extraction techniques might have their parameters changed, even online where possible, or in the case of data fusion, a voting module might have its weights reconsidered. Adaptivity might as well consider the parameters of the framework itself. By comparing sensitivity analysis plots for SVM and AISEC, it is clear that the two techniques respond very differently to the variations of some parameters. Let us assume that an instance of the framework contains only classifiers of a certain technique. At some point, the structural adaptivity replaces all the classifiers with new ones of a different technique. This change might require the parameters of the framework to be optimized for the new technique.

5.4.3 Structural adaptivity

Similar to the way stovepipes are replaced at the classification levels, creation, deletion and replacement should also be considered for other components of the framework: data fusion modules and boxes. In this way, an instantiation of the framework might switch between majority and weighting voting to optimise the performance. Similarly, assuming that the same instantiation includes multiple boxes, if one of these becomes ineffective, it should be replaced with a new one.

Structural adaptivity should also be extended to the levels of decision fusion and feature extraction. In particular, we believe that feature extraction has a vast and unexplored potential. In a near future, classifiers will be entrusted to select or extract the most suitable features starting from raw data, rather than being

dictated which features to use. Some research initiatives have started investigating this possibility [46]. In such a way, the features that are used across the lifetime of a classification system become likely to vary. Starting from a set of features that provides suboptimal performance, the framework should be able to find new features that increase the separation between the classes.

5.4.4 Different forms of feedback

Despite its simplicity, the mechanism of feedback proposed in this thesis can handle concept drift without supervision, although some constraints are necessary. This raises the question of what benefits could derive from the extension of the concept of feedback to a larger scale. Rather than being limited to a single level in depth, feedback might in fact extended to multiple lower levels. This is valid for every level. Moreover, the output of a box might be fed to the lower level of another box. This horizontal expansion would increase the internal connectivity of the framework. It would be interesting to investigate the outcomes of such an architecture.

5.4.5 Multiple techniques

The possibility of using multiple boxes implicitly assumes the co-existence of different techniques in different boxes. However, different techniques might reside in the same box thus working in symbiosis at a closer level than they would do if they were included in separate boxes.

Preliminary experiments in this direction have already been performed. However, the mechanism of drift inference is not designed to consider diverse techniques. In fact, when the distribution is stationary, it assumes that the average distance between naïve stovepipes, between mature stovepipes and between mature and naïve stovepipes have similar values. If the mature pool contains predominantly stovepipes of a technique, and the naïve pool contains techniques of a different type, the assumption may not hold any more. Since the average distances could be very different, when concept drift starts the average distance between mature and naïve stovepipes could not be higher than the other two distances and drift would not be inferred. This suggests that the inference of drift mechanism needs to be modified to account for different techniques.

5.5 Final comments

The adaptive framework that is proposed has very interesting characteristics. In fact, it allows for quick adaptation, detection of drift, limited memory requirements. We suggest that the ideas of the adaptive framework of modularity, comparison between different modules and feedback should be investigated to increase the level of adaptation of classifiers.

Appendices

Appendix A

Interpretation of the sensitivity analysis of the framework with SVM: additional plots

This appendix extends the interpretation of the sensitivity analysis of Section 4.2.1.4 with a set of plots. In particular, that section analysed the performance of two instances of the framework, each associated with a different setting of its parameters. In correspondence of the first setting, the instance of the framework maintains high accuracy and recall. Only precision has slightly smaller values with respect to the other two measures. For the second setting, the values of precision tend to 1, while accuracy and recall have low values. Section 4.2.1.4 presented the conclusion that an instance of the framework with the first set of parameters is able classify the data correctly (as suggested by its values of accuracy), while the instance with the second set of parameters is not able to cope with the drift of the data. Moreover, a rationale of the operation of the framework with SVM was developed based on that analysis.

All the plots are generated from the 50 runs that were performed for each sample. Each plot shows the distributions of the values of the measure being considered at different percentages of completion of a run. The plots associated with the first setting are presented in Section A.1, while the plots of the second setting are explained in Section A.2

A.1 Plots of the instance of the framework with the first parametric setting

The plots of TP, TN, FP and FN, in Figure 60 and Figure 61 regard the first sample. The number of TP and that of TN (Figure 60) do not vary considerably across the iterations. However, a slight decrease of the number of TP and a slight increase of the number of TN are observed. In fact, the median value of TP at the beginning of the classification (0% of completion of a run) is higher than the median values from 10% to 100% of completion of a run. On the contrary, the median value of the TN at the beginning of a run is higher than the subsequent values. The variations of FP and FN are larger, as shown in Figure 61. In particular, the number of FP decreases while the number of FN increases with respect to their initial values, which cause, respectively, the increase of precision and the decrease of recall that are observed in Figure 62. The same figure shows that the values of accuracy do not vary much across a run.

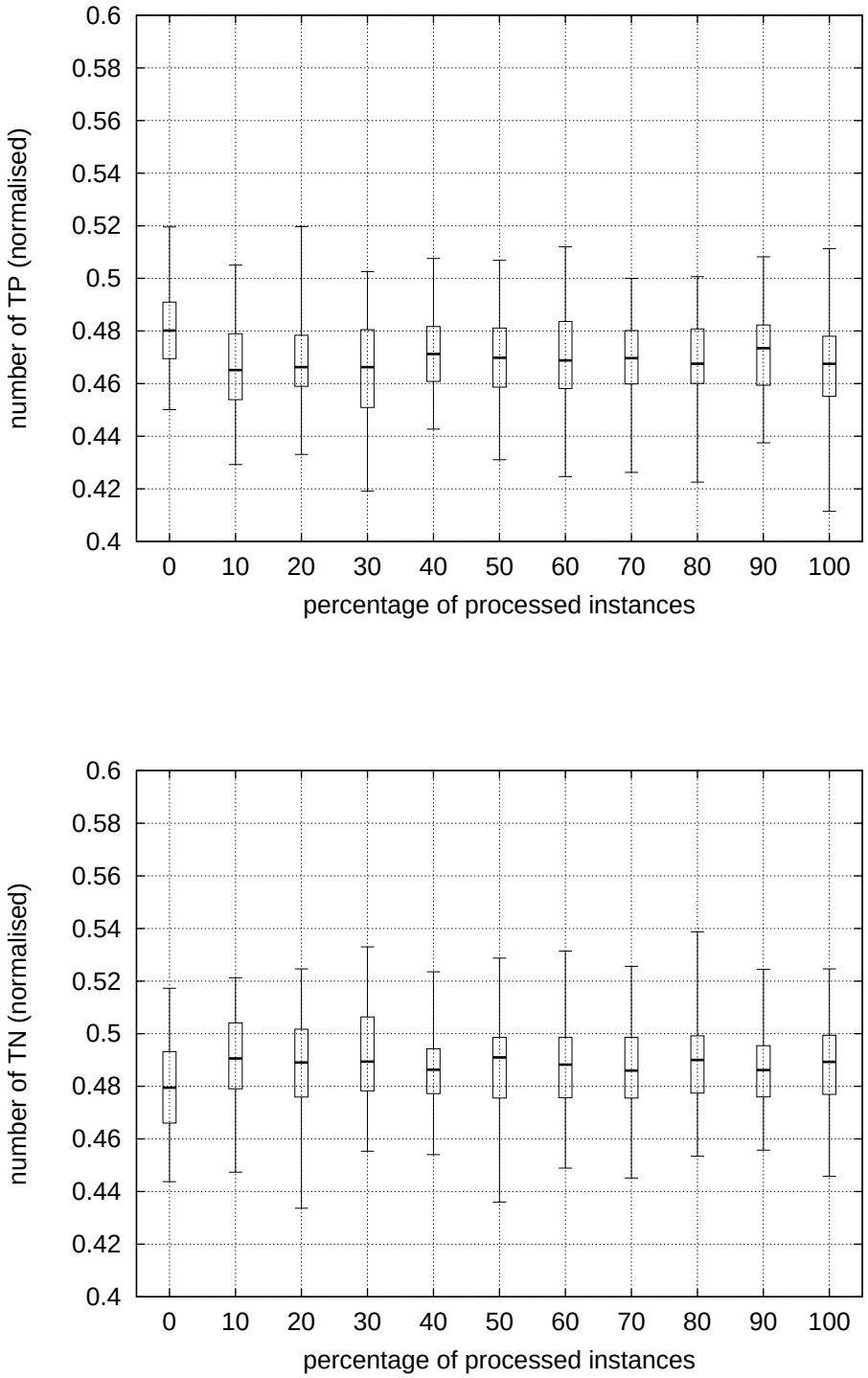


Figure 60: TP and TN in function of the percentage of completion of a run.

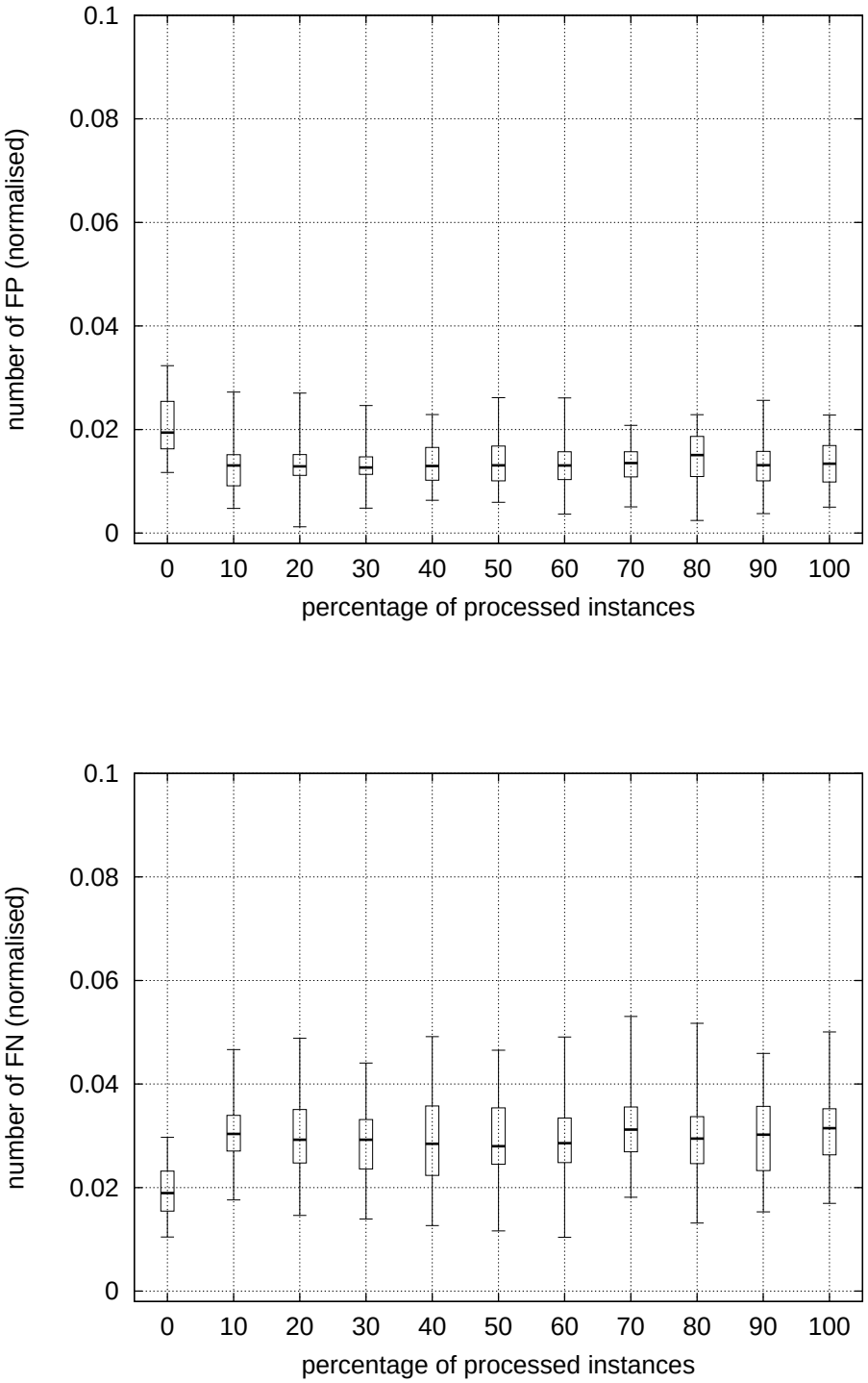


Figure 61: FP and FN in function of the percentage of completion of a run of the first parametric setting. After the start of a run, the number of FP decreases and that of FN increases with respect to their initial values (that correspond to 0% of completion of a run).

