Mariana de Araújo Souza

A SUPERVISED ALGORITHM FOR ENSEMBLE GENERATION

B.Sc. Dissertation

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2015

Federal University of Pernambuco
Center for Informatics
Undergraduate in Computer Engineering

Mariana de Araújo Souza

A SUPERVISED ALGORITHM FOR ENSEMBLE GENERATION

A B.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Advisor: *George Darmiton da Cunha Cavalcanti*

RECIFE
2015

,

Acknowledgements

This dissertation has benefited from the support of many people, so I would like to spend a few lines to sincerely thank some of them.

Firstly, to Prof George Darmiton, for introducing me to this field of study and offering me such an interesting topic to research about. As my advisor, Prof George has conducted everything most graciously with his kind and patient manner and his always highly qualified comments and suggestions. Thank you very much!

I would also like to thank everyone who, in one way or another, contributed to this work.

Thanks to all my professors for their time, support and knowledge they have transmitted me during my entire undergraduate course. Their guidance and instruction contributed greatly to my development as a professional.

On a more personal level, I would like to thank my family and friends for being there for me and believing in me in every step of the way.

To my boyfriend, Saulo, for his understanding and patience as well as his support, especially when I had to dedicate myself to this work.

Many thanks to my sister, whose support came as it needed to be, from distracting and entertaining me in stressful times to poking me when I was feeling a bit lazy. You are the best, Mel.

Finally, but first in my heart, I am forever grateful to my parents for their continued moral and financial support throughout my life, the former being of much greater importance. The education and discipline they taught me and their example as competent professionals and responsible citizens have deeply shaped both my professional and personal life, and this has proven invaluable to me.

Resumo

Os Sistemas de Múltiplos Classificadores (MCSs) baseados em seleção dinâmica de classificadores são capazes de atingir altas taxas de acerto por estimar a competência dos classificadores no pool em rotular cada instância de teste e selecionar os mais adequados para realizar a classificação do objeto. O pool de classificadores deve, portanto, consistir de classificadores precisos e diversos, para que se obtenha uma melhor performance. Entretanto, para os métodos de geração de ensemble clássicos, a saber Bagging, Random Subspace e Boosting, o tamanho do pool é um parâmetro de entrada para o algoritmo, o que pode tornar-se um problema, já que é difícil antecipar o número de classificadores necessários para solucionar cada problema de classificação. Além disso, o processo de geração de pool de classificadores geralmente possui um alto custo computacional, devido à grande quantidade de classificadores gerados e ao treinamento dos mesmos. Para solucionar essas questões, um método de geração de ensemble para MCSs baseados em seleção dinâmica é proposto neste trabalho. O método proposto gera uma certa quantidade, de acordo com a distribuição de dados de treinamento, de Perceptrons localmente precisos. Ademais, ao invés de treinar os classificadores, uma heurística é utilizada para obter seus pesos, reduzindo assim o tempo de geração do pool de classificadores. Nesta dissertação, uma análise passo-a-passo do processo de geração do pool de classificadores pelo método proposto é apresentada. Também é analisado o efeito de certos aspectos, como mudanças na distribuição dos dados, fronteiras de decisão complexas e problemas de múltiplas classes, no desempenho do algoritmo. Além disso, uma avaliação comparativa do desempenho do método proposto é feita por meio de experimentos realizados utilizando diferentes técnicas de seleção dinâmica com uma quantidade razoável de problemas de classificação. Resultados obtidos de análises utilizando dados sintéticos mostram que, em comparação com os métodos de geração de ensemble, o método proposto é menos sensível a variações na distribuição de dados de teste. Além disso, para problemas com bordas complexas entre classes, o método proposto é capaz de obter uma taxa de acerto maior que os outros métodos para o mesmo tamanho de pool. Ademais, o desempenho do método proposto é equivalente ao dos mérodos de geração de ensemble clássicos para problemas multi-classe. Experimentos usando 20 conjuntos de dados mostram que o método proposto gera muito poucos classificadores e é tão preciso quanto os métodos clássicos embora seja uma ordem de magnitude mais rápido.

Palavras-chave: Sistema de múltiplos classificadores, Seleção dinâmica de classificadores, Método de geração de ensemble

Abstract

Multiple Classifier Systems (MCSs) based on the dynamic selection of classifiers are able to achieve high accuracy rates by estimating the competence of the classifiers in the pool in labelling each test instance and selecting the most suitable ones to perform the classification of the object. The pool of classifiers should, therefore, consist of accurate and diverse classifiers, in order to obtain a better performance. However, for the classical ensemble methods, namely Bagging, Random Subspace and Boosting, the size of the pool is an input parameter to the algorithm, which may be an issue since it is hard to anticipate the number of classifiers required to solve each problem. Also, the pool of classifiers generation process usually have high computational cost, due to the large amount of generated classifiers and their training. In order to tackle these issues, a method of ensemble generation for MCSs based on dynamic selection is proposed in this work. The proposed method generates a certain amount, in accordance with the training data distribution, of locally accurate Perceptrons. Moreover, instead of training the classifiers, a heuristic is used to obtain their weights, thus reducing the pool of classifiers generation process time. In this dissertation, a step-by-step analysis of the pool of classifiers generation process by the proposed method is presented. Furthermore, the effect of certain aspects on the proposed method's performance, such as changes on data distribution, complex decision boundaries and multi-class problems, is analysed. In addition to this, a comparative performance evaluation of the proposed method is done through experiments using different dynamic selection techniques over a reasonable amount of classification problems. Results obtained from analysis using synthetic data show that, in comparison with the classical ensemble methods, the proposed method is less sensitive to variations on test data distribution. Moreover, for problems with complex class borders, the proposed method is able to achieve a higher accuracy rate than the other ensemble methods for the same pool size. Also, the performance of the proposed method is equivalent to the classical ensemble methods for multi-class problems. Experiments over 20 datasets show that the proposed method generates very few classifiers and is as accurate as the classical ensemble methods whilst being an order of magnitude faster.

Keywords: Multiple classifier system, Dynamic selection of classifiers, Ensemble method

List of Figures

1.1 XOR Problem dataset 18
1.2 Locally accurate pool of Perceptrons for the XOR Problem 18
1.3 Division of the XOR Problem dataset feature space 19

2.1 Possible phases of a MCS. 23
2.2 Generation of an ensemble. 24

3.1 Toy problem 31
3.2 Generation of Perceptrons using toy problem 32

4.1 Toy problem training and test datasets generated with the same distribution . . 34
4.2 Generation of Perceptrons using toy problem 35
4.3 Test datasets used in the analysis of case I. 38
4.4 Test datasets used in the analysis of case II. 39
4.5 Test datasets used in the analysis of case III. 40
4.6 Disposal of generated pool over case scenario I datasets 41
4.7 Disposal of generated pool over case scenario II datasets 42
4.8 Disposal of generated pool over case scenario III datasets 43
4.9 P2 Problem. I represents class 1, whereas II represents class 2. 43
4.10 P2 Problem dataset. 44
4.11 Example of a pool generated by the proposed method. The arrows show the areas that the Perceptrons label as Class 1. 44
4.12 Multi-class Problem dataset. 46
4.13 Example of a pool generated by the proposed method. 46
4.14 Win-tie-loss graph of the proposed method in terms of mean accuracy achieved in comparison with the classical ensemble methods with a pool size of (a) N_{PM} and (b) N_{100}. 54

List of Tables

4.1 Mean and standard deviation of accuracy for the pool of classifiers generation methods using Dynamic Classifier Selection (DCS) techniques for the three case scenarios. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold. 36

4.2 Mean and standard deviation of pool of classifiers generation time, in seconds, for the P2 Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. 45

4.3 Mean and standard deviation of accuracy for the pool of classifiers generation methods for the P2 Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold. 45

4.4 Mean and standard deviation of pool of classifiers generation time, in seconds, for the Multi-class Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold. 45

4.5 Mean and standard deviation of accuracy for the pool of classifiers generation methods for the Multi-class Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold. 47

4.6 Main characteristics of the datasets used in the experiment 48

4.7 Mean and standard deviation of the rate at which the right Perceptron is chosen by the DCS technique, using the training dataset for testing, for the datasets from Table 4.6. Best results are in bold. 49

4.8 Mean and standard deviation of pool of classifiers generation time, in seconds, for the datasets from Table 4.6. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold. 49

4.9 Mean and standard deviation of accuracy for the pool of classifiers generation methods using DCS techniques for the datasets from Table 4.6. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold. 51

List of Acronyms

MCS Multiple Classifier System 17

DS Dynamic Selection 17

SS Static Selection 17

DES Dynamic Ensemble Selection 24

DCS Dynamic Classifier Selection 24

OLA Overall Local Accuracy 33

LCA Local Class Accuracy 33

MCB Multiple Classifier Behaviour 33

Contents

1 Introduction **17**
1.1 Motivation 17
1.2 Problem Statement 19
1.3 Objective 20
1.4 Methodology 21
1.5 Organization of the Dissertation 21

2 Background **23**
2.1 Introduction 23
2.2 Ensemble Methods 24
2.2.1 Bagging 25
2.2.2 Random Subspace 26
2.2.3 Boosting 26

3 The Proposed Method **29**

4 Experimental study **33**
4.1 Performance analysis 33
4.1.1 Variations on data disposition 33
4.1.2 P2 Problem 40
4.1.3 Multi-class Problem 44
4.2 Comparative study 47

5 Conclusion **55**

References **57**

1 Introduction

The "No Free Lunch Theorem" [31] leads to the conclusion that there is not a single machine learning algorithm capable of yielding a better performance than other algorithms for all problems. Since there is no universally superior classifier, an alternative to this would be combining classifiers, in order to explore the competence of each one locally to obtain a better overall performance [32, 14].

A Multiple Classifier System (MCS) is, then, divided in three phases [5]: Generation, Selection and Integration. In the first phase, the pool of classifiers is generated. In the second phase, a non-empty subset of classifiers from the pool is selected. In the third and last phase, the outputs of the previously selected classifiers are combined to produce the system's final output. There are two possible approaches in the Selection phase: Static Selection (SS), in which the same classifier ensemble is used to classify all test instances, or Dynamic Selection (DS), which selects specific ensembles according to each test instance.

The DS techniques are based on the idea that the pool of classifiers contain locally accurate classifiers. Therefore, their aim is to select a subset of classifiers that are best fit, according to some criterion, for classifying each test instance in particular. This strategy has been obtaining higher accuracy rates than SS techniques [20, 2, 5], in which the competence estimation of each classifier in the pool and the ensemble setting happen during the training phase.

1.1 Motivation

As described previously, the Generation phase of an MCS is responsible for creating the classifiers to form the pool. The objective is to generate locally specialists and reasonably complementary classifiers, so that they make different mistakes in different regions, and their strengths can be explored.

Figure 1.1, shows a dataset structured in an exclusive-OR (XOR) Problem configuration and exemplifies a situation in which DS techniques work with a pool of locally, but not globally, accurate classifiers. This example can be found in [19]. Consider a pool of classifiers which

contains only Perceptron 1 and Perceptron 2 from Figure 1.2(a). It can be observed that their responses are complementary, that is, every time Perceptron 1 classifies an instance as Class 1, Perceptron 2 classifies it as Class 2, and vice versa.