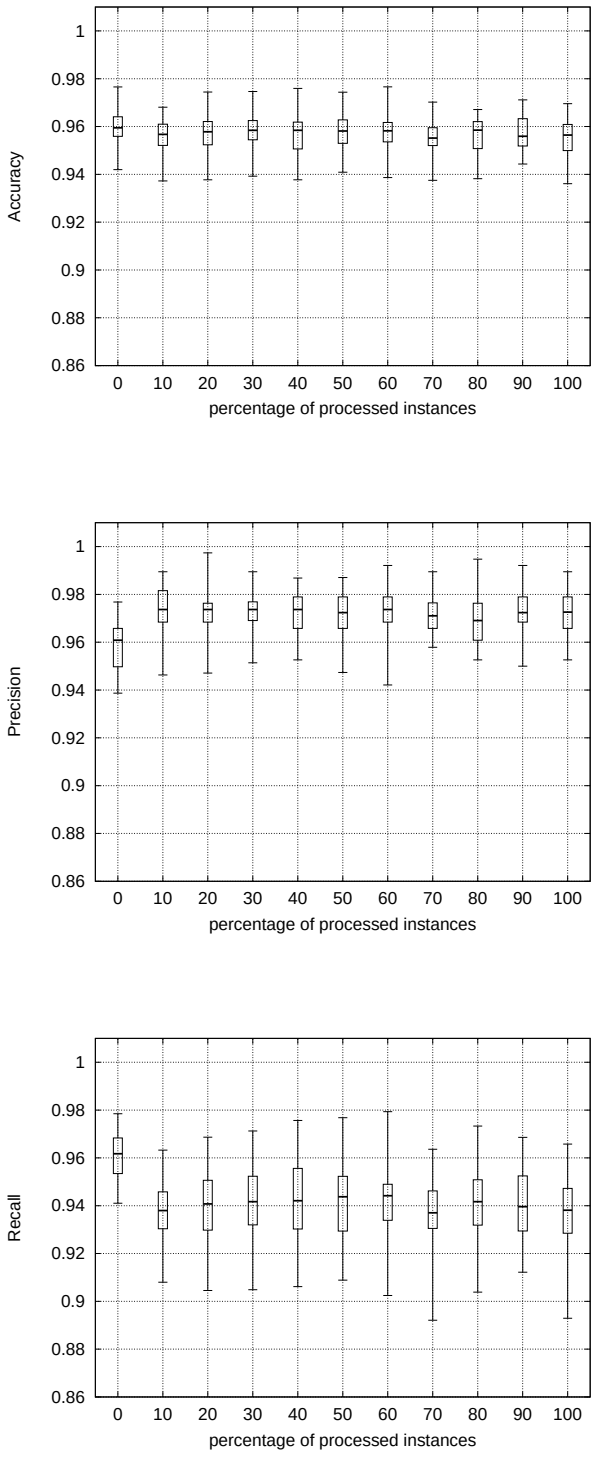
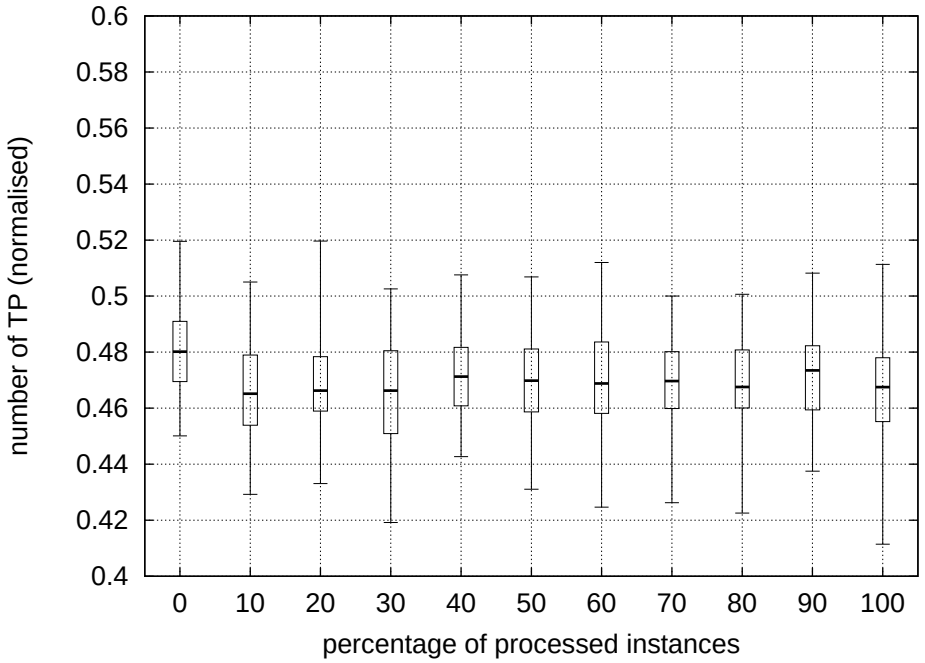


Figure 62: Accuracy, precision and recall in function of the percentage of completion of a run. While the the accuracy does not seem to vary consistently across a run, precision increase and recall decreases.

A.2 Plots of an instance of the framework associated with the second parametric setting

We now analyse the plots of TP, TN, FP and FN of the second parametric setup. In particular, accuracy and recall are smaller with respect to the first setup, but precision is higher.

Figure 63 and 64 show, respectively, the normalised numbers of TP, TN and the normalised numbers of FP, FN across different fractions of a run. In particular, the number of TP decreases towards 0, while the rate of FN tends to 0.5. These results are different from those displayed in Figure 60 and 61, in which both measures underwent only small variations. In a similar way, the number of FP tends to 0, while the fraction of FN tend to 0.5. The increment of the the sum of FP and FN for the second setting, as shown in Figure 65, causes a decrease of the values of accuracy (Figure 66). Moreover, because of the variations of TP, FP and FN, precision tends to 1, and recall tends to 0, as shown in Figure 66.



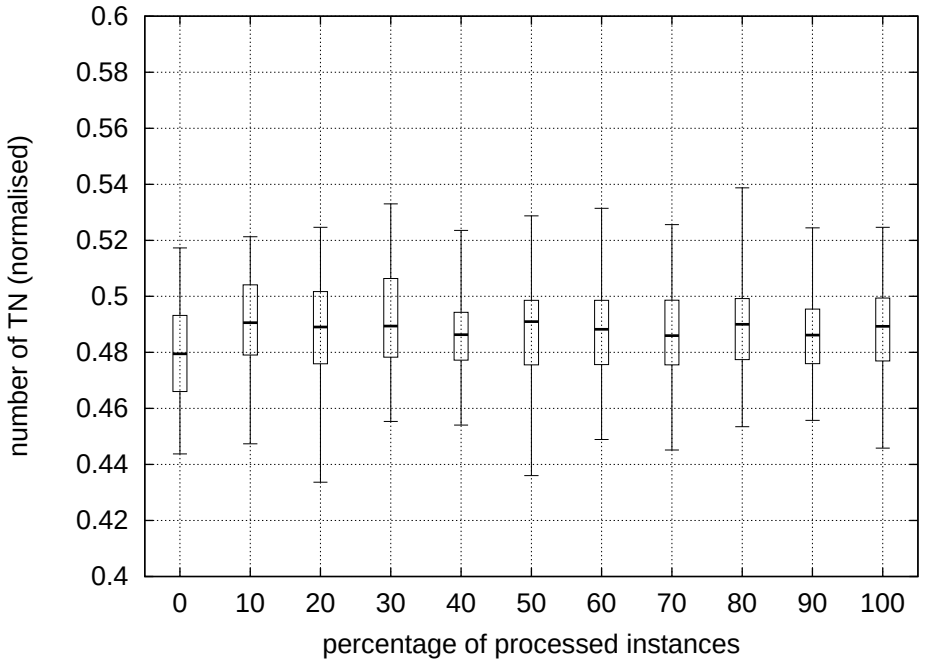
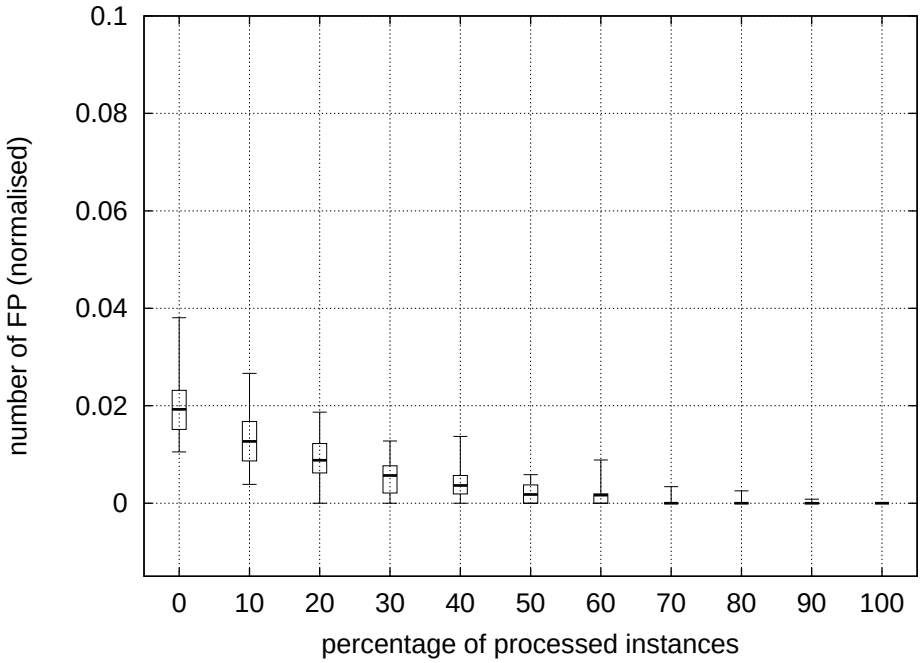


Figure 63: TP and TN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5.



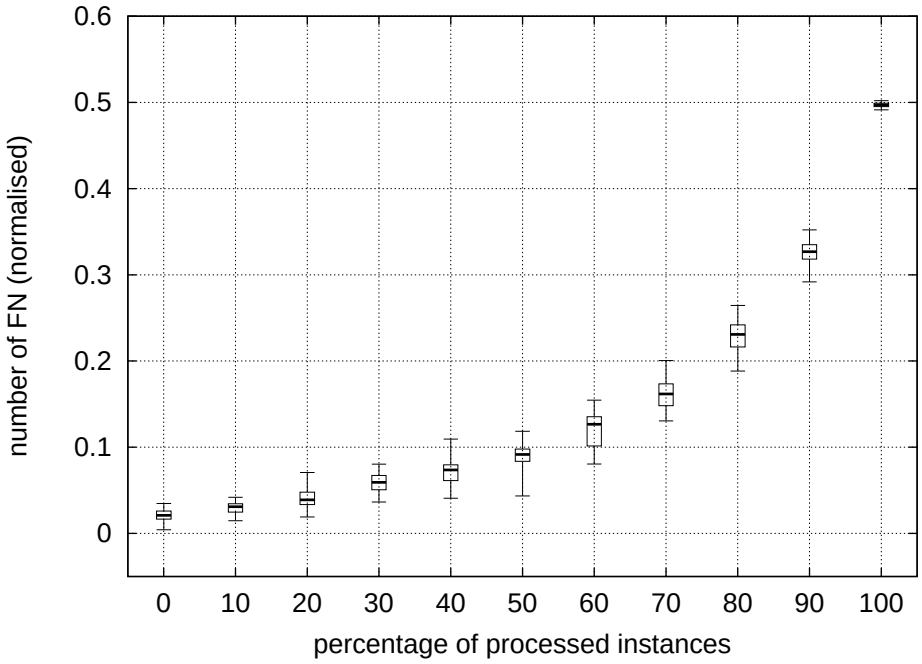


Figure 64: FP and FN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5.

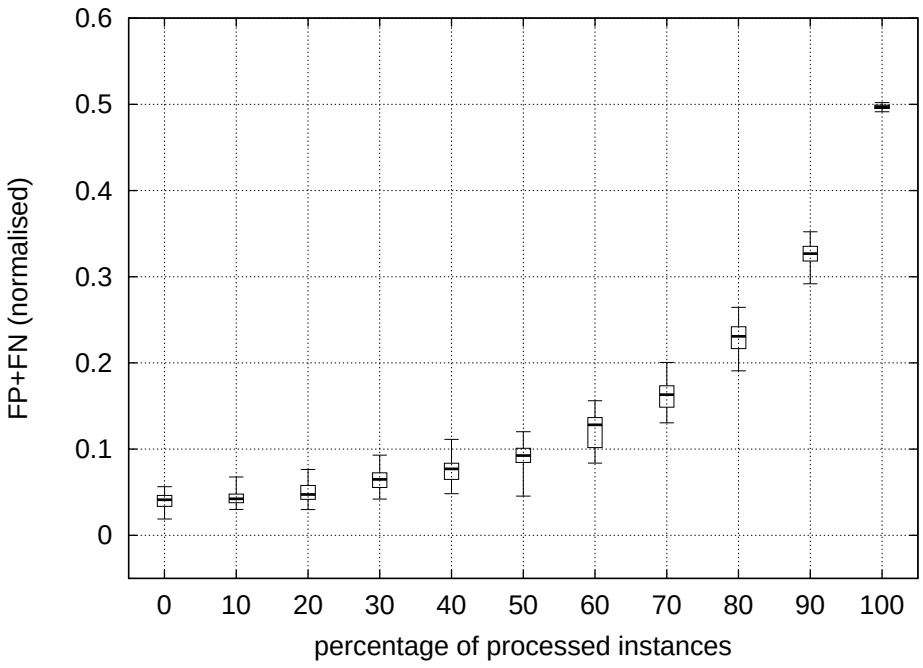


Figure 65: Error rate against percentage of completion of a run for sample 2. Its values is complementary to the value of accuracy.