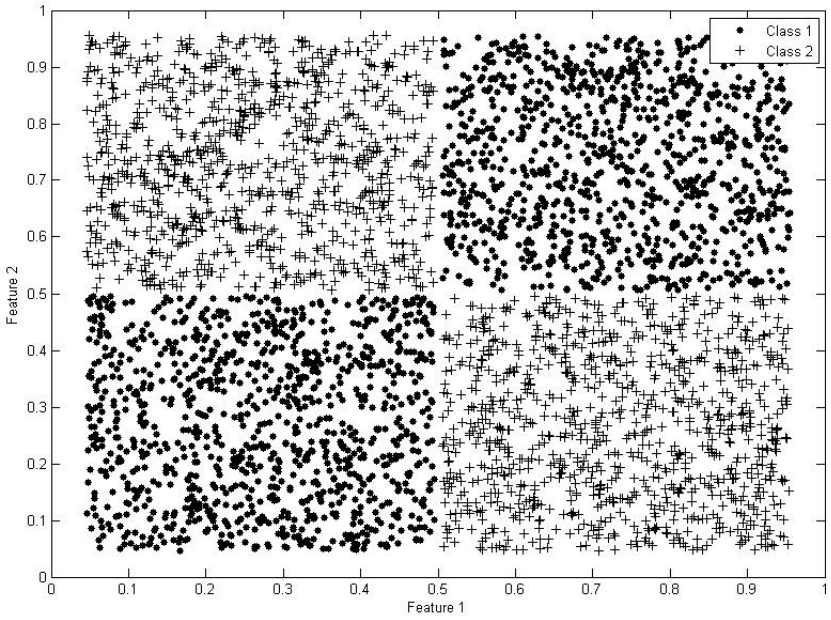
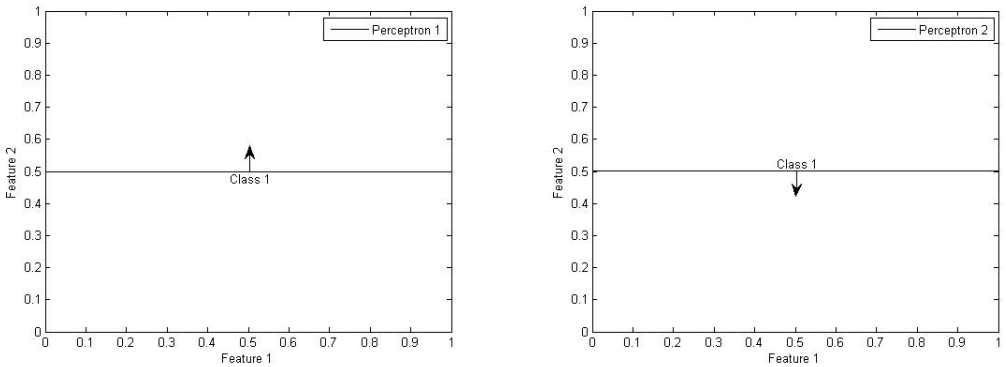


Figure 1.1: XOR Problem dataset



(a) Decision made by Perceptron 1 (b) Decision made by Perceptron 2

Figure 1.2: Locally accurate pool of Perceptrons for the XOR Problem

It is important to note that, although both classifiers have an individual accuracy rate of 50%, each one of them is highly accurate in specific regions. If the feature space is divided into four parts, as in Figure 1.3, it becomes clear that the accuracy rate of Perceptron 1 in regions **Q1** and **Q4** is 100%, whereas Perceptron 2 has an accuracy rate of 100% in regions **Q2** and **Q3**. However, as both Perceptrons never label the same instance as belonging to the same class, classical static combination rules, such as majority vote or mean, would not obtain a satisfactory accuracy rate.

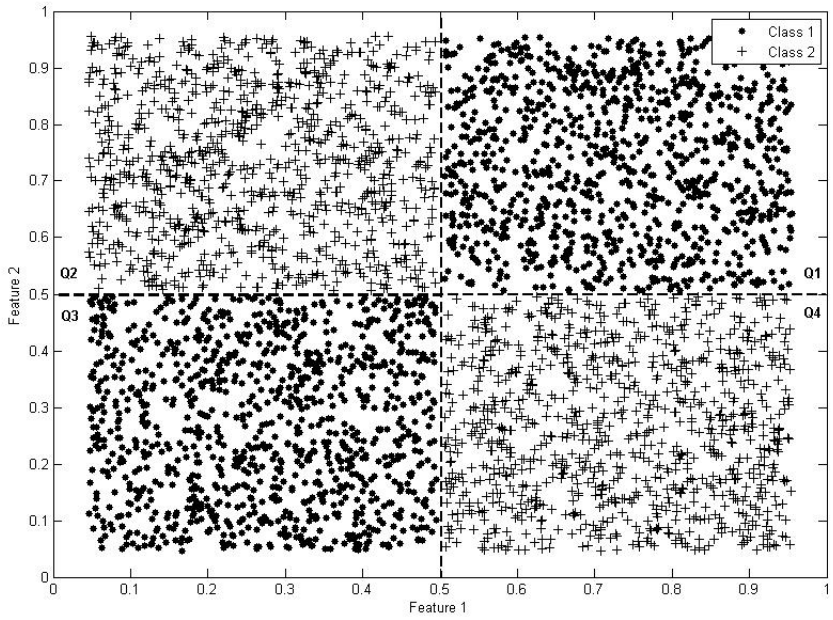


Figure 1.3: Division of the XOR Problem dataset feature space

$$Select \begin{cases} \text{Perceptron 1}, & \text{if } x_i \in \mathbf{Q1} \vee x_i \in \mathbf{Q4} \\ \text{Perceptron 2}, & \text{if } x_i \in \mathbf{Q2} \vee x_i \in \mathbf{Q3} \end{cases} \tag{1.1}$$

Nevertheless, the local accuracy of each Perceptron can be explored using DS techniques. If, for any test instance x_i, the DS technique selects the classifier to label it according to expression (1.1), an accuracy rate of 100% can be achieved. This is possible because, in the pool formed by Perceptron 1 and Perceptron 2, there is always at least one classifier able to correctly classify each instance, that is, it is guaranteed that the correct classifier to label each instance is inside the pool.

Thus, it can be observed that, for DS techniques, the presence of locally accurate classifiers in the pool is of great importance. As the Generation phase can cost a lot in terms of computational time, the methods of pool of classifiers generation for an MCS that uses DS techniques must be able to efficiently generate classifiers that are accurate in different regions of the feature space.

1.2 Problem Statement

During the Generation phase of an MCS, a reasonable number of classifiers are generated and trained to solve the same classification problem. This approach differs from the classical classification models, in which only one strong classifier is trained to find the borders between the problem's classes. As obtaining a strong classifier is much harder than weak ones, a considerable

amount of those weak classifiers are then generated to comprise a strong classifier by combining them [7, 33].

The classical methods of pool of classifiers generation are Bagging [3], Random Subspace [12] and Boosting [24], but a fair amount of variations from those methods were developed in the last years.

Bagging is a technique in which a prefixed number of training datasets are generated by bootstrapping [8] the original training dataset. Each dataset generated is then used to train one classifier, for example a Perceptron, to form the pool [16, 33].

The Random Subspace method also generates a prefixed number of training dataset using the original one. However, this is achieved by randomly reducing the number of features of the original training dataset. Each classifier in the pool is, then, trained with one of the datasets generated [16, 33].

On the other hand, Boosting is based on the idea that a lot of weak classifiers can surpass in performance a strong classifier. The generation of those weak classifiers is incremental: the algorithm generates a classifier on each step, and each of those classifiers is trained with a subset of the original training dataset. Each training dataset is generated by updating its distribution in every step, such that the classifier generated in step $n+1$ is more likely to correctly classify the instances that the classifier generated in step n missed [16, 33].

All of the methods previously mentioned require training of the classifiers in the ensemble. Even though it is common to have a pool of simple classifiers to reduce the training time, it is still a costly process. Moreover, the number of classifiers in the pool must be set beforehand, which can be disadvantageous, since it is reasonable to believe that the amount of classifiers necessary to yield a good solution depends on the problem.

1.3 Objective

This work aims to perform a study on pool of classifiers generation for DS methods, in order to implement an algorithm to generate a pool of classifiers. The size of this pool must vary with the training dataset distribution, and the generated classifiers must not be trained, so that the process can be optimized in terms of time.

The proposed algorithm is capable of producing a pool of classifiers such that each training instance is correctly classified by at least one of the hyperplanes in it, that is, the theoretical limit (oracle) for the training dataset is 100%.

This approach allows the necessary amount of locally accurate classifiers to be generated, according to the training data. Also, in order to eliminate the need to train them, a heuristic is used to obtain the hyperplanes, and, therefore, reduce the pool of classifiers generation time.

1.4 Methodology

The design started by stating the research objective, which was defined in Section 1.2, and performing the initial literature review. This last provided the basic concepts and understanding of the area, as well as an analysis of the existent techniques and methods related to the research problem. Then, a method for pool of classifiers generation was designed and implemented. The proposed method was analysed using synthetic two-dimensional datasets. Also, an experiment was then conducted using public datasets for classification, and the performance of the studied techniques and the proposed method was measured by:

1. The computational costs associated with the pool of classifiers generation process
2. The accuracy rate obtained by the generated pool, using five different DS methods

Finally, the obtained results were then analysed and compared, and the proposed approach validated.

1.5 Organization of the Dissertation

This work is organized as follows. In Chapter 2, the basic concepts of MCSs are introduced. Also, an overview of the classical methods of pool of classifiers generation is presented. The proposed method is presented in Chapter 3, along with an illustrative example of the algorithm. Analysis of the proposed method using synthetic data and experimental studies are conducted in Chapter 4. Finally, in the last chapter, experimental results are summarized and possible future works are suggested.

2

Background

2.1 Introduction

As stated by WOLPERT; MACREADY [31], each classifier has its particular competence domain, in such a way that it surpasses in performance the other algorithms. One advantage of MCSs over single classifier models is that having a pool of classifiers allows the MCSs to attempt to select a local optimal model from the available classifiers in their pool, and thus overcome the fact that there is not a single classifier optimal for all pattern recognition tasks. Another benefit of using MCS is that, by averaging the outputs of the classifiers in the pool, it is possible to avoid selecting the worst classifier, even though a better performance than the single best classifier cannot be ensured. In addition to it, aggregating many classifiers may result in a better overall classifier than any individual classifier, since many algorithms perform a local search that leads to different local optima. Moreover, the optimal classifier may not be in the classifiers space, although the latter may be expanded by combining them, increasing the space of representable functions [32, 16, 7].

The MCSs are usually structured in three phases, as Figure 2.1 illustrates. In the first phase, Generation, the training dataset is used as the input to an ensemble method, which then generates the pool of classifiers. In the Selection phase, the most adequate classifier(s) in the pool, based on some criterion, is selected to perform the classification task at hand. This is not a necessary phase for all MCS, since it is possible to always use all classifiers in the ensemble to classify the test instances. During the last phase, Integration, an aggregation scheme is used to combine the outputs of the selected classifiers [5].

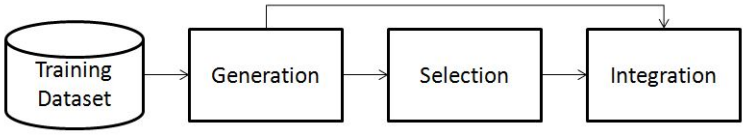


Figure 2.1: Possible phases of a MCS.

The classifiers selection can be performed statically, that is, using a subset of the pool of classifiers for all test instances, or dynamically, which means a different subset of the pool of classifiers is chosen for each test instance. since not all test instances present the same classification difficulties, it is reasonable to believe, and it has been shown, that DS techniques usually yield a better performance than SS ones. When the DS method only selects one classifier, it is called a Dynamic Classifier Selection (DCS) technique. On the other hand, when more than one classifier is chosen dynamically to perform the classification task, the process is called Dynamic Ensemble Selection (DES) [5, 2].

Regardless of the amount of classifiers to be dynamically selected, the selection itself is done based on an evaluation of competence of each classifier for a given test instance. This competence estimation can be based on numerous aspects, such as local accuracy, ranking, the concept of the oracle, probability, diversity, ambiguity, and so on. For the pool to contain competent classifiers for each test instance, and therefore be able to select an adequate subset of the ensemble, it is important that the pool is diverse and accurate [16, 5]. The generation of such pool of classifiers, as well as the classical ensemble methods, are further discussed in the following section.

2.2 Ensemble Methods

Ensemble methods are responsible for constructing, from the training dataset, the classifiers that will form the pool, during the Generation phase of a MCS [5]. Figure 2.2 illustrates this process. The resulting pool can be homogeneous, which means it contains only the same type of base classifiers, or heterogeneous, in which the pool has different types of base classifiers.

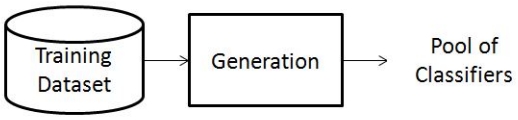


Figure 2.2: Generation of an ensemble.

The main goal of an ensemble method is to generate a pool as accurate and diverse as possible, for it is generally believed that, in order to exploit each classifier's strength and thus yield a better solution than a single strong classifier, the ensemble should contain locally expert base classifiers [7, 33].

The process of generating a pool of classifiers can be done using many general techniques: by Bayesian voting, by manipulating the training examples, by manipulating the input features, by manipulating the output targets and by injecting randomness [7, 32]. Two of the classical ensemble methods, Bagging and Boosting, generate the pool of classifiers by using different subsets of the training dataset, while the other classical method, Random Subspace, uses different

feature subspaces of the training dataset to construct the ensemble. These three ensemble methods will be presented in more details next.

2.2.1 Bagging

Bagging [3] is a pool of classifiers generation method based on bootstrapping [8], which applies random sampling with replacement to generate data subsets of a given dataset. For example, given a training dataset Γ of size N, Γ_i, a bootstrap replicate of Γ, is constructed by sampling with replacement N instances from Γ, which means Γ_i may contain several replications of an original instance in Γ, while another original instance may not be present in the subset generated.

In Bagging, each classifier c_i in the pool is trained using a bootstrap replicate (Γ_i) of the original training dataset (Γ) as its training dataset. By doing so, it is possible that misleading instances in Γ may not be present in Γ_i, which may result in a better classifier [16, 26, 33].

Algorithm 1 describes the Bagging method for any base classifier. Note that the number L of classifiers to be generated must be set beforehand. In order to avoid generating almost identical classifiers, the base classifier should be unstable, in which small changes in the training dataset result in largely different classifiers [16, 26].

Algorithm 1 Bagging method

```
1:  procedure GENERATEPOOL(L)
2:      Γ ← {z_1, z_2, ..., z_N}                    ▷ Training dataset
3:      Pool ← {}
4:      for i ← 1, L do                             ▷ L: Number of classifiers
5:          Γ_i ← bootstrapSample(Γ)
6:          c_i ← constructClassifier(Γ_i)
7:          Pool ← Pool ∪ {c_i}                     ▷ Add classifier c_i to the Pool
8:      end for
9:      return Pool
10: end procedure
```

The instances from the training dataset Γ that are not present in its bootstrap replicate Γ_i are called *out-of-bag* examples. For a training dataset of size N, the probability of each instance to be left out of Γ_i is approximately Poisson distributed with $\lambda = 1$, which yields $1/e$. Therefore, approximately 37% of the original instances of Γ are out-of-bag. Thus, better classifiers may be obtained from Γ_i than from Γ, for it is possible that outliers in the original training dataset were not selected in the process [16, 26, 33].

Furthermore, these out-of-bag examples can be used to estimate the generalization error of the generated pool of classifiers. If an instance z_j in Γ was not present in $M < L$ bootstrap replicates, z_j is an out-of-bag example for the sub-ensemble formed by those M classifiers. So, if the ensemble error on each instance in Γ is scored, the generalization error of the ensemble can be estimated. [16, 33].

2.2.2 Random Subspace

Introduced in [12], the Random Subspace method was originally proposed to build the so called "Decision Forest", a classifier created from the combination of a pool of Decision Trees. The generation of those Decision Trees is fairly simple: consider a training dataset Γ with N instances and dimension d. Each Decision Tree in the pool is generated using a training dataset Γ_i of dimension $k < d$, whose k features are a randomly selected subset of the original d features. Γ_i contains the same N instances from Γ, but represented only with the k randomly selected features [16].

The generation phase of the Random Subspace method is described with more detail in Algorithm 2, for any base classifier. It can be observed that the algorithm requires two parameters: the number L of classifiers in the pool and the dimensionality k of the generated subspaces.

Algorithm 2 Random Subspace method

```
1:  procedure GENERATEPOOL(k,L)
2:      Γ ← {z_1, z_2, ..., z_N}                    ▷ Training dataset
3:      Pool ← {}
4:      for i ← 1, L do                              ▷ L: Number of classifiers
5:          Γ_i ← selectRandomSubspace(Γ, k)         ▷ k: Dimension of subspaces
6:          c_i ← constructClassifier(Γ_i)
7:          Pool ← Pool ∪ {c_i}                      ▷ Add classifier c_i to the Pool
8:      end for
9:      return Pool
10: end procedure
```

The Random Subspace method can be advantageous for problems which the amount of training instances is small compared to the amount of features, for the method reduces the original training dataset dimensionality while maintaining the number of instances in all generated subspaces. Also, for problems with redundant data, using random subspaces to generate the classifiers may yield a better solution than using the original feature space, since the former performs, albeit randomly, a feature selection. For those characteristics, the Random Subspace method have been widely used for problems with high dimensional data, such as facial recognition [6], traffic flow prediction [28], functional magnetic resonance imaging data analysis [18], cancer diagnosis [1], among others [16, 26].

2.2.3 Boosting

The main idea of Boosting [24] and its variants is to create several weak classifiers incrementally and combine them to obtain a better prediction rule. As opposed to Bagging and Random Subspace, in which the process of generating the classifiers is random and parallel, the Boosting method generates the pool of classifiers in a sequential and, depending on which variant, strictly deterministic manner [16, 26].

The process of generating each classifier is fairly simple: consider a training dataset Γ. In the first step of the algorithm, a base classifier c_1 is trained with the original training dataset. But, since c_1 is a weak classifier, it will probably not be able to satisfactorily generalize the problem. So, the error of c_1 is evaluated using the original training dataset, and another training dataset, Γ_2, is generated from the one used to train c_1. The data distribution of Γ_2 is derived so that it makes the mistakes made by c_1 more evident. This is performed in order to make the incorrectly classified instances by c_1 more apparent. In the next step, classifier c_2 is trained with Γ_2, so it will probably classify correctly the instances that c_1 did not. c_2 will then be tested, and the distribution of the new dataset Γ_3 to be created from Γ_2 will be adjusted according to the classification errors of c_1. This process is repeated until the amount of classifiers set to be generated is reached. In short, the Boosting method generates in each step a classifier more apt to correctly classify the instances the classifier created in the previous step misclassified [16, 26, 33].

Algorithm 3 describes the general Boosting method. Note that the amount of classifiers to be generated is an input to the algorithm.

Algorithm 3 General Boosting method

```
1:  procedure GENERATEPOOL(L)
2:      Γ ← {z_1, z_2, ..., z_N}                    ▷ Training dataset
3:      Γ_i ← Γ
4:      Pool ← {}
5:      for i ← 1, L do                              ▷ L: Number of classifiers
6:          c_i ← constructClassifier(Γ_i)
7:          ε_i ← test(Γ, c_i)                       ▷ Evaluate the error of c_i
8:          Γ_{i+1} ← adjustDistribution(Γ_i, ε_i)
9:          Pool ← Pool ∪ {c_i}                      ▷ Add classifier c_i to the Pool
10:     end for
11:     return Pool
12: end procedure
```

In AdaBoost [11], the most influential boosting algorithm, the data distribution adjustment can be achieved by resampling, in which the training instances are sampled in each step to form the desired distribution, or re-weighting, in which the training instances are weighted in each iteration according to the sample distribution. By doing so, the margins between the training instances are maximized [7, 16, 33].

The Boosting algorithms have been acclaimed for their high accuracy, robustness and wide applicability, which includes face recognition [30], text categorization [25], routing [22], medical diagnosis [27], and so on. Aside from AdaBoost, variants include Arc-x4 [4], BrownBoost [9] and RobustBoost [10], among others.

3

The Proposed Method

The proposed method is presented in Algorithm 4. It consists of generating hyperplanes iteratively in such a way that each instance in the training set must be correctly classified by at least one of the base classifiers in the pool, that is, the Oracle for the training dataset is 100%. The base classifiers chosen to produce such hyperplanes were Perceptrons. Therefore, the algorithm itself discovers the amount of Perceptrons needed in the pool, according to the training dataset. Furthermore, the use of a heuristic to obtain the Perceptrons weights makes it not necessary to train them.

Algorithm 4 Proposed method

1: **procedure** GENERATEPOOL
2: $\Gamma \leftarrow \{z_1, z_2, ..., z_N\}$ ▷ Training dataset
3: $C \leftarrow \{c_1, c_2, ..., c_k\}$ ▷ Problem classes
4: $Pool \leftarrow \{\}$
5: **while** $\Gamma \neq \{\}$ **do**
6: **for** $j \leftarrow 1, k$ **do**
7: $R(j) \leftarrow centroid(c_j)$ ▷ Centroid of class j
8: **end for**
9: $d \leftarrow max(pairwiseDistance(R))$ ▷ Maximum distance between centroids
10: $a, b \leftarrow findIndex(d)$
11: $midPoint \leftarrow (R(a) + R(b))/2$
12: $normal \leftarrow (R(a) - R(b))/d$
13: $w_p \leftarrow \{normal\}$ ▷ Perceptron p weights
14: $\theta_p \leftarrow -midPoint \cdot normal$ ▷ Perceptron p bias
15: $p \leftarrow perceptron(w_p, \theta_p)$
16: **for** $i \leftarrow 1, N$ **do**
17: **if** $test(p, z_i) = label(z_i)$ **then** ▷ Perceptron p classifies instance i correctly
18: $\Gamma \leftarrow \Gamma - \{z_i\}$ ▷ Excludes instance i from dataset
19: **end if**
20: **end for**
21: $Pool \leftarrow Pool \cup \{p\}$ ▷ Add Perceptron p to the Pool
22: **end while**
23: **return** *Pool*
24: **end procedure**

Figure 3.1(a) shows a training dataset with $N = 350$ instances and $k = 5$ classes. The step-by-step execution of the algorithm for this example happens as follows. Steps 1 and 2 of Algorithm 4 consist of assigning to Γ the entire training dataset and to C the five classes of the problem ({1,2,3,4,5}). Step 4 assigns the Pool to an empty set.

In the first iteration of the algorithm's outer loop, the centroids of each of the five classes are calculated and stored in the matrix R (steps 6 to 8). All five centroids are represented by asterisks (*) in Figure 3.2(a). Then, in steps 9 and 10, the two most distant points in R are chosen, in this case *R(3)* and *R(5)*, and their classes *a,b = 3,5* identified. In Figure 3.2(a), Class 3 and Class 5 centroids are the red asterisks. From step 11 to step 15, the weights w_p and bias θ_p of Perceptron 1 are calculated, so that it separates *R(3)* and *R(5)* halfway between them, as can be observed in Figure 3.2(a). In steps 16 to 20, each instance in Γ is tested with Perceptron 1, and the instances correctly classified are then excluded from Γ. Since Perceptron 1 correctly classifies all instances of Class 3 and Class 5, as can be seen in Figure 3.2(a), by the end of that iteration Γ no longer contains instances of both classes, though it still contains all instances of the other classes. In step 21 the Perceptron 1 is then added to the Pool.

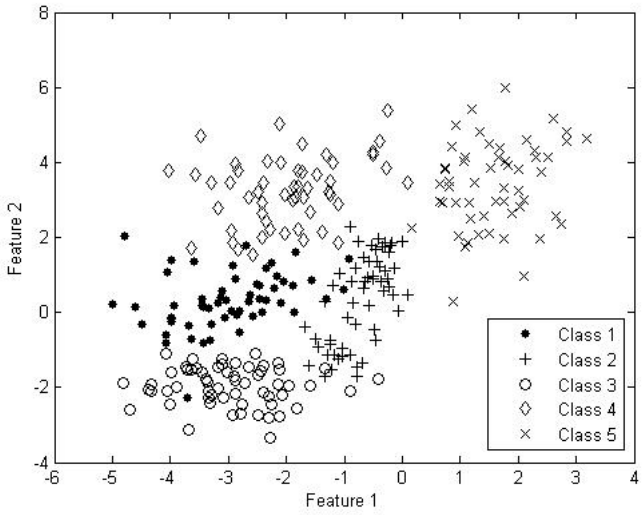
In the second iteration of the outer loop, Γ contains all instances of Class 1, Class 2 and Class 4, so the centroids of these classes are calculated from steps 6 to 8 and stored in R. These centroids are represented by the three asterisks in Figure 3.2(b). The centroids chosen in steps 9 and 10 are *R(2)* and *R(4)*, since they are the most distant to each other, as can be noticed in Figure 3.2(b), in which they are the red asterisks. Then, the weights w_p and bias θ_p of Perceptron 2 are calculated from step 11 to step 15, dividing the space between *R(2)* and *R(4)* right in the middle, as Figure 3.2(b) shows. Perceptron 2 is then used to test each instance in Γ from steps 16 to 20, and the instances that remain in Γ are the ones Perceptron 2 classifies incorrectly. Since Class 2 and Class 4 are not linearly separable, Perceptron 2 is not able to eliminate all instances of those classes. Perceptron 2 is added to the Pool in step 21.