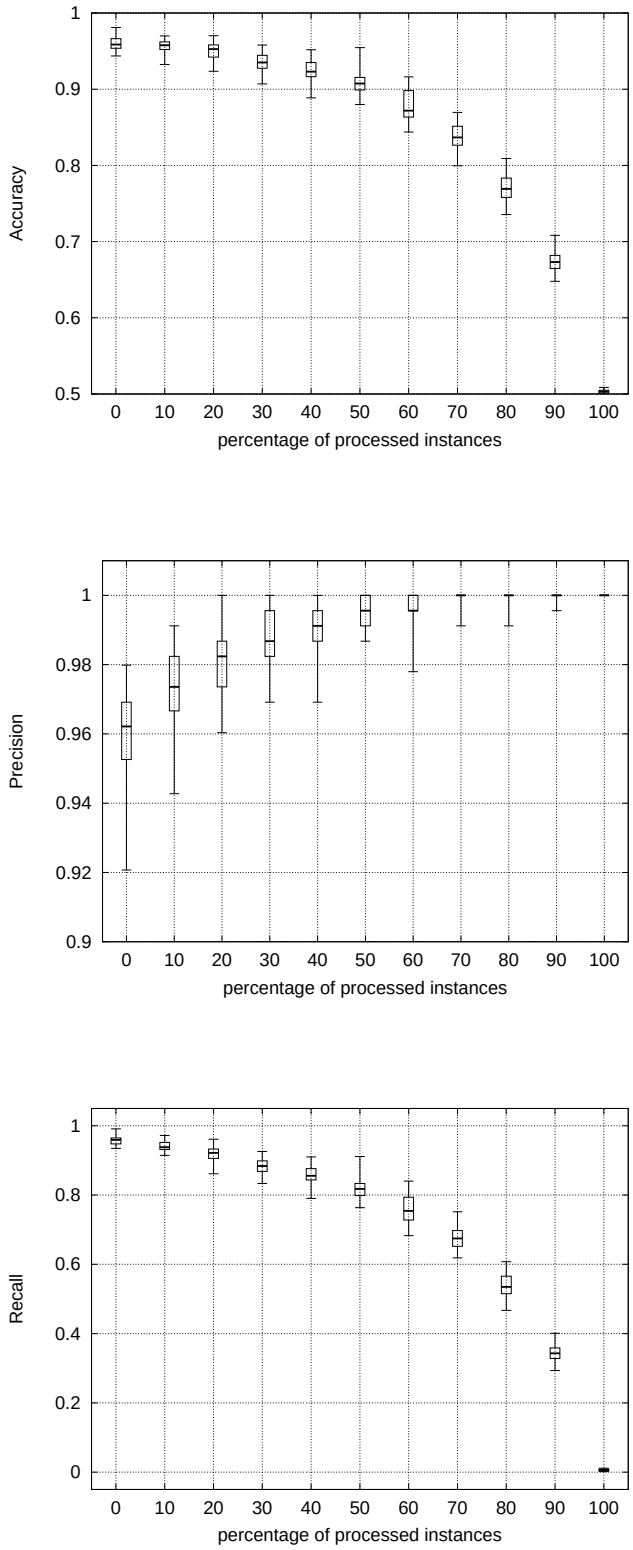


Figure 66: Accuracy, precision and recall in function of the percentage of completion of a run.

A.3 Conclusions

The interpretation of the results given in Section **??** clarifies higher precision are not associated with is the capability of an instance of the framework of dealing with concept drift.

Appendix B
Interpretation of the sensitivity analysis of the framework with AISEC: additional plots

This appendix presents additional plots of the interpretation of the sensitivity analysis of Section 4.3.1.2. Two sets of instances of the framework are analysed. The first set contains the samples (associated with parametric settings of the framework) that form the top clusters of the accuracy plots and the recall plots of the sensitivity analysis of AISEC of Figures 30-32. The second set of instances of the framework contains the samples of the bottom clusters of the same plots. In particular, the first set of samples represent parametric settings that allow the framework to deal with the drift of the Gaussian data, while the second set of samples contains parameter for which the framework cannot deal with the drift of the data. The plots of the first set of samples are descrbed in Section B.1, while those of the second set are presented in Section B.2.

B.1 Plots of the instances of the framework of the first cluster of samples

The figures 67, 68, 68 and 69 are generated from a set of samples with high accuracy and precision according to the figures (Figures 30-32). The measures of TP, TN, FP and FN are normalised. The first figure shows that the number of TP is smaller than that of TN, while for SVM the two measures have similar values. Moreover, the number of TP undergoes a small decrease, while that of TN does not seem to change. The plots of FP and FN are shown in Figure 68. The number of FP tends to 0 from a small initial value. This causes an increase of precision as shown in

Figure 69, for which the values tend to 1. The number of FN is higher than that of FP, and it increases after that drift starts. As a consequence, recall undergoes a decrement as depicted in Figure 69.

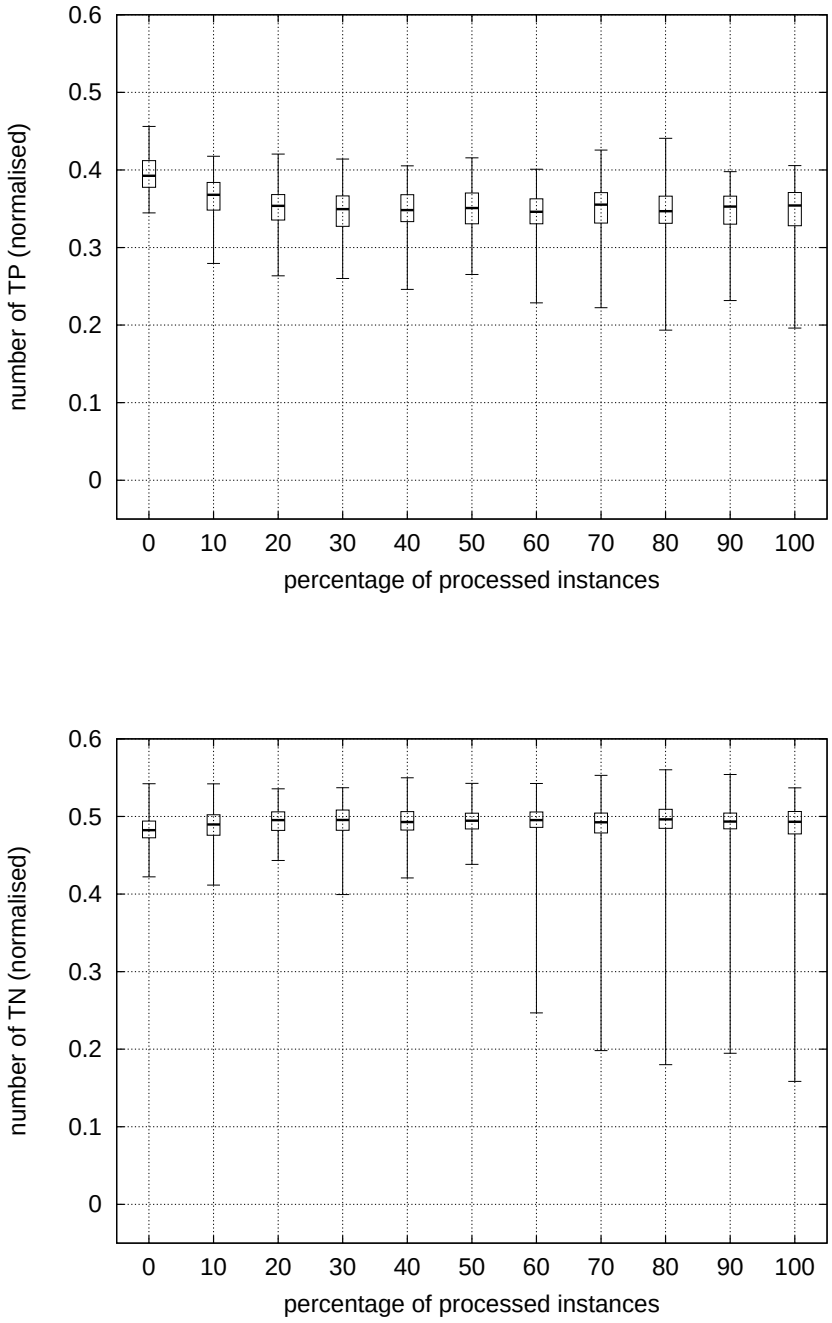


Figure 67: TP and TN in function of the percentage of completion of a run. A small decrement of TP and a small increment of TN are observed.

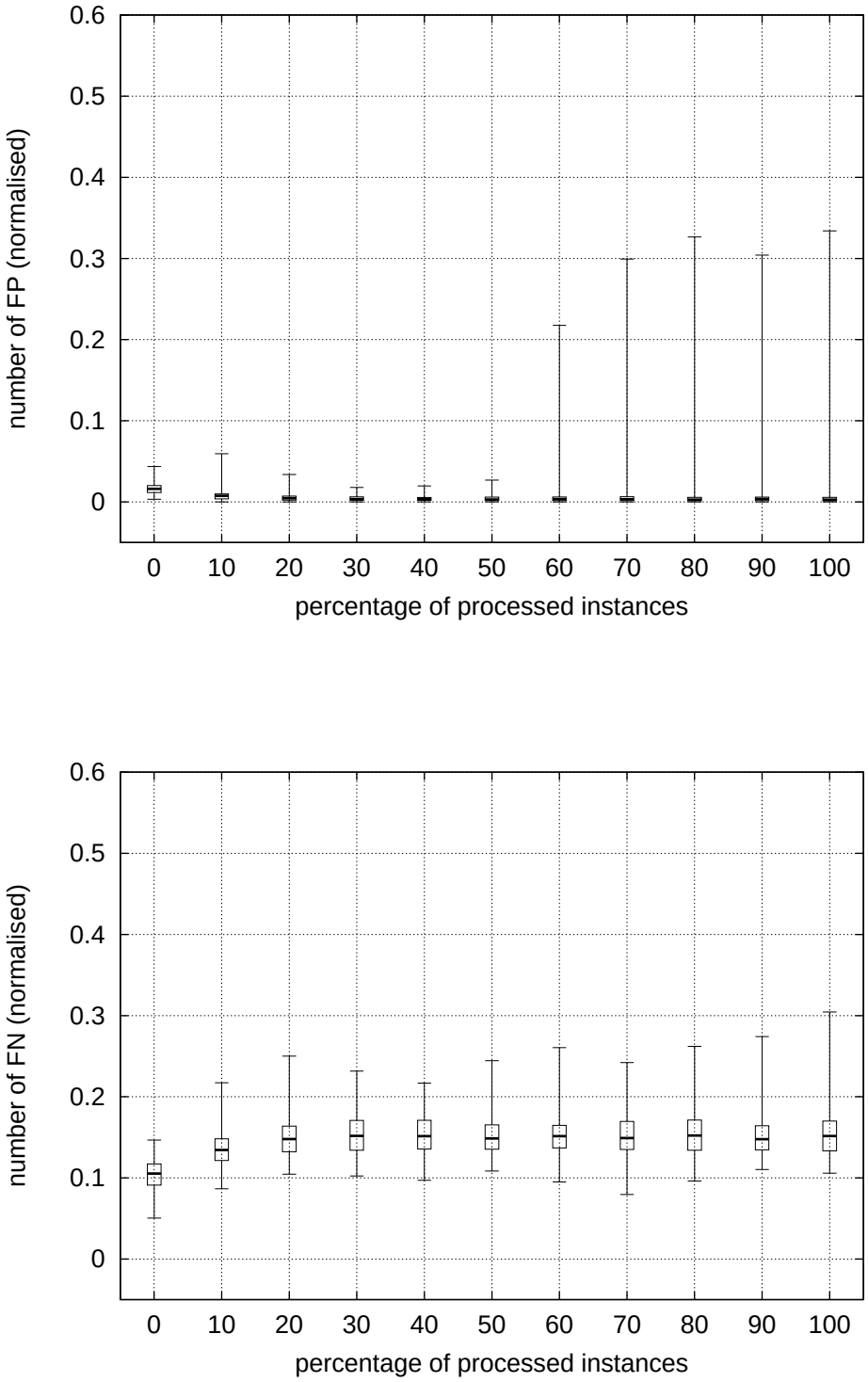


Figure 68: FP and FN in function of the percentage of completion of a run.

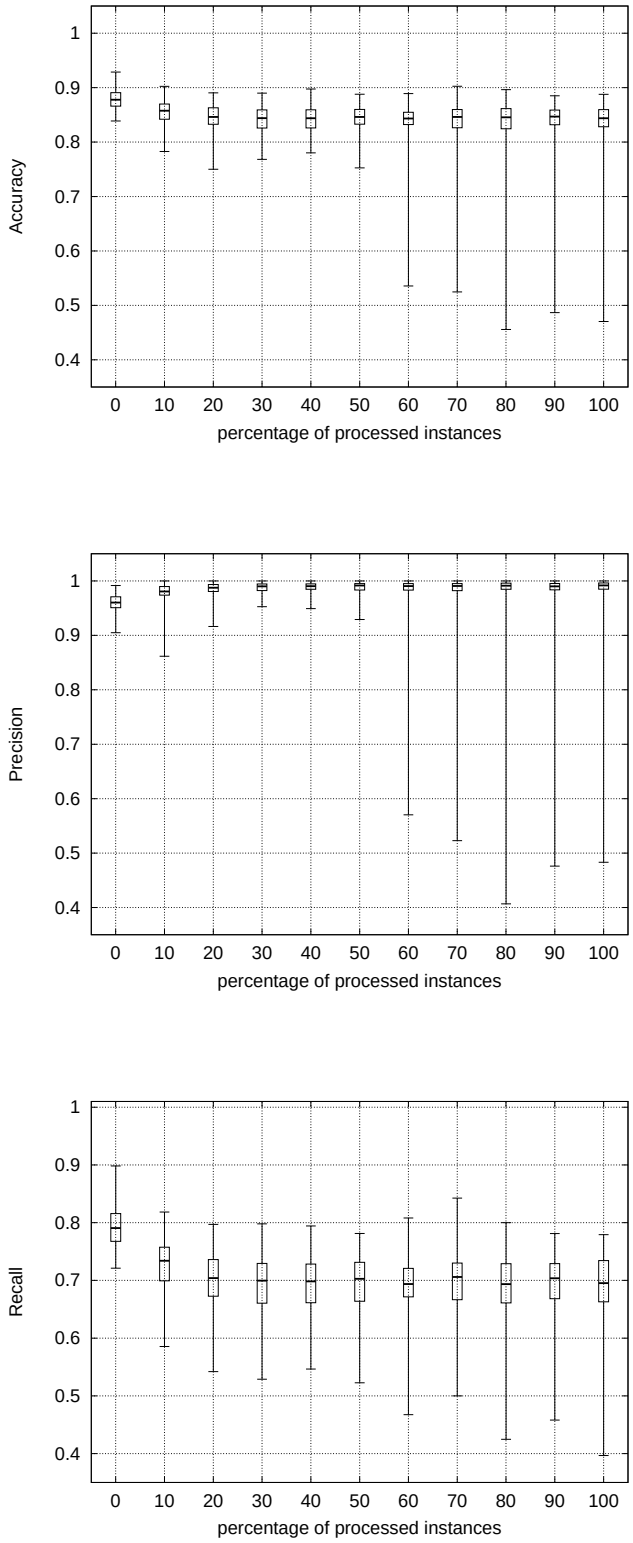


Figure 69: Accuracy, precision and recall in function of the percentage of completion of a run. Accuracy slightly decreases as soon as drift starts, precision increases and recall decreases.

B.2 Plots of the instances of the framework of the second cluster of samples

The Figures 70, 71, 72 and 73 are generated from the cluster of samples associated with low values of accuracy and precision. This suggests that the samples are not able to deal with the concept drift of the Gaussian dataset. The plots of accuracy and precision of the previous section show two distinct clusters. The number of TP is similar to that displayed in Figure 68, which depicts samples that are able to deal with concept drift. Instead, the number of TN drops as soon as drift starts, while for the other cluster of samples it increases. The number of FP increases considerably, so that precision drops as shown in Figure 66. By contrast, the number of FN does not seem to be affected, as the Figures 71 and 68 display similar values of FN. This is the reason why, recall does not vary much between Figure 73 and 69 and therefore the recall plots of the previous do not present distinct clusters as those of precision and accuracy do. However, recall undergoes a slight decrement after drift starts.

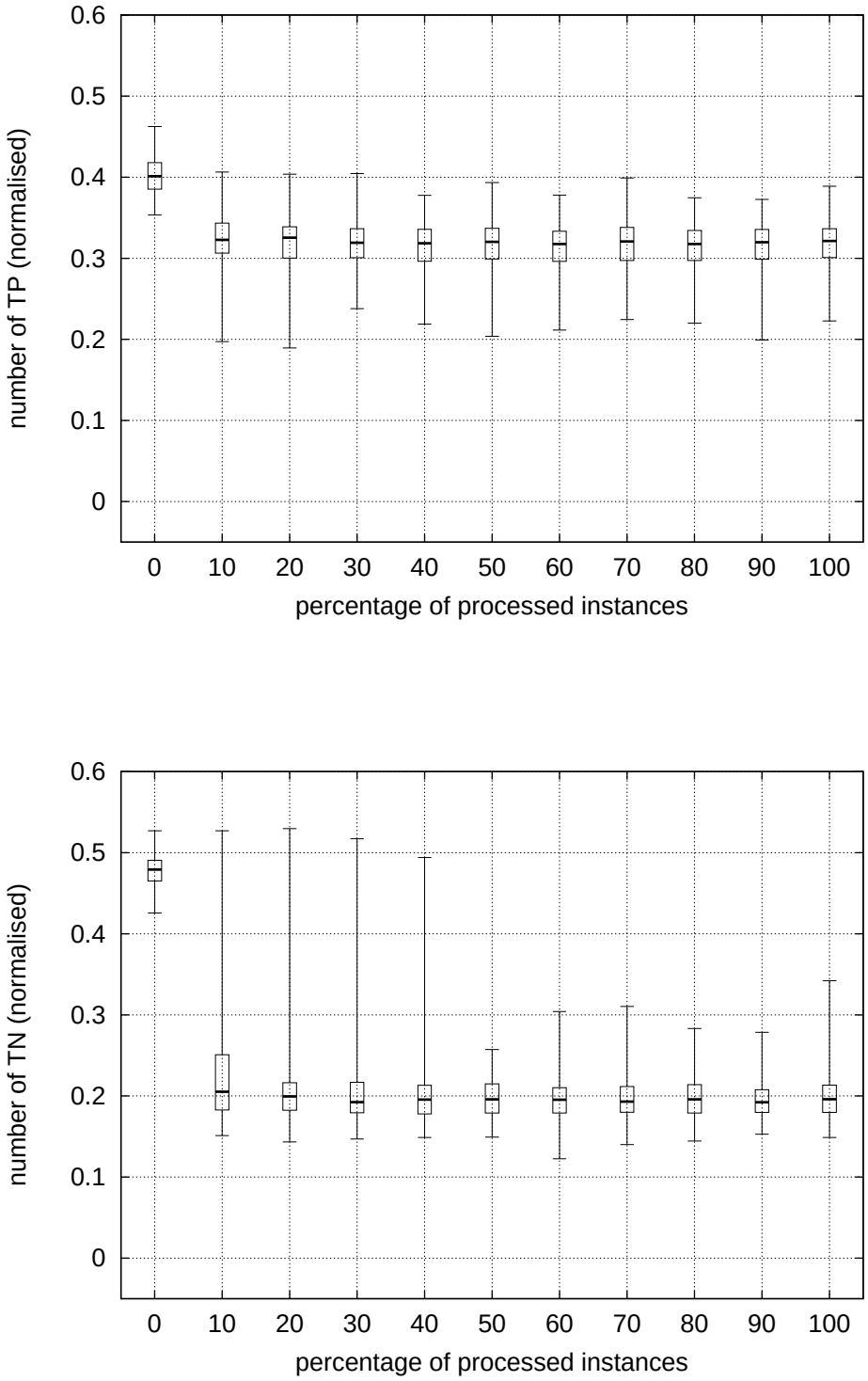


Figure 70: TP and TN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5.

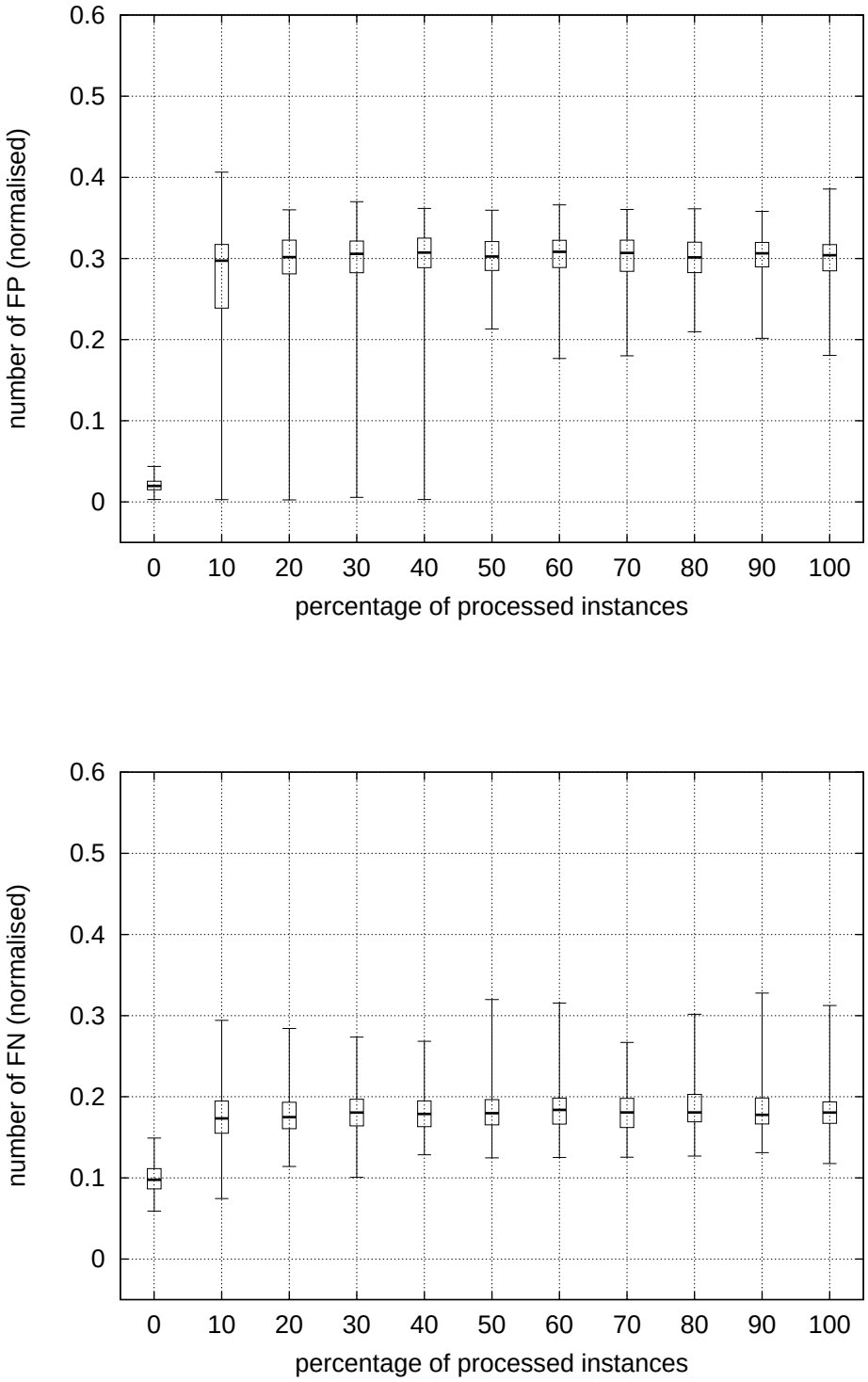


Figure 71: FP and FN in function of the percentage of completion of a run. Their normalised values tend respectively to 0 and to 0.5.