In the third iteration, Γ still has instances of Class 1, Class 2 and Class 4, so their centroids *R(1)*, *R(2)* and *R(4)* are calculated from steps 6 to 8. It can be observed that, since most of Class 2 and Class 4 instances were eliminated in the previous iteration, their centroids changed. It did not happened to centroid of Class 1, as neither Perceptron 1 nor Perceptron 2 were able to classify Class 1 instances. The red asterisks in Figure 3.2(c) show that centroids *R(1)* and *R(4)* are the most distant ones in this iteration, with centroid *R(2)* in black. Perceptron 3 is then created from step 11 to step 15 so that it splits the plane in a half. From steps 16 to 20, each instance remaining in Γ is then tested with Perceptron 3, and the instances it correctly classifies are further eliminated from Γ. It can be observed that the remaining Class 4 instance is correctly classified by Perceptron 3, so Γ only possesses Class 1 and Class 2 instances after the third iteration. In step 21 the Perceptron 3 is then added to the Pool.

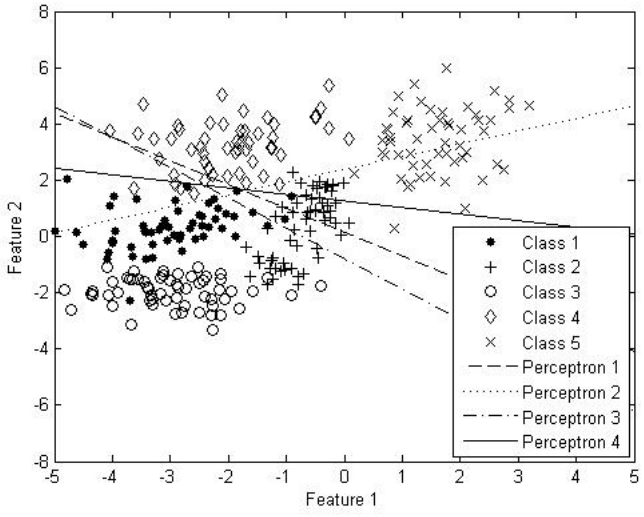
In the fourth and last iteration, Γ contains only 4 instances, 3 of Class 1 and 1 of Class 2, as showed in Figure 3.2(d). Centroids *R(1)* and *R(2)* are calculated from steps 6 to 8 and chosen to calculate the weights w_p and bias θ_p of Perceptron 4 from steps 11 to 15. Each instance in

Γ is tested with Perceptron 4 in steps 16 to 20, and since it correctly classifies all 4 remaining instances, they are eliminated and Γ turns into an empty set. Perceptron 4 is then added to the Pool in step 21, and the algorithm leaves the outer loop, returning the Pool in step 23.

Figure 3.1 shows the entire training dataset with all four Perceptrons generated by the proposed method. The spatial disposition of the only four Perceptrons necessary to "cover" the entire dataset can be observed.

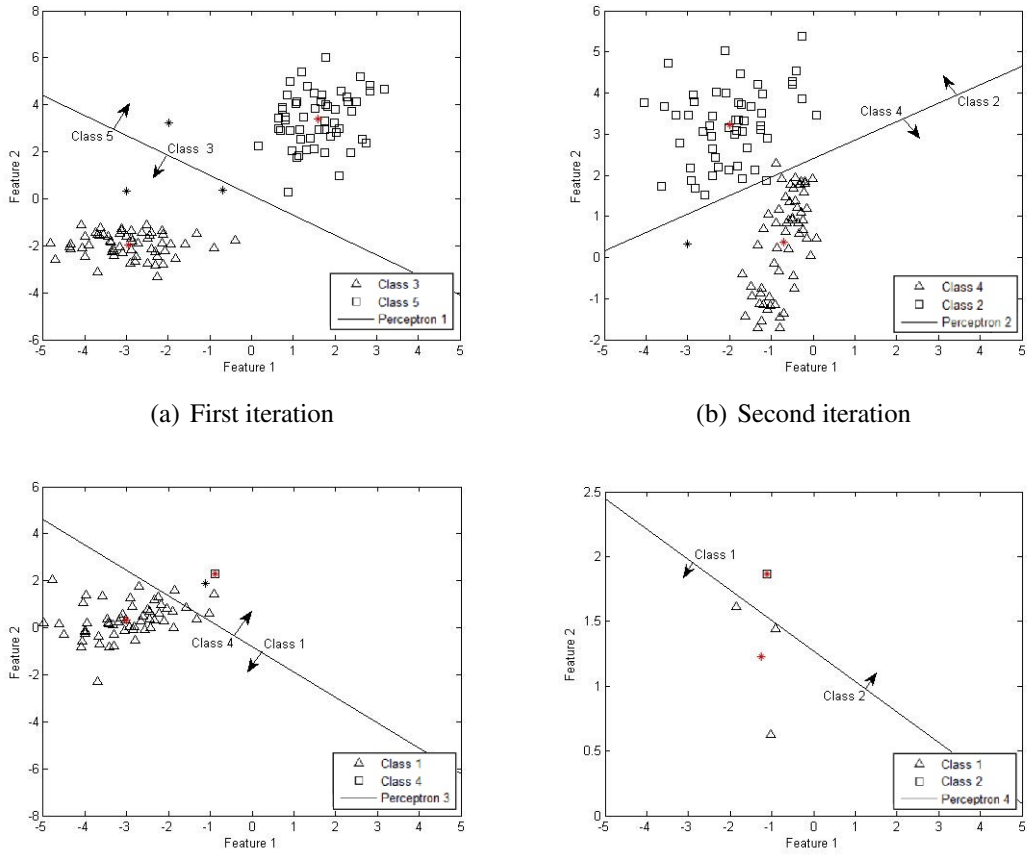


(a) Toy problem training dataset



(b) Perceptrons generated by the proposed method

Figure 3.1: Toy problem



(a) First iteration

(b) Second iteration

(c) Third iteration

(d) Last iteration

Figure 3.2: Generation of Perceptrons using toy problem

4

Experimental study

4.1 Performance analysis

In order to better understand the mechanics of the proposed method and identify in which situations it works best or fails, analysis using synthetic data were performed regarding the effect of the following factors on the accuracy rate of the method: differences on test data distribution, problems with complex borders between the classes and problems with multiple classes.

As the pool generated by the proposed method is generally not fit for DES techniques, since it generates very few and diverse classifiers, it was chosen five DCS techniques, instead of DES techniques, to evaluate the accuracy, in order to perform a fair comparison between the proposed method and the three classical pool of classifiers generation methods. The DS techniques used in the following analysis were the accuracy-based methods Overall Local Accuracy (OLA) and Local Class Accuracy (LCA), the behaviour-based method Multiple Classifier Behaviour (MCB) and the probabilistic-based methods A Priori and A Posteriori, which are the DCS techniques in [5].

4.1.1 Variations on data disposition

Since the data disposition has a strong influence on the proposed method's pool generation process, the impact on the performance due to differences between the data distributions of the training and test datasets was analysed.

The purpose of this analysis is to investigate the sensitivity of the proposed method in regards to the following three scenarios of variation on the distribution of the test dataset compared to the distribution of the training dataset:

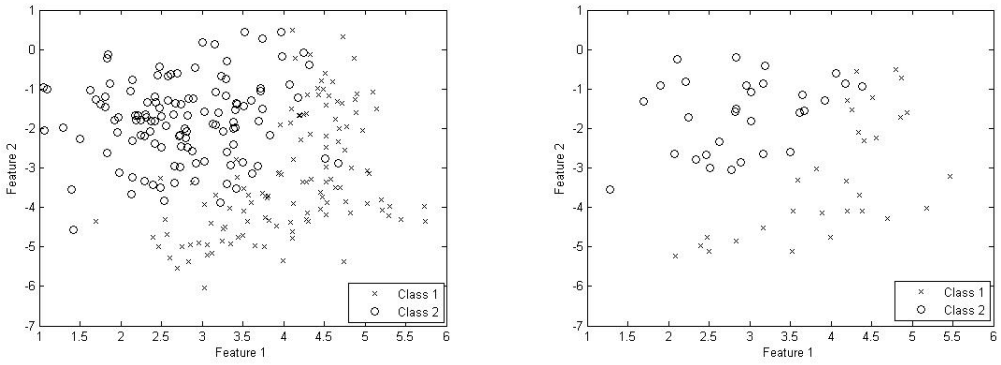
I The classes in the test dataset are farther from each other, but in the same relative position to each other as in the training dataset;

II The classes in the test dataset are closer to each other, but in the same relative position to each other as in the training dataset;

III The classes have a different relative position to each other in comparison with the training dataset.

The artificial training dataset used in this analysis as a toy problem is shown in Figure 4.1(a). It is a balanced binary classification problem with 240 instances, in which the training dataset contains 80% of those instances and the test datasets containing the remaining ones. The pool generation by the proposed method is described in Figure 4.2.

Figure 4.1(b) shows the artificial test dataset generated with the same distribution as the training dataset in Figure 4.1(a). This dataset was used as a reference for comparison with the case studies described previously.



(a) Toy problem training dataset (b) Test dataset

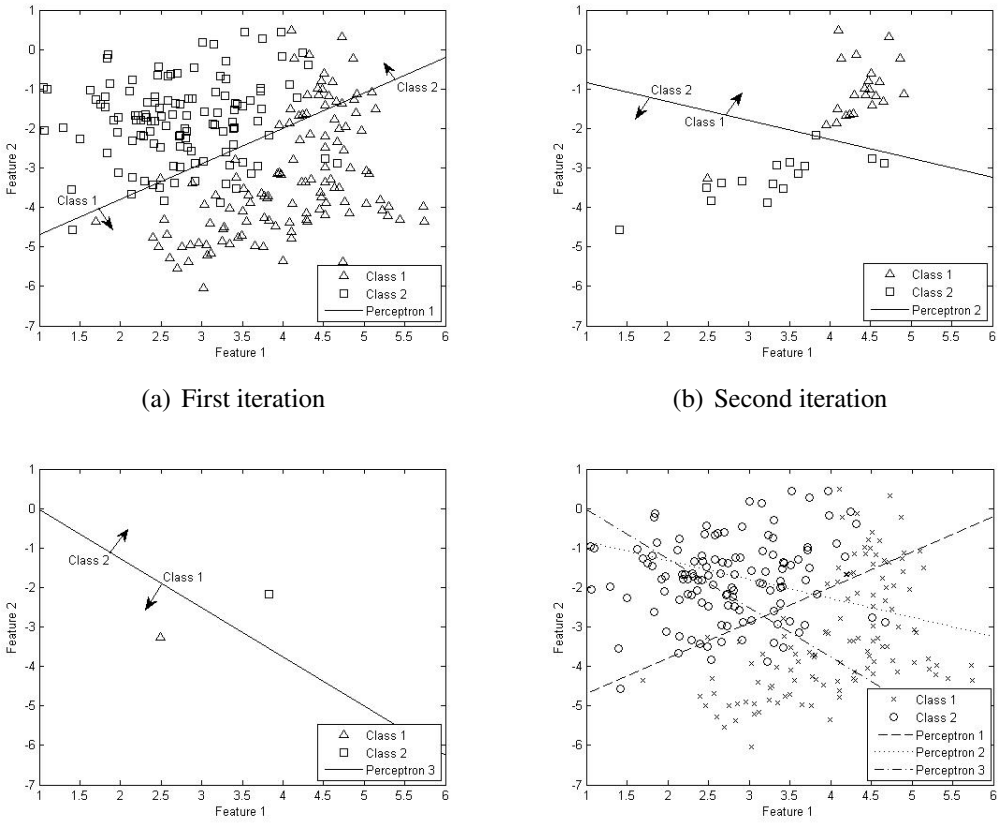
Figure 4.1: Toy problem training and test datasets generated with the same distribution

The test datasets used in the analysis of case I scenario, in which the distance between the classes is increased, are described in Figure 4.3. In Test 1 dataset, Class 1 instances are farther from Class 2 instances than in the original test dataset. Both classes are even further away in Test 2 dataset.

For the case II scenario analysis, in which the distance between the classes is reduced, the test datasets used are described in Figure 4.4. The gap between Class 1 instances and Class 2 instances in Test 3 dataset is smaller than in the original test dataset. This gap is further shortened in Test 4 dataset and even further in Test 5 dataset, to become non-existent in Test 6 dataset.

The case scenario III, in which the relative position between the classes is changed, the test datasets used to evaluate the proposed method can be observed in Figure 4.4. The original test dataset was modified so that the position of the classes in relation to each other gradually shifted from Test 7 to Test 9 datasets.