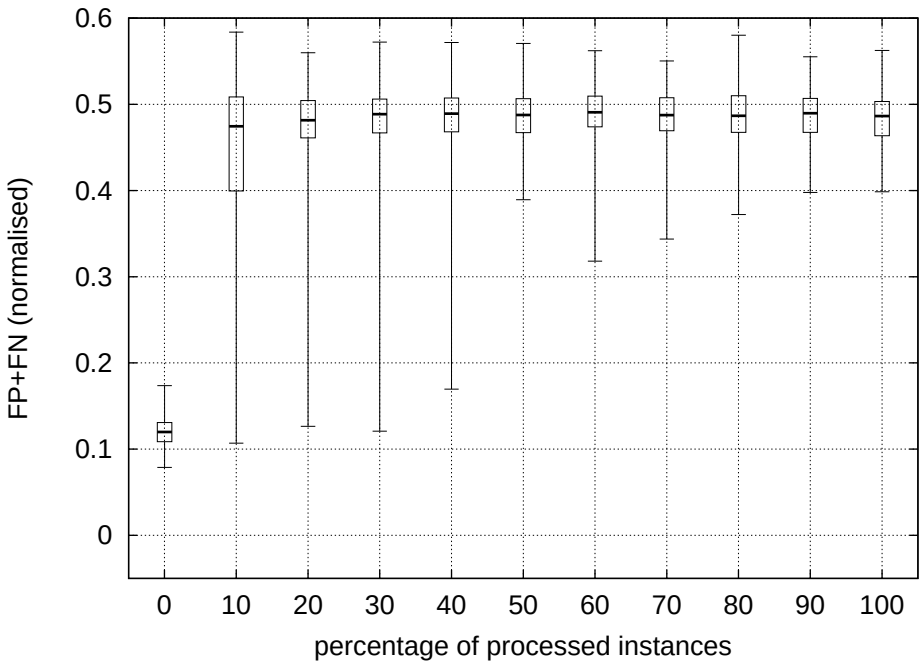


Figure 72: Error rate against percentage of completion of a run for sample 2. Its values is complementary to the value of accuracy.

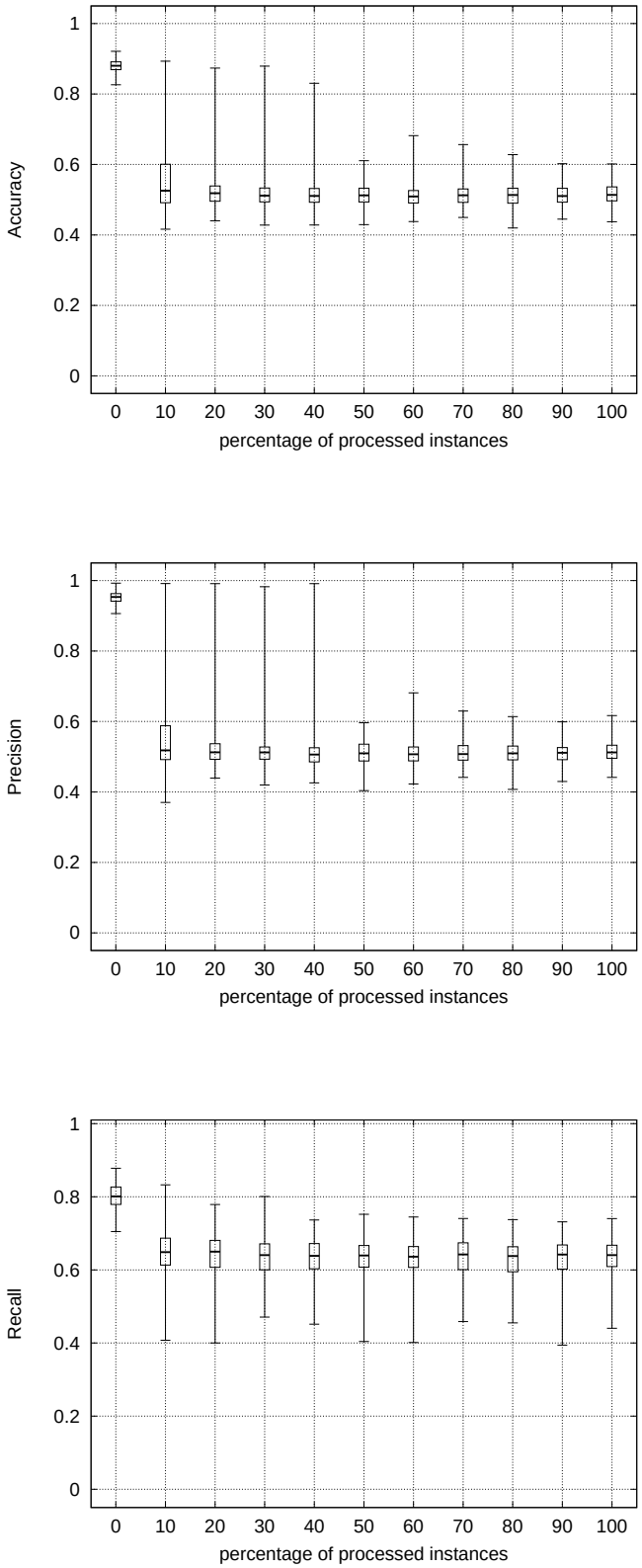


Figure 73: Accuracy, precision and recall in function of the percentage of completion of a run.

Appendix C
Sensitivity analyses: additional plots

This Appendix presents additional plots of the analyses of the sensitivity of the framework with SVM (C.1) on the Gaussian distribution, with AISEC on the same data (C.2) and with SVM on the uniform distribution (C.3).

C.1 Sensitivity Analysis plots of the framework with SVM and Gaussian data

This section presents some of the plots of the sensitivity analysis that was presented in Section 4.2.1. Figure 74 displays the plots related to the parameter $th_{training}$. The density of the top clusters of the accuracy plot (and the recall plot), which is higher on the right side of the plot, suggests that higher values of $th_{training}$ (that produce ensembles with few classifiers) generate better models. The density of the top cluster of the accuracy plot in Figure 75 entails that higher values of th_{ID} generate lower performance. In fact, high values of that threshold affect the activation of the mechanism of drift inference. The Figures 76 (parameter $ratio_{naiveMature}$) and 77 (parameter $FIFOsize_{ID}$) display homogeneous clusters. This indicates that these parameters do not affect much the performance of the framework.

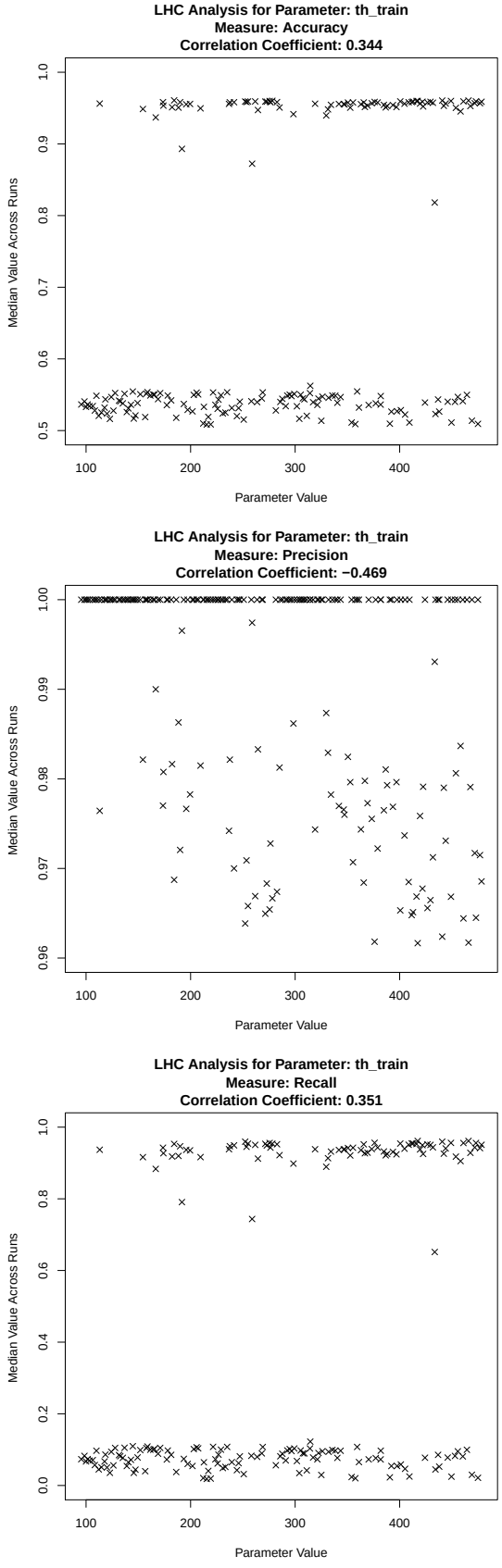


Figure 74: Latin hypercube sensitivity analysis plot of $th_{training}$ against accuracy, precision and recall.

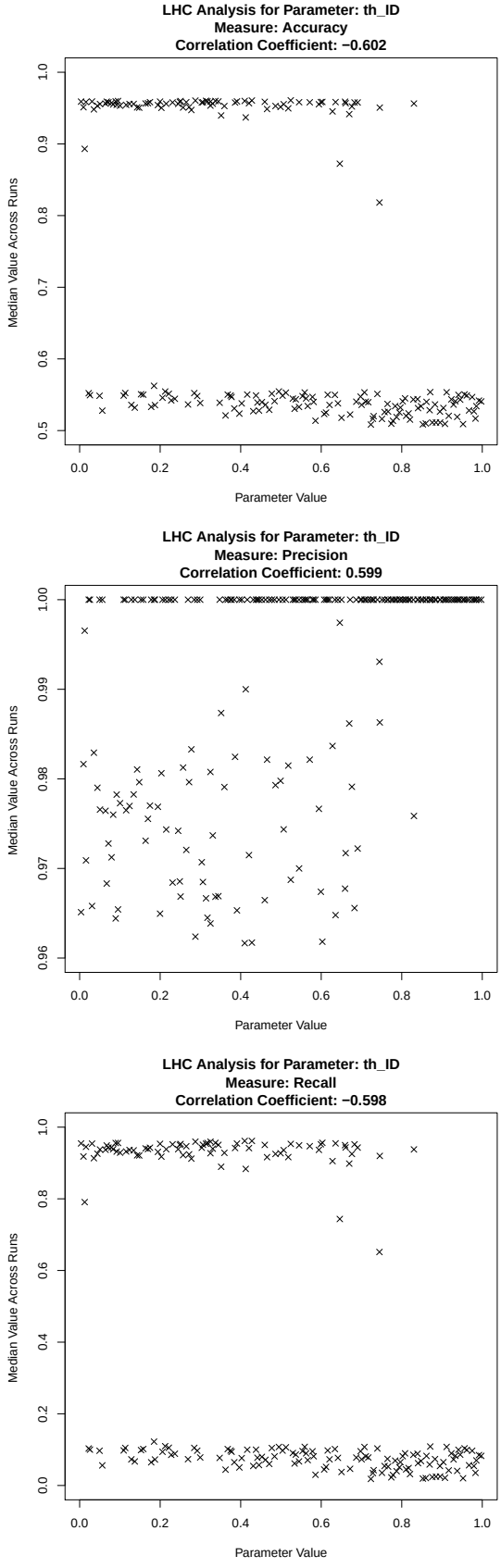


Figure 75: Latin hypercube sensitivity analysis plots of th_{ID} against accuracy, precision and recall.

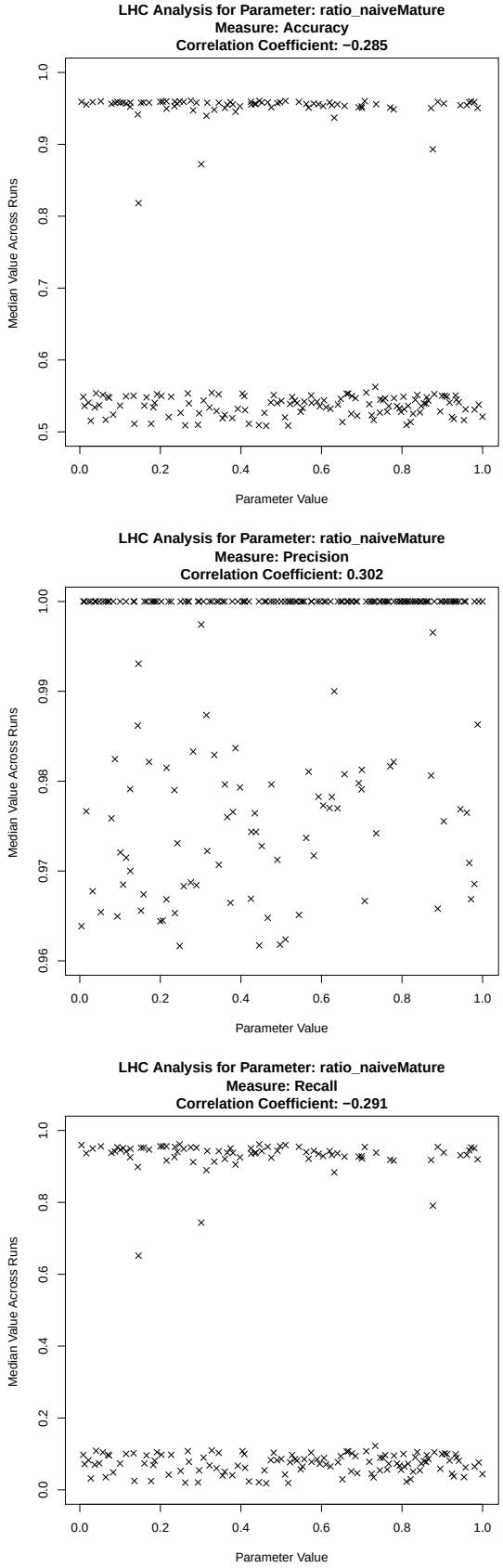


Figure 76: Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall.

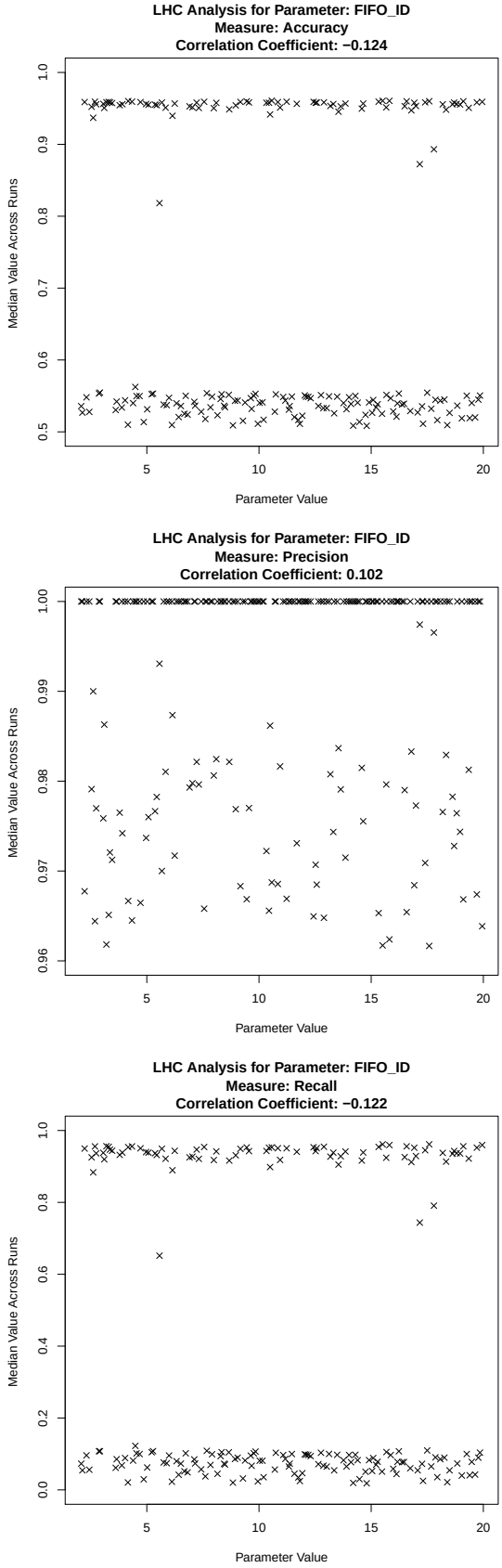


Figure 77: Latin hypercube sensitivity analysis plot of $FIFOsize_{ID}$ against accuracy, precision and recall.

C.2 Sensitivity Analysis plots of the framework with AISEC and Gaussian data

Sensitivity analysis plots of the experiment in Section 4.3.1. In that experiment, the framework uses AISEC as a classification technique, while the data has Gaussian distribution. Figure 78 and Figure 79 show the response of the output measures to variations of, respectively, th_{online} and $FIFOsize_{ID}$. The fact that the top and the bottom clusters of the plots are homogeneous indicates that these parameters do not have a significant influence on the performance of the framework.

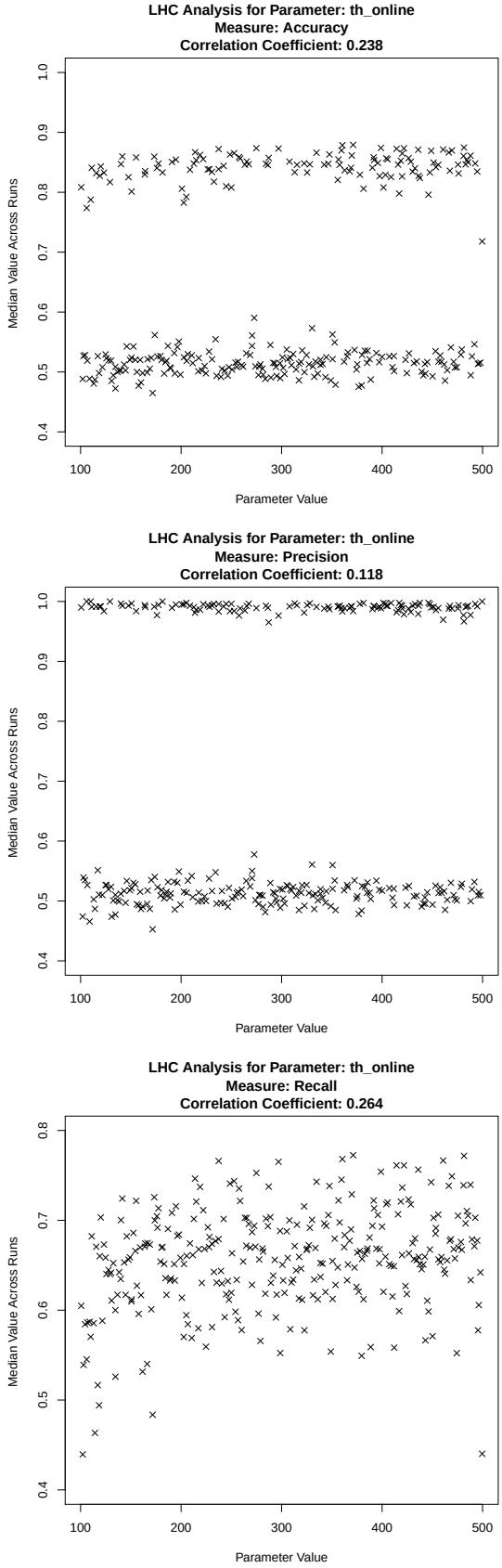


Figure 78: Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall.

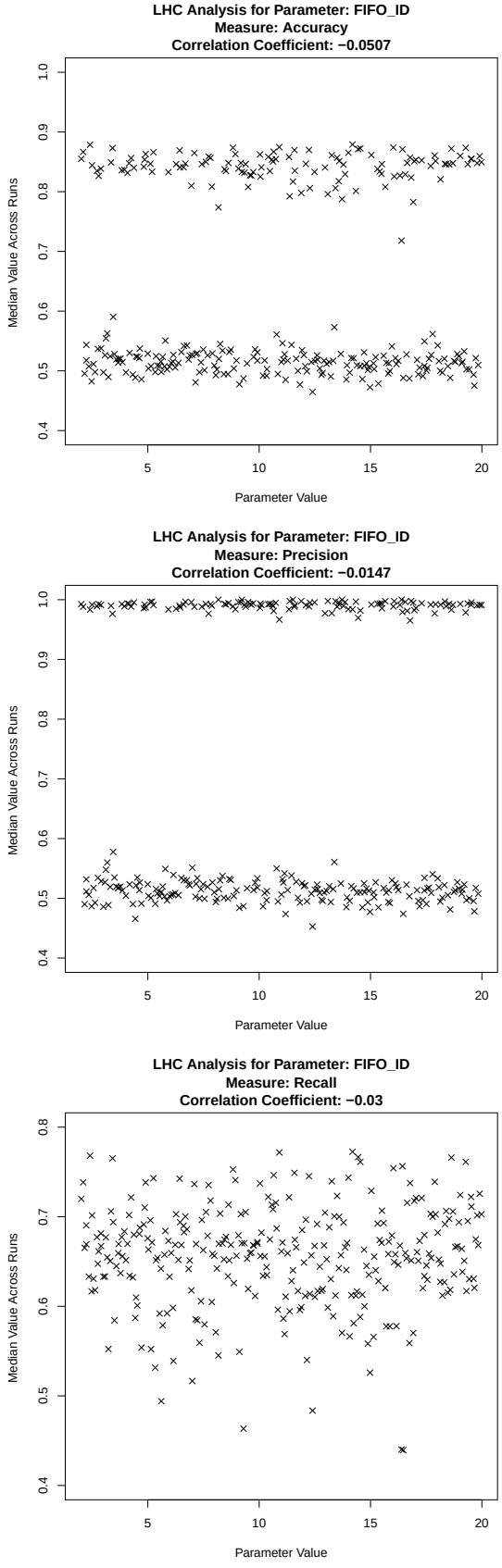


Figure 79: Latin hypercube sensitivity analysis plot of $FIFOsize_{ID}$ against accuracy, precision and recall.

C.3 Sensitivity Analysis plots of the framework with SVM and uniform data

Sensitivity analysis plots of the experiment with uniformly distributed data (Section 4.2.2.1). The experiment is presented in Section 4.2.2. The plots displayed in the Figures 26-83 show accuracy, precision and recall in function of the parameters of the framework. In particular, the majority of the values of precision and recall are equal to zero. The fact that some values of precision are high, is due ot the fact that the median values were considered for the analyssi of the sensitivity. The values at the end of the runs are low, as shown in Figure 17. Moreover, the values of accuracy are higher than the values of the other two measures. However, since the imbalance between the number of positive instances and the number of negative instances is high for the uniform dataset, such values of accuracy do not imply that the framework is able to deal with the concept drift of the data.

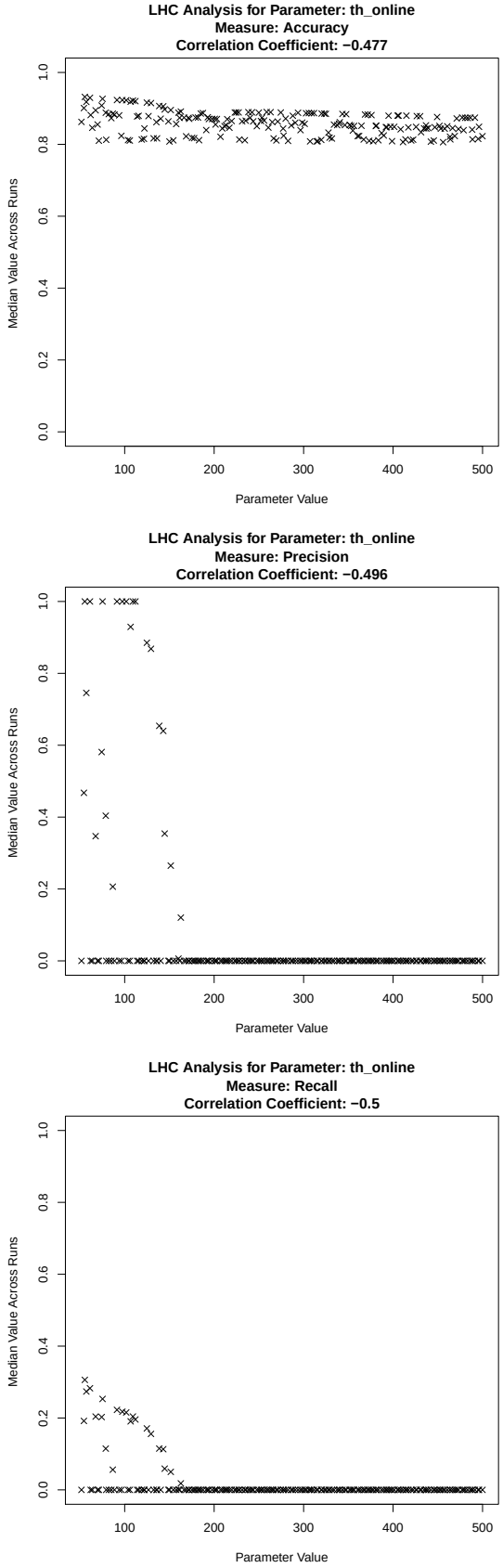


Figure 80: Latin hypercube sensitivity analysis plot of th_{online} against accuracy, precision and recall.