The pool of Perceptrons in Figure 4.2(d) was tested with all test datasets previously presented and was evaluated in terms of accuracy using the DCS techniques formerly stated. For comparison, the same techniques were applied on the pools generated by Bagging, Random Subspace and Boosting, all three with the same number of Perceptrons as the proposed method's pool size (N_{PM}=3 Perceptrons). Also, a pool of size N_{100} = 100 was generated with Bagging,



(a) First iteration

(b) Second iteration

(c) Last iteration

(d) Disposal of generated pool over training dataset

Figure 4.2: Generation of Perceptrons using toy problem

Random Subspace and Boosting and tested for each case scenario as well. The Boosting algorithm used in this analysis was AdaBoost. The experiment was executed 20 times for the last three methods for both pool sizes. The accuracy yielded by the four methods of pool of classifiers generation can be observed in Table 4.1 for all DCS techniques aforementioned.

For case scenario I, the disposal of the pool of Perceptrons generated by the proposed method over the test datasets can be seen in Figure 4.6. It can be observed that, since the relative position of the classes with respect to each other have not changed, the position of the Perceptrons favours the correct classification in this case, given that the border region between the classes is increasing.

As expected, the accuracy for case I scenario increased for all methods, in comparison with the original test dataset, as Table 4.1 shows. Note that, from Test 1 to Test 2, the accuracy of the proposed method only increased for the A Posteriori DS technique, whereas all other pool of classifiers generation methods had its performance increased with the increase in the distance between the classes, which suggests the proposed method is more resistant to changes of this nature, in comparison with the other methods.

The pool generated by the proposed method over the test datasets from case scenario II are shown in Figure 4.7. Note that, even though the border region between the classes is being

Table 4.1: Mean and standard deviation of accuracy for the pool of classifiers generation methods using DCS techniques for the three case scenarios. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

(a) OLA

Dataset	Proposed Method	Bagging		Random Subspace		Boosting	
	N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Original test	95.00	**95.66** (2.05)	93.00 (4.91)	90.58 (7.57)	94.66 (2.62)	94.16 (1.98	88.91 (9.22)
Test 1	98.33	**99.33** (1.25)	97.66 (3.39)	94.33 (5.95)	98.33 (1.20)	98.75 (0.91)	91.75 (9.38)
Test 2	98.33	**99.92** (0.37)	99.33 (2.25)	98.16 (2.58)	99.83 (0.51)	**99.92** (0.37)	93.25 (8.91)
Test 3	90.00	**91.67** (3.37)	89.58 (3.93)	87.33 (7.20)	90.75 (2.56)	90.75 (1.75)	84.25 (8.89)
Test 4	**86.67**	82.67 (3.12)	83.08 (3.75)	83.00 (4.70)	83.92 (2.37)	85.08 (2.26)	79.67 (8.08)
Test 5	**78.33**	74.08 (3.31)	75.50 (3.29)	73.67 (1.99)	76.25 (3.32)	78.08 (2.97)	72.33 (6.60)
Test 6	**75.00**	67.00 (4.03)	68.83 (3.59)	67.25 (3.34)	69.16 (3.22)	71.58 (3.72)	70.41 (6.11)
Test 7	41.67	**48.00** (7.14)	45.83 (9.71)	53.50 (12.9)	46.25 (6.77)	45.16 (4.21)	44.25 (7.26)
Test 8	**31.67**	25.25 (4.30)	25.58 (6.69)	28.83 (8.95)	24.83 (5.23)	25.66 (3.76)	30.16 (5.35)
Test 9	11.67	11.41 (0.97)	11.41 (1.89)	**12.33** (2.98)	10.67 (1.47)	12.16 (1.44)	16.33 (7.77)

(b) LCA

Dataset	Proposed Method	Bagging		Random Subspace		Boosting	
	N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Original test	95.00	**96.50** (1.70)	94.08 (4.82)	90.75 (7.48)	95.75 (2.26)	95.41 (1.70)	89.08 (8.82)
Test 1	98.33	**99.25** (1.26)	97.75 (3.30)	94.58 (5.56)	98.25 (1.14)	98.66 (0.87)	91.83 (9.20)
Test 2	98.33	**99.92** (0.37)	99.41 (2.25)	98.33 (2.29)	99.83 (0.51)	**99.92** (0.37)	93.66 (8.89)
Test 3	90.00	92.16 (2.76)	90.25 (4.05)	87.50 (7.26)	**92.25** (1.97)	90.91 (1.57)	84.58 (8.54)
Test 4	**86.67**	83.83 (3.24)	84.08 (3.44)	83.08 (4.75)	85.58 (1.97)	85.50 (1.80)	79.50 (7.91)
Test 5	76.67	74.58 (3.32)	75.67 (2.93)	73.41 (2.19)	75.41 (2.52)	**77.67** (1.74)	72.50 (6.58)
Test 6	**75.00**	68.67 (3.57)	69.58 (3.70)	67.67 (3.35)	69.08 (2.98)	72.83 (2.23)	70.50 (6.35)
Test 7	45.00	47.16 (6.12)	46.16 (9.64)	**53.16** (13.3)	47.50 (5.93)	43.91 (3.63)	44.33 (7.18)
Test 8	**30.00**	24.16 (3.64)	24.41 (6.51)	29.00 (9.05)	24.00 (4.20)	25.41 (3.05)	29.50 (5.07)
Test 9	11.67	10.50 (0.78)	11.00 (2.38)	**12.00** (3.31)	10.08 (1.00)	11.16 (1.22)	15.83 (7.97)

(c) MCB

Dataset	Proposed Method	Bagging		Random Subspace		Boosting	
	N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Original test	95.00	95.66 (2.05)	94.83 (2.69)	90.58 (7.57)	**96.00** (2.12)	94.16 (1.98)	94.83 (1.42)
Test 1	98.33	**99.33** (1.25)	98.67 (1.27)	94.33 (5.95)	98.41 (1.66)	98.75 (0.91)	98.75 (0.74)
Test 2	98.33	99.92 (0.37)	99.92 (0.37)	98.16 (2.58)	99.83 (0.51)	99.92 (0.37)	**100.0** (0.00)
Test 3	90.00	91.67 (3.37)	90.91 (3.64)	87.33 (7.20)	**92.16** (2.16)	90.75 (1.75)	89.58 (2.69)
Test 4	**86.62**	82.67 (3.12)	83.75 (3.41)	83.00 (4.70)	85.83 (2.19)	85.25 (2.37)	84.08 (1.81)
Test 5	**78.24**	74.08 (3.31)	76.00 (3.43)	73.67 (1.99)	73.83 (2.16)	78.08 (2.97)	77.16 (2.10)
Test 6	**75.00**	67.00 (4.03)	68.25 (4.09)	67.25 (3.34)	67.00 (3.52)	71.50 (3.85)	71.00 (3.39)
Test 7	42.51	48.00 (7.14)	48.41 (6.82)	**53.50** (12.9)	47.41 (12.9)	45.16 (4.21)	44.67 (4.31)
Test 8	**30.51**	25.25 (4.30)	25.16 (5.16)	28.83 (8.95)	25.50 (5.16)	25.91 (3.76)	26.16 (2.42)
Test 9	11.67	11.41 (0.97)	11.41 (1.55)	12.33 (2.98)	10.58 (0.97)	**12.16** (1.44)	11.41 (1.55)

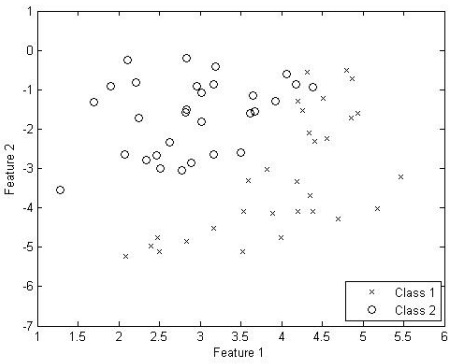
Table 4.1: Mean and standard deviation of accuracy for the pool of classifiers generation methods using DCS techniques for the three case scenarios. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons.

(d) A Priori

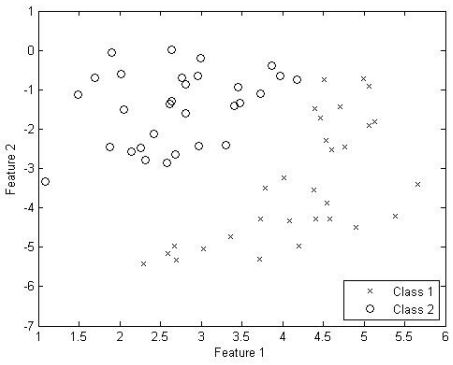
Dataset	Proposed Method	Bagging		Random Subspace		Boosting	
	N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Original test	**96.67**	95.08 (2.38)	94.50 (2.96)	90.58 (7.57)	95.83 (2.13)	94.75 (1.64)	95.25 (1.11)
Test 1	98.33	**99.33** (1.25)	98.67 (1.27)	94.33 (5.95)	98.41 (1.66)	98.75 (0.91)	98.75 (0.74)
Test 2	98.33	99.92 (0.37)	99.92 (0.37)	98.16 (2.58)	99.83 (0.51)	99.92 (0.37)	**100.0** (0.00)
Test 3	88.33	90.16 (3.50)	89.08 (3.39)	85.83 (6.34)	**90.25** (2.11)	88.58 (1.35)	87.58 (2.67)
Test 4	**86.67**	82.67 (3.17)	83.75 (3.19)	83.08 (4.62)	85.91 (2.26)	85.33 (2.45)	84.75 (2.05)
Test 5	**80.00**	74.58 (3.37)	76.25 (3.28)	74.00 (2.18)	74.50 (2.10)	78.83 (2.23)	77.25 (1.89)
Test 6	**73.33**	67.83 (3.38)	67.75 (3.75)	66.75 (2.56)	66.33 (2.62)	71.67 (2.75)	70.67 (2.77)
Test 7	48.33	49.16 (8.19)	49.33 (7.71)	**55.58** (12.6)	49.00 (8.11)	46.00 (5.08)	47.83 (4.36)
Test 8	**33.33**	24.41 (4.96)	25.50 (4.52)	29.33 (8.74)	25.16 (4.64)	27.83 (4.01)	27.41 (3.80)
Test 9	11.67	11.67 (0.76)	11.58 (0.85)	**12.75** (2.71)	11.41 (0.81)	11.58 (0.37)	11.83 (1.51)

(e) A Posteriori

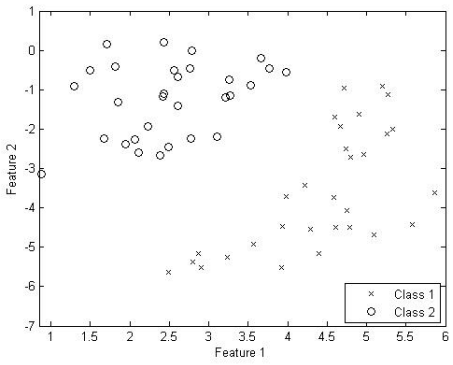
Dataset	Proposed Method	Bagging		Random Subspace		Boosting	
	N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Original test	95.00	95.75 (1.83)	95.16 (2.58)	90.67 (7.44)	**95.83** (2.13)	95.16 (1.52)	95.33 (0.87)
Test 1	98.33	98.50 (1.42)	98.41 (0.85)	94.50 (5.49)	98.16 (1.19)	98.41 (0.65)	**98.58** (0.61)
Test 2	**100.0**	99.92 (0.37)	**100.0** (0.00)	98.33 (2.29)	99.83 (0.51)	99.92 (0.37)	**100.0** (0.00)
Test 3	86.67	88.75 (3.10)	88.25 (2.72)	85.58 (6.26)	**89.75** (2.04)	87.91 (1.31)	87.75 (2.31)
Test 4	85.00	84.41 (2.37)	84.67 (2.62)	83.16 (4.67)	**86.08** (2.04)	85.41 (1.06)	84.16 (2.44)
Test 5	**78.33**	75.25 (3.07)	75.41 (2.85)	75.16 (2.01)	76.16 (2.16)	78.00 (1.49)	76.41 (1.55)
Test 6	**73.30**	67.08 (3.51)	68.25 (2.98)	67.67 (2.67)	67.16 (2.03)	71.67 (2.80)	71.33 (2.84)
Test 7	48.33	47.75 (6.67)	48.16 (6.88)	**52.75** (13.2)	45.67 (6.93)	44.66 (4.76)	47.16 (4.36)
Test 8	**31.67**	24.16 (3.22)	24.00 (3.95)	27.67 (9.21)	22.25 (2.87)	26.83 (3.66)	27.25 (4.05)
Test 9	11.67	11.41 (0.61)	11.33 (0.87)	**12.83** (2.81)	11.50 (0.51)	11.58 (0.37)	11.41 (0.97)



(a) Original test dataset



(b) Test 1 dataset



(c) Test 2 dataset

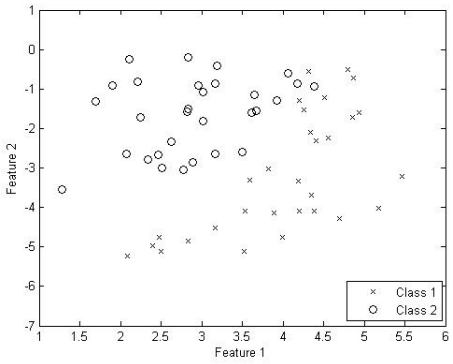
Figure 4.3: Test datasets used in the analysis of case I.

reduced, the feature space is still rather covered by the three Perceptrons in the pool, considering that the relative position of the classes with respect to each other are the same.

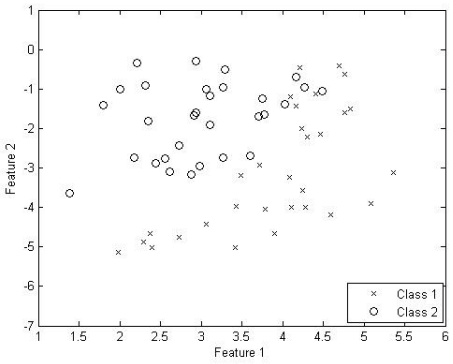
Table 4.1 shows that, as the border region diminished, the performance of all methods was degraded, as expected. It can be observed that, from Test 3 to Test 6, the accuracy of the proposed method only decreased 15 percentage points at most. For Bagging, the performance was degraded by at least 20 percentage points on average, and for Random Subspace the minimum accuracy decrease was of 19 percentage points overall. On the other hand, the decrease in the accuracy of Boosting was of at least 16 percentage points on average, the smaller performance degradation between the three classical methods and the closest to the one presented by the proposed method, though the later was less sensitive to the reduction of the classes boundaries. This also implies the proposed method possess a certain resistance to changes in the distances between the classes, as long as their dispositions are the same as in the training dataset.

Figure 4.8 shows the disposal of the Perceptrons over the test datasets from case scenario III. It can be seen that the coverage of the feature space by the three Perceptrons in the pool was strongly affected by the change in the relative position of the classes with respect to each other.

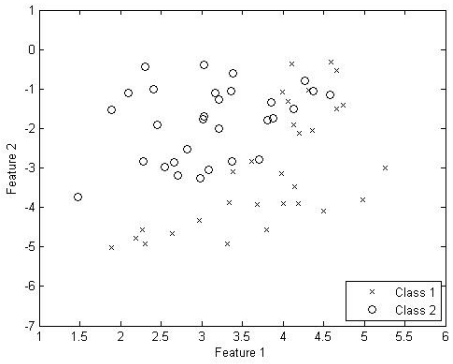
As expected, the accuracy of all methods was degraded by the alteration in the disposition of the classes, as Table 4.1 shows. It can be observed that, from Test 7 to Test 8, the accuracy of the proposed method decreased less percentage points on average, for most DS techniques, than



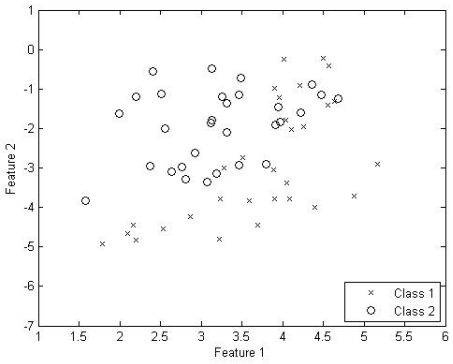
(a) Original test dataset



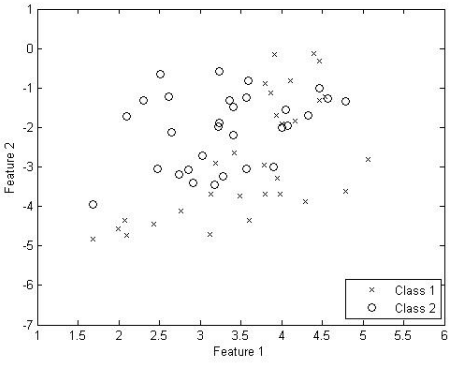
(b) Test 3 dataset



(c) Test 4 dataset



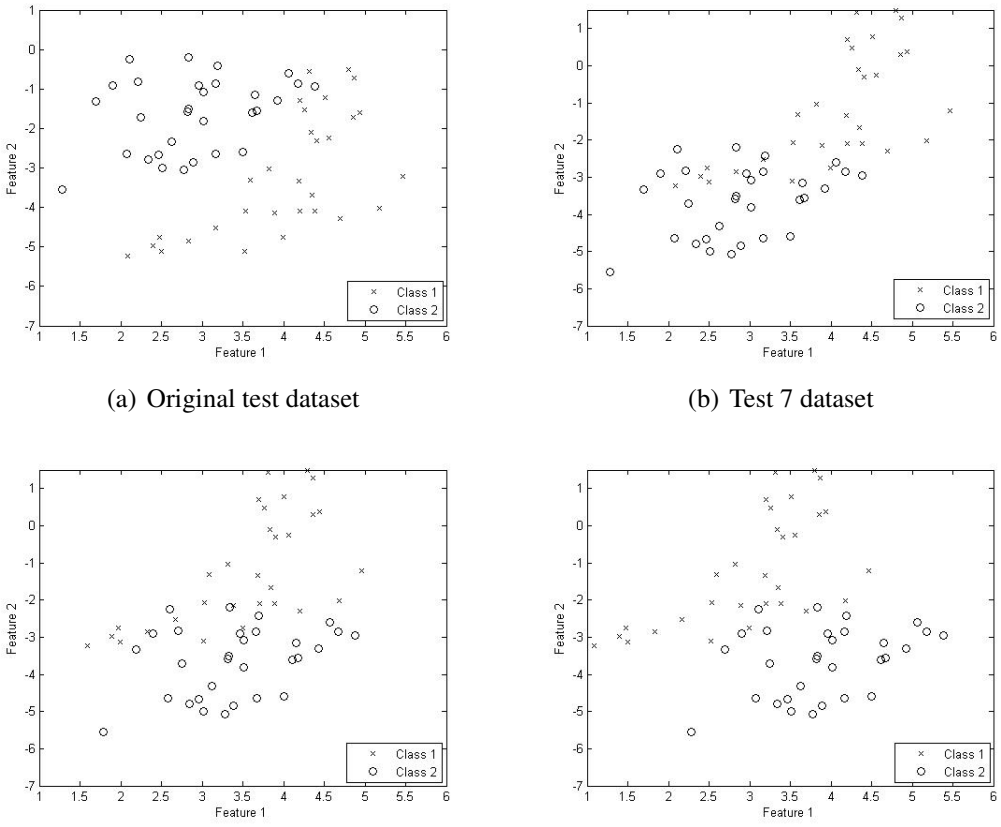
(d) Test 5 dataset



(e) Test 6 dataset

Figure 4.4: Test datasets used in the analysis of case II.

the other three methods. Since the data distribution from Test 7 and Test 8 reasonably resemble the data distribution from the original test dataset, this shows again the resistance presented by the proposed method in the previous cases. However, from Test 8 to Test 9, the accuracy of all methods were approximately the same. As the classes in Test 9 are almost in opposite positions compared to the original test dataset, this result was expected.



(a) Original test dataset

(b) Test 7 dataset

(c) Test 8 dataset

(d) Test 9 dataset

Figure 4.5: Test datasets used in the analysis of case III.

4.1.2 P2 Problem

Another important aspect to analyse is the effect of complex decision boundaries in the performance of the proposed method. To do so, the P2 Problem [29], which consists of four complex boundaries and has no overlapping of classes, was used to test the accuracy of the proposed method and compare to the remaining pool of classifiers generation methods presented in this work.

The P2 is a bidimensional two-class problem in which each decision region is delimited by one or more of the following four simple polynomial and trigonometric functions:

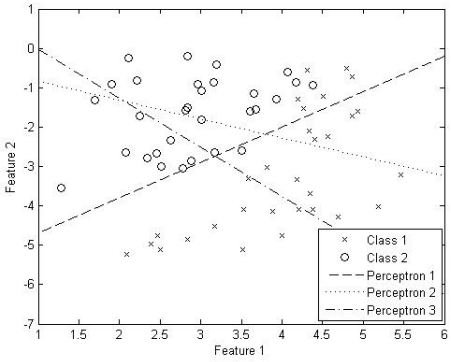
$$E1(x) = \sin(x) + 5 \tag{4.1a}$$

$$E2(x) = (x-2)^2 + 1 \tag{4.1b}$$

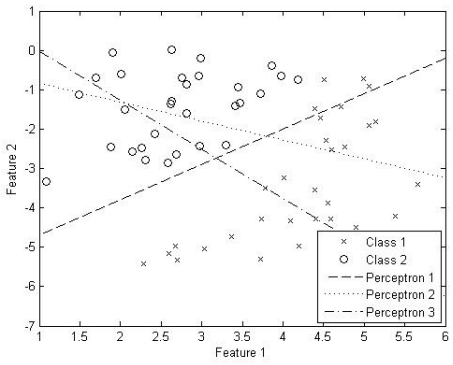
$$E3(x) = -0.1x + 0.6\sin(4x) + 8 \tag{4.1c}$$

$$E4(x) = \frac{(x-10)^2}{2} + 7.902 \tag{4.1d}$$

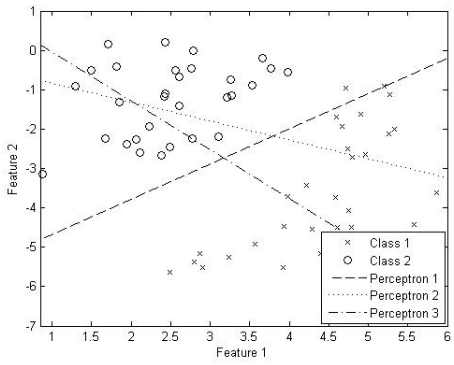
Figure 4.9 illustrates the decision boundaries, delimited by equations (4.1), and the class



(a) Original test dataset



(b) Test 1 dataset



(c) Test 2 dataset

Figure 4.6: Disposal of generated pool over case scenario I datasets

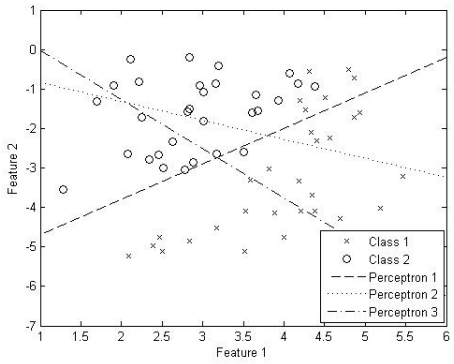
distribution of the P2 Problem. The equation $E4(x)$ was modified from the original equation in [29] to make the areas occupied by each class approximately the same.

For this analysis, the P2 Problem dataset generated consists of 1000 instances, as can be observed in Figure 4.10, and 20 replications of it were produced. For each replication, 75% of the instances were randomly selected to form the training dataset and the remaining instances were used for testing. The DCS techniques used to evaluate the methods were the same as the previous analysis, and for Bagging, Random Subspace and Boosting, the pool size was set to N_{PM} and N_{100} as well. AdaBoost was also used as the Boosting representative algorithm.

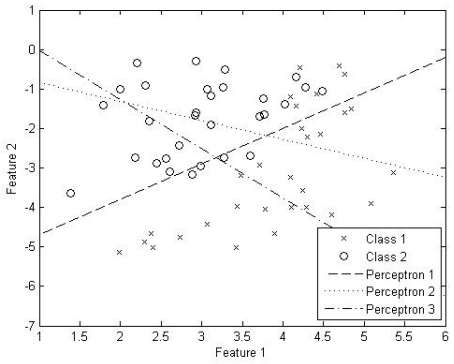
The mean and standard deviation of the pool size generated by the proposed method was N_{PM} = 3.20 (0.41). Note that it generated very few Perceptrons for this problem, despite the complexity of the boundaries between the classes. Figure 4.11 shows a pool generated by the proposed method using one of the replications. It can be observed that Perceptron 1 and Perceptron 2 have almost completely opposite responses, which suggests the method generates quite diverse classifiers.

The execution time of the four pool of classifiers generation methods are in Table 4.2. It can be noticed that the proposed method is around ten times faster than the other three methods for the same pool size, and almost eighty times faster for a pool size of N_{100} = 100 Perceptrons.