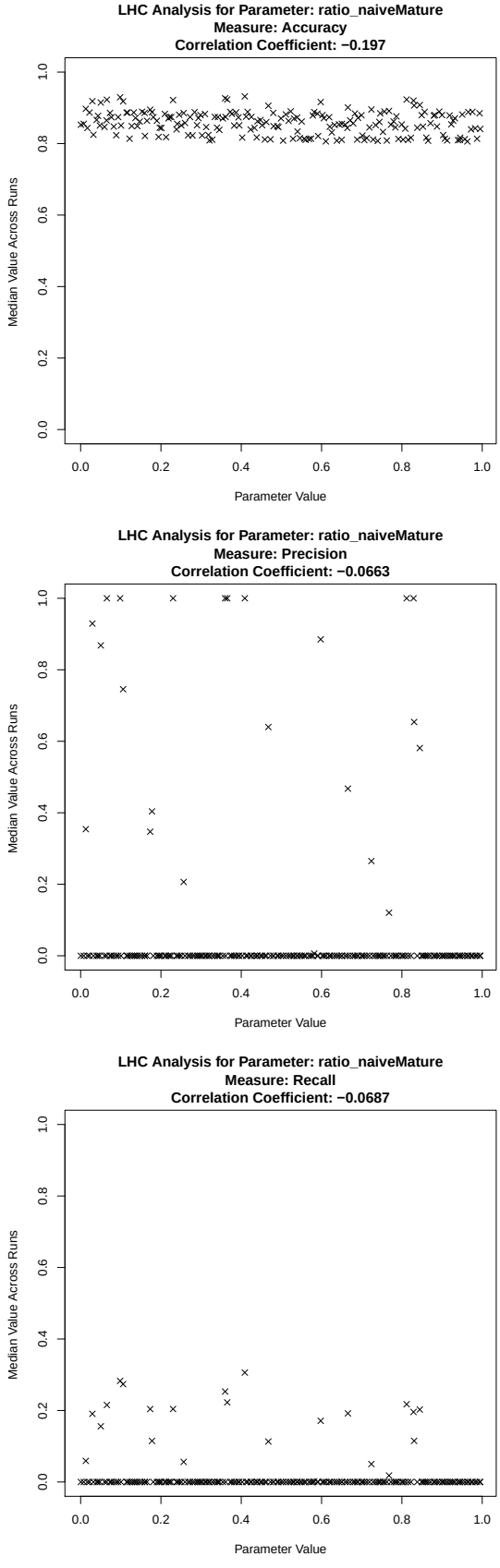


Figure 81: Latin hypercube sensitivity analysis plot of $ratio_{naiveMature}$ against accuracy, precision and recall.

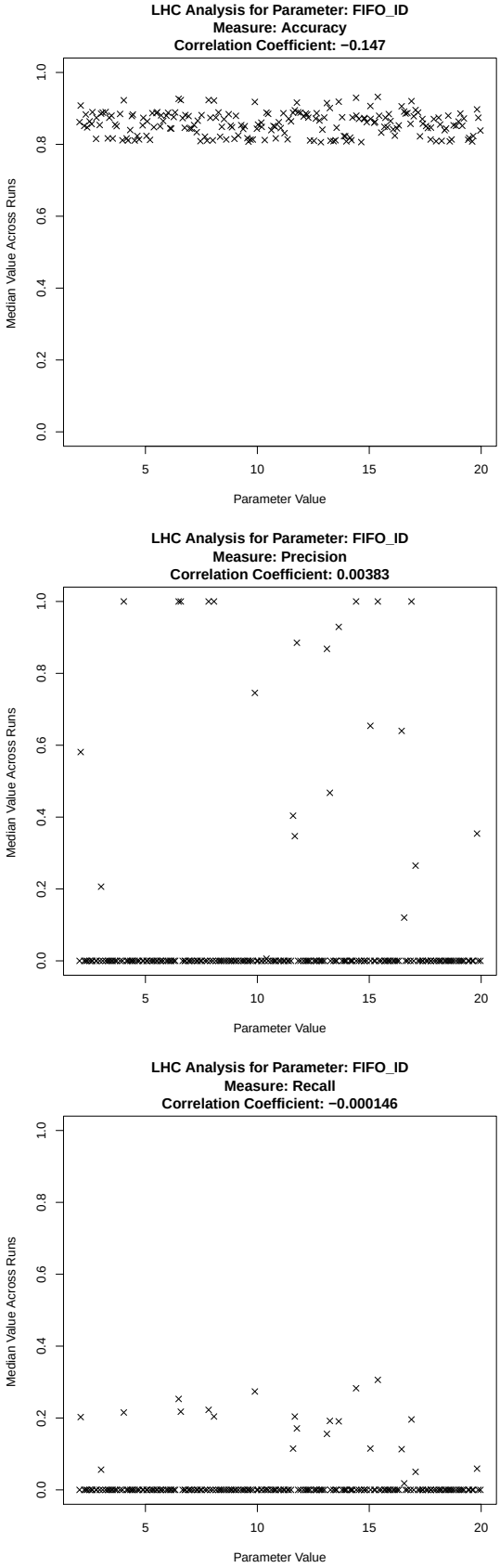


Figure 82: Latin hypercube sensitivity analysis plot of $FIFOsize_{ID}$ against accuracy, precision and recall.

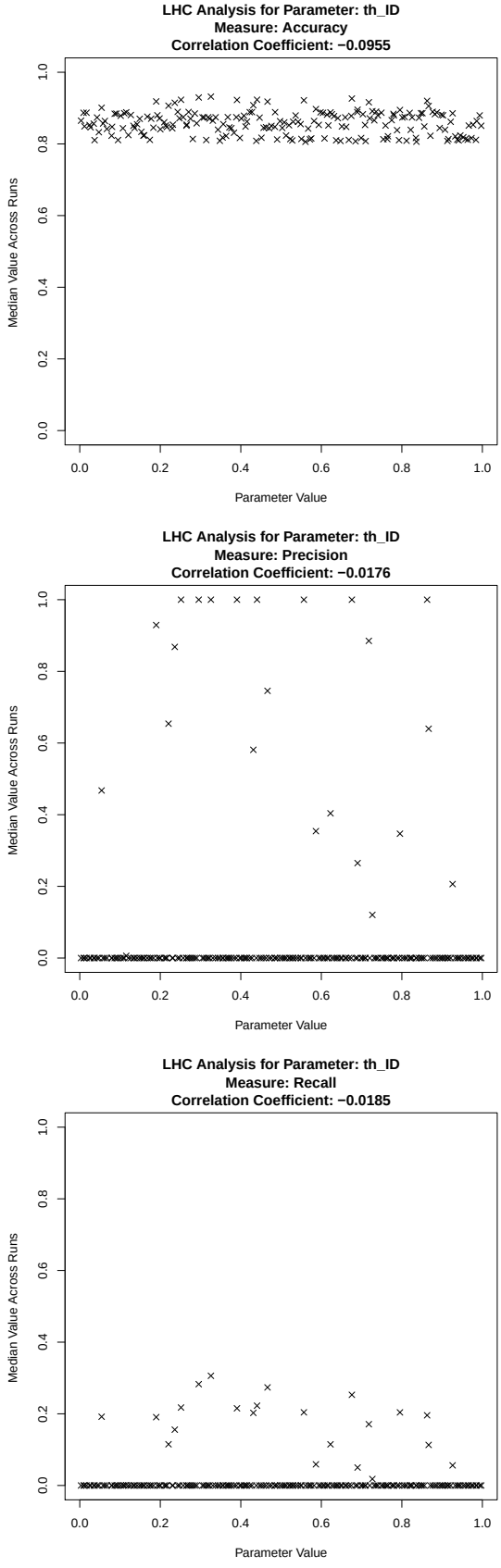


Figure 83: Latin hypercube sensitivity analysis plot of th_{ID} against accuracy, precision and recall.

Appendix D
Data plots

Figure 84 shows the data points (red is used for the positive points, blue for the negative ones) and the boundaries of the mature stovepipes (black lines) of a specific run, as soon as drift starts (a), after several iterations of an instance of the framework that can deal with drift (b), and after several iterations of an instance of the framework that cannot deal with drift (c). The orange dotted line represents the boundary with maximum distance from the centers of the Gaussian clusters. At the start of a run (a), the boundaries of the mature stovepipes are "close" to the orange line, after several iterations (b), the boundaries are on the left of the orange line. If an instance of the framework is not able to deal with the drift of the data, the boundaries are much more shifted towards the left. This was anticipated in Figure 16.

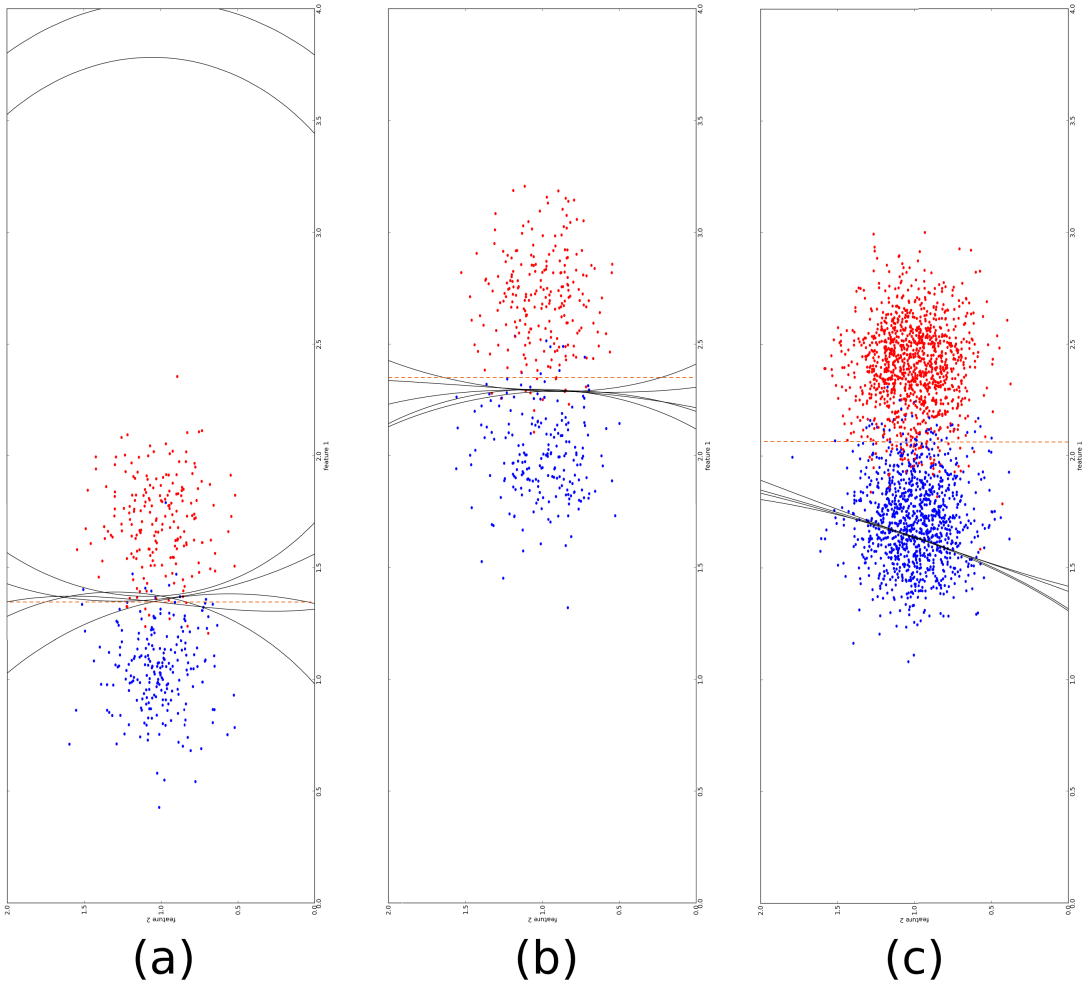


Figure 84: Example of data and boundaries. Red points indicate positive instances, while blue points are used for negative instances. The black lines represent the boundaries of the mature classifiers, while the orange dotted line maximises the distance between the center of the Gaussian distributions. At the start of the run (a) the black lines and the orange line have a similar position. After concept drift has started, the black lines are on the right of the orange one.

Acronyms

SVM Support Vector Machines

AISEC Artificial Immune System for Email Classification

ANN Artificial Neural Network

OWA Optimal Weight Adjustment

O-LDC Online Linear Discriminant Function

VFDT Very Fast Decision Tree

CVFDT Concept-adapting Very Fast Decision Tree

TP True Positive

TP True Positive

TN True Negative

FP False Positive

FN False Negative

DWM Dynamic Weighted Majority

ADWIN ADaptive WINdowing

MOA Massive Online Analysis

EA Evolutionary algorithm

PSO Particle Swarm Optimization

pdf probability density function

Bibliography

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. On demand classification of data streams. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 503–508, New York, USA, 2004. ACM.

[2] Charu C. Aggarwal, T. J. Watson, Resch Ctr, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the Twenty-nineth International Conference on Very Large Data Bases*, volume 29 of *VLDB '03*, pages 81–92, Berlin, Germany, 2003. VLDB Endowment.

[3] Jesper Andersson, Rogério Lemos, Sam Malek, and Danny Weyns. Software engineering for self-adaptive systems. In Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Lecture Notes in Computer Science*, volume 5525, chapter Modeling Dimensions of Self-Adaptive Software Systems, pages 27–47. Springer-Verlag, Berlin, Heidelberg, 2009.