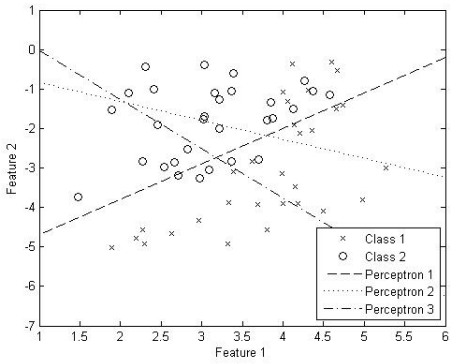
Table 4.3 summarizes the performance of the pool of classifiers generation methods



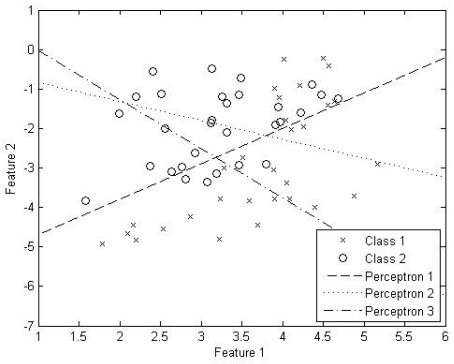
(a) Original test dataset



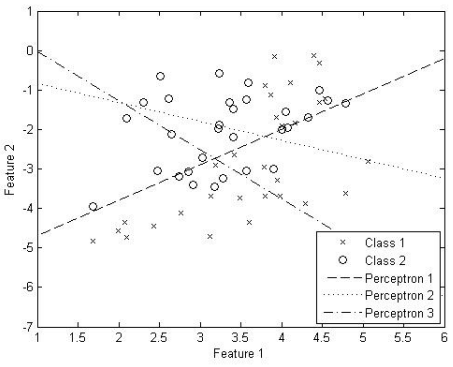
(b) Test 3 dataset



(c) Test 4 dataset



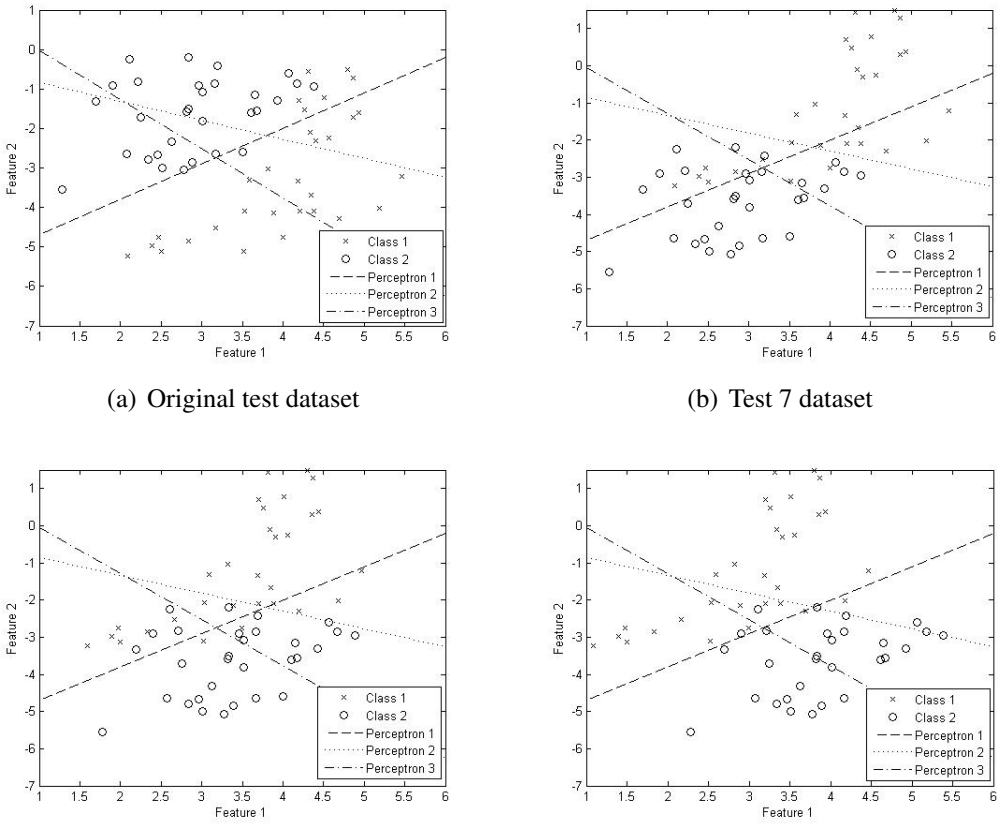
(d) Test 5 dataset



(e) Test 6 dataset

Figure 4.7: Disposal of generated pool over case scenario II datasets

for the P2 Problem. It can be observed that the accuracy yielded by the proposed method was significantly better than the other methods for the same pool size. Also, the accuracy of the proposed method with N_{PM} Perceptrons was approximately the same as the other methods when their pool size was set to N_{100}, which shows that the proposed method is able to perform as well as the other pool of classifiers generation methods with a much smaller pool size, for problems with complex boundaries in general.



(a) Original test dataset

(b) Test 7 dataset

(c) Test 8 dataset

(d) Test 9 dataset

Figure 4.8: Disposal of generated pool over case scenario III datasets

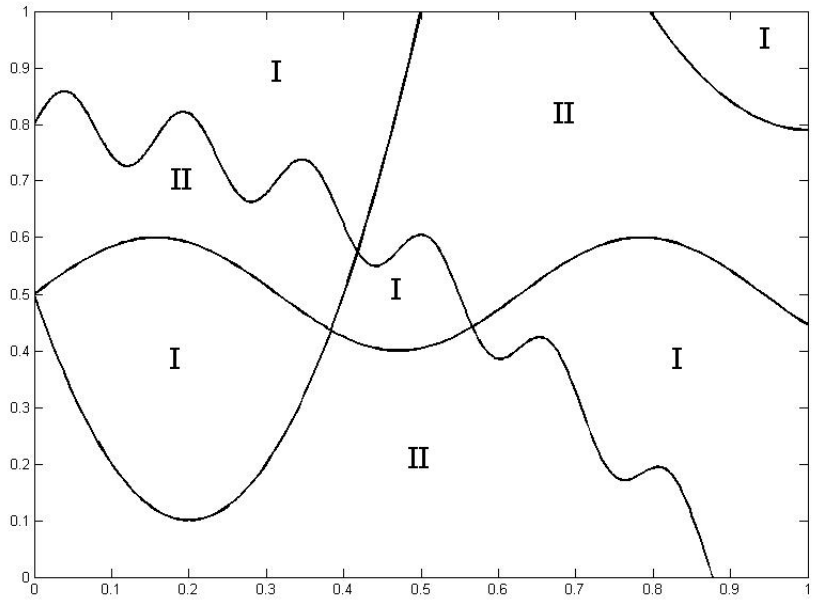


Figure 4.9: P2 Problem. I represents class 1, whereas II represents class 2.

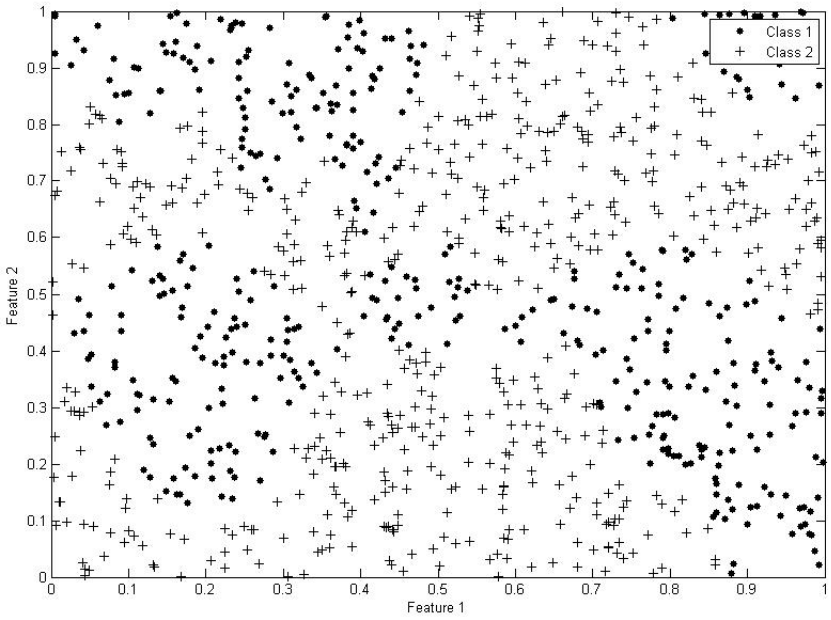


Figure 4.10: P2 Problem dataset.

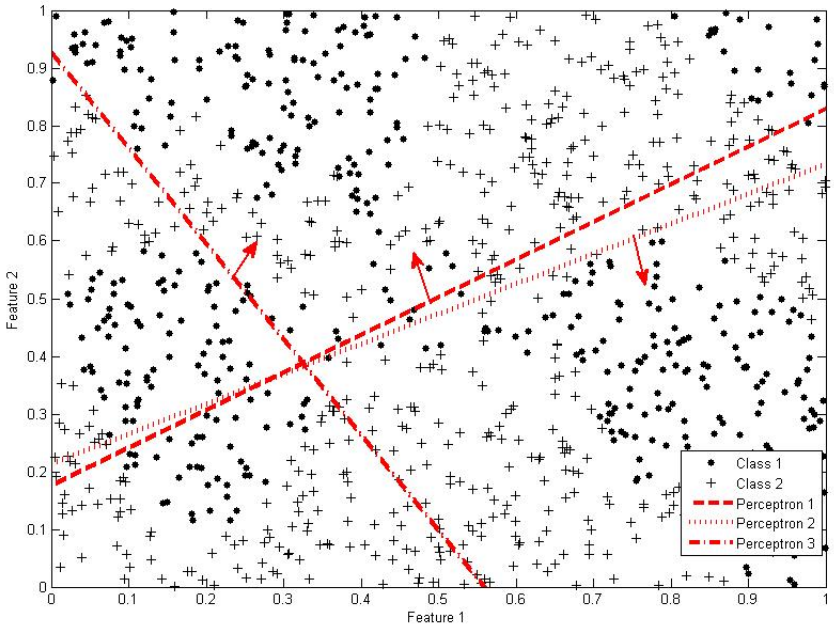


Figure 4.11: Example of a pool generated by the proposed method. The arrows show the areas that the Perceptrons label as Class 1.

4.1.3 Multi-class Problem

The purpose of this analysis is to investigate the performance of the proposed method in a multi-class problem. This is important because the Perceptrons in the pool generated by the proposed method are binary, that is, each one of them recognizes only two classes. Therefore, it

Table 4.2: Mean and standard deviation of pool of classifiers generation time, in seconds, for the P2 Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons.

No. of Perceptrons	Proposed Method	Bagging	Random Subspace	Boosting
N_{PM}	0.05 (0.01)	1.24 (0.14)	1.24 (0.16)	1.28 (0.16)
N_{100}	-	38.96 (0.62)	39.36 (0.67)	40.80 (0.88)

Table 4.3: Mean and standard deviation of accuracy for the pool of classifiers generation methods for the P2 Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

DCS Technique	Proposed Method	Bagging		Random Subspace		Boosting	
	N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
OLA	93.00 (1.49)	81.42 (10.88)	**93.40** (1.33)	84.56 (8.24)	92.96 (1.49)	83.12 (11.40)	93.08 (1.27)
LCA	93.22 (1.41)	81.58 (10.78)	93.46 (1.34)	84.52 (8.31)	93.14 (1.39)	83.28 (11.35)	**93.50** (1.46)
MCB	93.00 (1.49)	81.42 (10.88)	**93.40** (1.33)	84.56 (8.24)	92.96 (1.49)	83.12 (11.40)	93.08 (1.27)
A Priori	93.46 (1.27)	81.96 (11.03)	93.70 (1.26)	84.90 (8.14)	**93.96** (1.28)	83.68 (11.59)	93.54 (1.22)
A Posteriori	93.86 (1.04)	82.04 (10.97)	**93.96** (1.12)	84.90 (8.15)	93.24 (1.12)	83.74 (11.68)	93.94 (1.12)
Average	93.30	81.68	93.58	84.68	**93.85**	83.38	93.42

may or may not have an effect on the method's accuracy for non-binary problems.