[4] Stephen H. Bach and Marcus A. Maloof. Paired learners for concept drift. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 23–32, Washington, DC, USA, 2008. IEEE Computer Society.

[5] Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Semi-supervised clustering by seeding. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 27–34, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[6] P. Berkhin. A survey of clustering data mining techniques. In Jacob Kogan, Charles Nicholas, and Marc Teboulle, editors, *Grouping Multidimensional Data*, pages 25–71. Springer Berlin Heidelberg, 2006.

[7] Albert Bifet, Joao Gama, Mykola Pechenizkiy, and Indre Zliobaite. Handling concept drift: Importance, challenges & solutions. Tutorial at Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2011.

[8] Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In *Proceedings of the Eighth International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis*, volume 8 of *IDA '09*, pages 249–260, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the Fifteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 139–148, New York, NY, USA, 2009. ACM.

[10] Mikhail Bilenko and Raymond J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 39–48, New York, NY, USA, 2003. ACM.

[11] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[12] Avrim Blum and Shuchi Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 19–26, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[13] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[14] Jurgen Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[15] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

[16] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, June 1998.

[17] F. M. Burnet. *The clonal selection theory of acquired immunity*. Vanderbilt University Press, 1959.

[18] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, SDM '06, pages 328–339. SIAM, 2006.

[19] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, Cambridge, USA, 1st edition, 2010.

[20] Fang Chu and Carlo Zaniolo. Fast and light boosting for adaptive mining of data streams. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 282–292. Springer Berlin Heidelberg, 2004.

[21] David Cohn, Rich Caruana, and Andrew Mccallum. Semi-supervised clustering with user feedback. Technical report, The University of Texas at Austin, 2003.

[22] Corinna Cortes and Vladimir N. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[24] Gregory Ditzler and Robi Polikar. An ensemble based incremental learning framework for concept drift and class imbalance. In *Proceedings of the 2010 International Joint Conference on Neural Networks*, IJCNN '10, pages 1 –8, 2010.

[25] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80. ACM, 2000.

[26] Anton Dries and Ulrich Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining*, 2(5-6):311–327, December 2009.

[27] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2001.

[28] Martin Ester, Hans peter Kriegel, Jrg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD '96, pages 226–231. AAAI Press, 1996.

[29] George Forman. Tackling concept drift by temporal inductive transfer. In *Proceedings of the Twenty-ninth annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 252–259, New York, USA, 2006. ACM.

[30] Jerome H. Friedman and Lawrence C. Rafsky. Multivariate generalizations of the Wald-Wolfowitz and smirnov two-sample tests. *Annals of Statistics*, 7:697–717, 1979.

[31] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In AnaL.C. Bazzan and Sofiane Labidi, editors, *Advances in Artificial Intelligence SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer Berlin Heidelberg, 2004.

[32] Zoubin Ghahramani. Unsupervised learning. In Olivier Bousquet, Ulrike Luxburg, and Gunnar Rtsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 72–112. Springer Berlin Heidelberg, 2004.

[33] Hinco J Gierman and Rogier Versteeg. *Clustering of Highly Expressed Genes in the Human Genome*. John Wiley & Sons, Ltd, 2001.

[34] Nizar Grira, Michel Crucianu, and Nozha Boujemaa. Unsupervised and semi-supervised clustering: a brief survey. *A review of machine learning techniques for processing multimedia content, Report of the MUSCLE European Network of Excellence (FP6)*, 2004.

[35] Stephen Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1(1):17–61, 1988.

[36] John A. Hartigan. *Clustering algorithms*. Wiley series in probability and mathematical statistics: Applied probability and statistics. John Wiley & Sons, Inc., New York, NY, USA, 1975.

[37] Shohei Hido, Tsuyoshi Ide, Hisashi Kashima, Harunobu Kubo, and Hirofumi Matsuzawa. Unsupervised change analysis using supervised learning. In Takashi Washio, Einoshin Suzuki, KaiMing Ting, and Akihiro Inokuchi, editors, *Advances in Knowledge Discovery and Data Mining*, volume 5012 of *Lecture Notes in Computer Science*, pages 148–159. Springer Berlin Heidelberg, 2008.

[38] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[39] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and data mining*, KDD '01, pages 97–106. ACM, 2001.

[40] Anil K. Jain and Dubes Richard C. *Algorithms for clustering data*. Prentice Hall, Inc., 1988.

[41] Petr Kadlec and Bogdan Gabrys. Architecture for development of adaptive on-line prediction models. *Memetic Computing*, 1(4):241–269, 2009.

[42] N. Kasabov. Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 31(6):902–918, 2001.

[43] Nikola Kasabov. Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 31(6):902–918, 2001.

[44] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Dynamic feature space and incremental feature selection for the classification of textual

data streams. In *ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams. 2006*, page 107. Springer Verlag, 2006.

[45] Alden Kieran, Read Mark, Andrews Paul, Timmis Jon, Veiga-Fernandes Henrique, and Coles Mark. spartan: A comprehensive tool for understanding uncertainty in computer simulation results. *PLoS Computational Biology*, 2:1–9, 2012.

[46] YongSeog Kim, W. Nick Street, and Filippo Menczer. Optimal ensemble construction via meta-evolutionary ensembles. *Expert Systems with Applications*, 30(4):705 – 714, 2006.

[47] Ralf Klinkenberg. Using labeled and unlabeled data to learn drifting concepts. In *Workshop notes of the IJCAI-01 Workshop on Learning from Temporal and Spatial Data*, pages 16–24. AAAI Press, 2001.

[48] Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.

[49] Ralf Klinkenberg and Thorsten Joachims. Detecting concept drift with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 487–494, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[50] Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning drifting concepts. In *Proceedings of AAAI-98/ICML-98 workshop Learning for Text Categorization*, pages 33–40, 1998.

[51] Adam Knowles. *Immunologically Inspired Data Fusion for Anomaly Detection in Electromechanical Devices*. PhD thesis, University of York, 2009.

[52] Ron Kohavi and Ross Quinlan. Decision-tree discovery. In *Handbook Of Data Mininig And Knowledge Discovery*, chapter 16, pages 267–276. Oxford University Press, 2002.

[53] Jeremy Z. Kolter and Marcus A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 123–, Washington, DC, USA, 2003. IEEE Computer Society.

[54] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[55] Ivan Koychev. Gradual forgetting for adaptation to concept drift. In *Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning*, pages 101–106, 2000.

[56] Ludmila I. Kuncheva and Friedrich Steimann. Fuzzy diagnosis. *Artificial Intelligence in Medicine*, 16(2):121 – 128, 1999.

[57] Peipei Li, Xindong Wu, and Xuegang Hu. Mining recurring concept drifts with limited labeled streaming data. *ACM Transsactions in Intelligent Systems Technology*, 3(2):29:1–29:32, February 2012.

[58] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 1947.

[59] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Proccedings of the Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 929–934, 2008.

[60] Leandro L. Minku, Allan P. White, and Xin Yao. The Impact of Diversity on Online Ensemble Learning in the Presence of Concept Drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, May 2010.

[61] T. Mitchell. *Machine Learning (Mcgraw-Hill International Edit)*. McGraw-Hill Education (ISE Editions), 1997.

[62] Tom Mitchell. Generative and discriminative classifiers: Naive bayes and logistic regression, 2005.

[63] Tom Mitchell. The discipline of machine learning. Technical Report CMU ML-06 108, Carnegie Mellon University, 2006.

[64] Michael D. Muhlbaier and Robi Polikar. An ensemble approach for incremental learning in nonstationary environments. In *Proceedings of the Seventh*

International Conference on Multiple Classifier Systems, MCS'07, pages 490–500, Berlin, Heidelberg, 2007. Springer-Verlag.

[65] Fionn Murtagh and Pedro Contreras. Methods of hierarchical clustering. *Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[66] Anand Narasimhamurthy and Ludmila I. Kuncheva. A framework for generating data to simulate changing environments. In *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications*, pages 384–389, Anaheim, CA, USA, 2007. ACTA Press.

[67] Detlef Nauck and Rudolf Kruse. Obtaining interpretable fuzzy classification rules from medical data. *Artificial Intelligence in Medicine*, 16(2):149 – 169, 1999.

[68] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.

[69] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *NIPS: Advances in Neural Information Processing Systems*, pages 841–848. MIT Press, 2001.

[70] Nikunj Chandrakant Oza. Online ensemble learning. Technical report, Massachussetts Institute of Technology, 2001.

[71] A. Pocock, P. Yiapanis, J. Singer, M. Luján, and G. Brown. Online non-stationary boosting. *Lecture Notes in Computer Science*, 5997:205–214, 2010.

[72] R. Polikar. Ensemble Based Systems in Decision Making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

[73] Massimiliano Pontil and Ro Verri. Properties of support vector machines. *Neural Computation*, 1998.

[74] J. R. Quinlan. Bagging, boosting, and c4.5. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730. AAAI Press, 1996.

[75] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 1993.

[76] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back- propagating errors. *Nature*, 323:533–536, 1986.

[77] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[78] Zoheir Sahel, Abdelhamid Bouchachia, Bogdan Gabrys, and Paul Rogers. Adaptive mechanisms for classification problems with drifting data. In *Proceedings of the Eleventh International Conference on Knowledge-Based Intelligent Information and Engineering Systems and the XVII Italian Workshop on Neural Networks*, KES '07, pages 419–426, Berlin, Heidelberg, 2007. Springer-Verlag.

[79] Marcos Salganicoff. Density-adaptive learning and forgetting. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 276–283. Morgan Kaufmann, 1993.

[80] A. Saltelli, K. Chan, and Scott. *Sensitivity analysis*. Wiley series in probability and statistics. Wiley, 2000.

[81] A. L. Samuel. Some studies in machine learning using the game of checkers. II Recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967.

[82] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[83] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, July 1990.

[84] Jeffrey C. Schlimmer and Jr. Granger, RichardH. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[85] A. Secker, A Freitas, and J. Timmis. AISEC: An Artificial Immune System for E-mail Classification. In R. Sarker, R. Reynolds, H. Abbass, T. Kay-Chen, R. McKay, D Essam, and T. Gedeon, editors, *Proceedings of the Congress*

on Evolutionary Computation, pages 131–139, Canberra. Australia, December 2003. IEEE.

[86] Aihua Shen, Rencheng Tong, and Yaochen Deng. Application of classification models on credit card fraud detection. In *Proceedings of 2007 International Conference on Service Systems and Service Management*, pages 1–4, 2007.

[87] Lindsay I. Smith. A tutorial on Principal Components Analysis. Technical report, Cornell University, USA, 2002.

[88] W. Nick Street and Yongseog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 377–382. ACM Press, 2001.

[89] Gerald Tesauro. Practical issues in temporal difference learning. *Mach. Learn.*, 8(3-4):257–277, May 1992.

[90] Gerald Tesauro. Temporal difference learning and TD-gammon. *Commununi-cations of the ACM*, 38(3):58–68, March 1995.

[91] Alexey Tsymbal. The problem of concept drift: Definitions and related work. Technical report, Trinity College Dublin, Ireland, 2004.

[92] Vladimir N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer, 1982.

[93] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[94] Andras Vargha and Harold D. Delaney. A critique and improvement of the "CL" common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[95] A. Wald and J. Wolfowitz. On a test whether two samples are from the same population. *The Annals of Mathematical Statistics*, 11(2):147–162, 1940.

[96] SholomM. Weiss, Nitin Indurkhya, and Tong Zhang. Overview of text mining. In *Fundamentals of Predictive Text Mining*, volume 41 of *Texts in Computer Science*, pages 1–12. Springer London, 2010.

[97] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[98] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[99] S. Yang. Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. *Evolutionary Computation*, 16(3):385–416, 2008.

[100] S. Yang and C. Li. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *Evolutionary Computation, IEEE Transactions on*, 14(6):959–974, 2010.

[101] S. Yang, Y.S. Ong, and Y. Jin. *Evolutionary computation in dynamic and uncertain environments*. Springer Verlag, 2007.

[102] Ying Yang, Xindong Wu, and Xingquan Zhu. Mining in anticipation for concept change: Proactive-reactive prediction in data streams. *Data Mining and Knowledge Discovery*, 13(3):261–289, 2006.

[103] Peng Zhang, Xingquan Zhu, and Li Guo. Mining data streams with labeled and unlabeled training examples. In *Proceedings of the Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 627–636, Washington, DC, USA, 2009. IEEE Computer Society.

[104] Xiaojin Zhu. Semi-Supervised Learning Literature Survey. Technical report, Computer Sciences, University of Wisconsin-Madison, 2005.

[105] I. Zliobaite. Learning under concept drift: an overview. Technical report, Overview, Technical report, Vilnius University, 2009 techniques, related areas, applications Subjects: Artificial Intelligence, 2009.

[106] Indre Zliobaite. *Adaptive Training Set Formation*. PhD thesis, Vilnius University, Lithuania, 2010.