In order to do that, the dataset used in this analysis was generated using the MATLAB PRTools toolbox [21] function **gendatm**, which creates an 8-class dataset. Figure 4.12 shows the multi-class problem dataset, which consists of 1000 instances. For this analysis, 20 replications of this dataset were produced. As in the previous analysis, 75% of the instances were randomly selected to form the training dataset and the remaining instances were used for testing for each replication. The DCS techniques used to evaluate the methods were also the same as the previous analysis, and for Bagging, Random Subspace and Boosting, the pool size was set to N_{PM} and N_{100} too. The Boosting representative algorithm used in this analysis was AdaBoost.

For the Multi-class Problem, the proposed method generated, in all iterations, $N_{PM} = 5$ Perceptrons. The amount of Perceptrons was, again, fairly small, considering that the Perceptrons generated by the proposed method are binary. This means the proposed method generated almost one Perceptron for every two classes. Figure 4.13 shows a pool generated by the proposed method using one of the replications. It can be observed that most Perceptrons fully divide two classes, since it is not difficult to find a pair of linearly separable classes in this problem.

The time necessary to generate each pool of classifiers by the four ensemble methods can

Table 4.4: Mean and standard deviation of pool of classifiers generation time, in seconds, for the Multi-class Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

No. of Perceptrons	Proposed Method	Bagging	Random Subspace	Boosting
N_{PM}	0.07 (0.01)	1.87 (0.19)	1.93 (0.35)	1.76 (0.34)
N_{100}	-	37.10 (2.58)	38.63 (5.54)	41.01 (3.48)

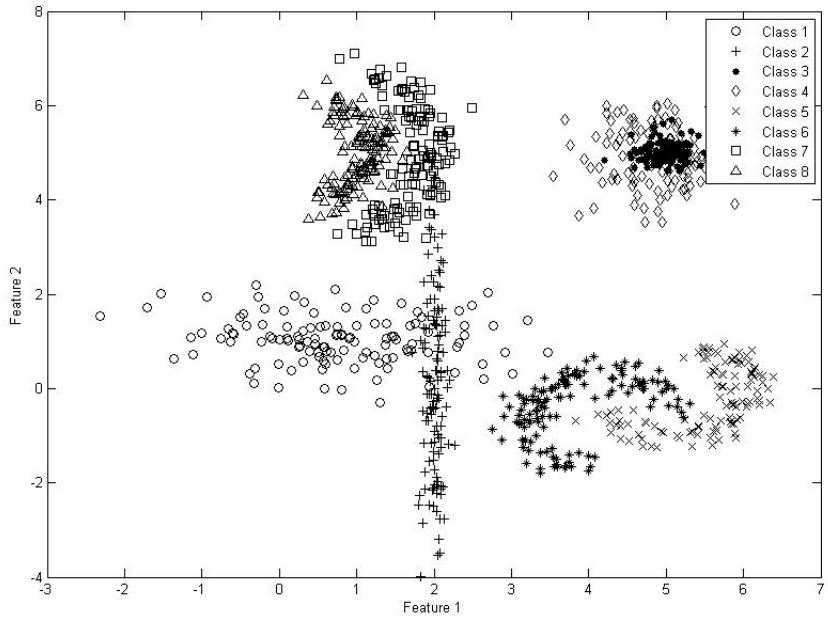


Figure 4.12: Multi-class Problem dataset.

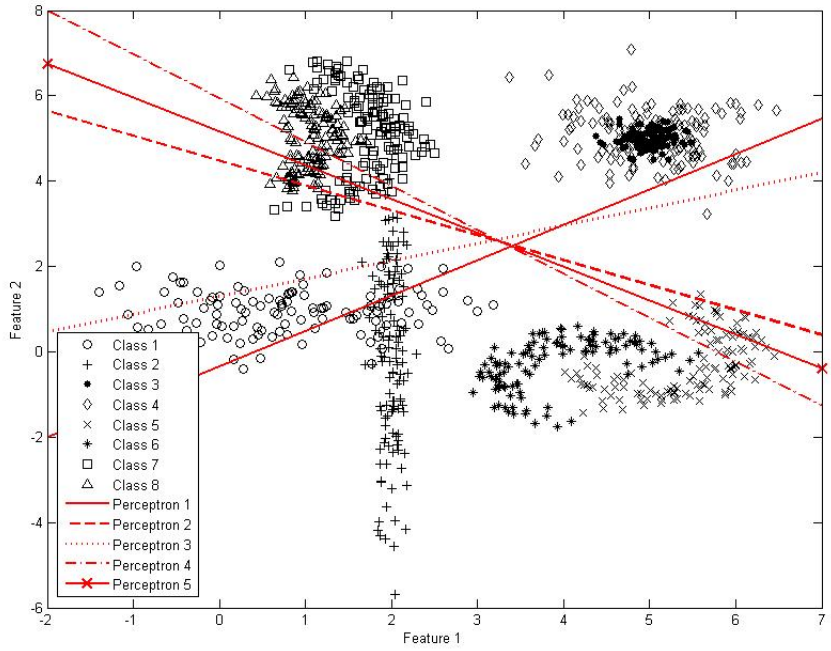


Figure 4.13: Example of a pool generated by the proposed method.

be found in Table 4.4. For the same pool size, the proposed method was more than ten times faster than the other three methods. For a pool size of N_{100} = 100 Perceptrons, the classical ensemble methods took nearly seventy times longer to generate the pool than the proposd method.

Table 4.5 shows the accuracy of the pool of classifiers generation methods for the multi-class problem. The proposed method obtained a very similar accuracy rate for all DCS techniques

Table 4.5: Mean and standard deviation of accuracy for the pool of classifiers generation methods for the Multi-class Problem. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

DCS Technique	Proposed Method	Bagging		Random Subspace		Boosting	
	N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
OLA	**91.36** (1.80)	88.24 (2.30)	89.72 (1.54)	86.30 (4.77)	88.98 (1.81)	90.76 (1.55)	90.88 (1.71)
LCA	**91.36** (1.79)	88.62 (2.33)	90.16 (1.54)	86.78 (4.65)	89.36 (1.83)	91.02 (1.76)	91.30 (1.87)
MCB	**91.36** (1.80)	88.24 (2.30)	89.72 (1.54)	86.30 (4.77)	88.98 (1.81)	90.78 (1.55)	90.88 (1.71)
A Priori	**91.36** (1.54)	88.20 (2.09)	89.42 (1.52)	86.40 (4.54)	88.92 (1.50)	90.90 (1.50)	91.02 (1.46)
A Posteriori	**91.36** (1.54)	88.56 (2.05)	90.16 (1.28)	86.56 (4.45)	89.46 (1.35)	91.14 (1.57)	91.28 (1.53)
Average	**91.36**	88.37	89.83	86.46	89.14	90.92	91.07

because, as previously mentioned, the proposed method generates nealy one Perceptron for every two classes. This means that each class is recognized by, on average, one classifier. Thus, the DS techniques do not have much choice when selecting a Perceptron to classify each instance. It can be observed that the accuracy yielded by the proposed method was approximately equivalent to Bagging and Boosting, and superior to Random Subspace, for the same pool size. Also, for Bagging and Boosting, the accuracy was not greatly improved by increasing to a hundred the size of the pool, so the accuracy of the proposed method with N_{PM} Perceptrons was still the same as these two methods when their pool size was set to N_{100}. The Random Subspace yielded a better performance with a pool size of N_{100}, but it was still similar to the proposed method's performance. Therefore, it can be concluded that, even though the proposed method only generates binary Perceptrons, it can still perform at least as well as the other methods with the same or a greater pool size for general multi-class problems.

4.2 Comparative study

In order to evaluate the performance of the proposed method, both in time and in accuracy, and compare it with Bagging, Random Subspace and Boosting, all four methods were tested with a total of 20 datasets. All of the them are public or synthetic datasets. Ten come from the UCI machine learning repository [15], four from the Ludmila Kuncheva Collection [17] of real medical data, three from the STATLOG project [23], two from the Knowledge Extraction based on Evolutionary Learning (KEEL) repository [13] and one artificial datasets generated with the Matlab PRTOOLS toolbox [21]. The main characteristics of each dataset are shown in Table 4.6.

For this experiment, 20 replications of each dataset were used. Each replication was generated by randomly splitting the dataset in two parts: 70% for training and 30% for testing. As in previous analysis, the Boosting algorithm used in this comparative study was AdaBoost. Also, the DS techniques used to evaluate the performance of the four pool of classifiers generation methods were OLA, LCA, MCB, A Priori and A Posteriori, for the same reason stated in the previous section.

The proposed method was evaluated after training by using the training dataset as the test dataset for the DCS techniques previously mentioned, in order to measure the number of times

Table 4.6: Main characteristics of the datasets used in the experiment

Dataset	No. of Instances	No. of Features	No. of Classes	Source
Adult	48842	14	2	UCI
Blood Transfusion	748	4	2	UCI
Breast (WDBC)	568	30	2	UCI
German credit	1000	20	2	STATLOG
Heart	270	13	2	STATLOG
ILPD	214	9	6	UCI
Ionosphere	315	34	2	UCI
Laryngeal1	213	16	2	LKC
Laryngeal3	353	16	3	LKC
Lithuanian	1000	2	2	PRTOOLS
Liver Disorders	345	6	2	UCI
Mammographic	961	5	2	KEEL
Monk2	4322	6	2	KEEL
Pima	768	8	2	UCI
Sonar	208	60	2	UCI
Thyroid	215	5	3	LKC
Vehicle	846	18	4	STATLOG
Vertebral Column	310	6	2	UCI
Weaning	302	17	2	LKC
Wine	178	13	3	UCI

the DCS techniques chose the right Perceptron to classify each test instance. Table 4.7 shows the rate at which OLA, LCA, MCB, A Priori and A Posteriori methods select the right classifier, and therefore correctly classifies the instances. It can be observed that, on average, all five DCS techniques choose the right Perceptron most of the time, for all datasets, which suggests the pool generated by the proposed method is fairly accurate. For some datasets, such as Breast, Lithuanian, Thyroid and Wine, the accuracy was reasonably close to the oracle's performance (100%), for some DCS methods.

Table 4.8 shows a summary of the pool generation for each method. N_{PM} is the average number of Perceptrons generated by the proposed method. Bagging, Random Subspace and Boosting were tested with a pool size equal to the proposed method's pool size, N_{PM}, as well as with $N_{100} = 100$ Perceptrons. It can be noticed that the number of Perceptrons generated by the proposed method is generally very low, which implies there is no need to produce a large number of classifiers to guarantee the best classifier is included in the pool. Also, the proposed method generates the same amount of classifiers in one order of magnitude less time than the other ensemble methods.

The average accuracy yielded by the ensemble methods using OLA, LCA, MCB, A Priori and A Posteriori techniques is shown in Table 4.9 for all datasets from Table 4.6. The average pool size generated by the proposed method was N_{PM}, and all ensemble methods were tested with this amount of Perceptrons. Bagging, Random Subspace and Boosting were also tested

Table 4.7: Mean and standard deviation of the rate at which the right Perceptron is chosen by the DCS technique, using the training dataset for testing, for the datasets from Table 4.6. Best results are in bold.

Dataset	OLA	LCA	MCB	A Priori	A Posteriori
Adult	86.90 (0.86)	86.76 (0.92)	87.13 (0.72)	**89.52** (1.55)	89.29 (1.56)
Blood	79.58 (0.51)	**80.19** (0.35)	79.60 (0.51)	69.54 (0.94)	69.37 (0.95)
Breast	95.29 (0.56)	80.19 (0.36)	**95.31** (0.56)	95.25 (0.93)	95.08 (0.67)
German	71.05 (1.43)	**75.74** (1.34)	71.22 (1.47)	72.82 (1.19)	72.10 (1.42)
Heart	84.06 (1.92)	83.86 (2.39)	83.96 (1.71)	84.38 (1.77)	**84.55** (1.61)
ILPD	70.36 (2.54)	**72.49** (2.56)	70.36 (2.54)	62.57 (2.43)	64.69 (1.80)
Ionosphere	86.46 (1.48)	**87.33** (1.53)	86.42 (1.42)	84.23 (2.31)	84.82 (2.37)
Laryngeal1	84.75 (2.07)	**84.81** (2.38)	84.75 (1.92)	81.37 (2.52)	81.34 (2.70)
Laryngeal3	74.81 (2.94)	73.97 (1.98)	**74.84** (2.90)	42.46 (6.43)	42.72 (5.89)
Lithuanian	93.60 (1.34)	**96.35** (1.26)	93.60 (1.34)	86.63 (1.85)	86.67 (2.13)
Liver	67.22 (1.39)	**70.61** (2.91)	67.33 (1.27)	58.37 (4.67)	59.98 (4.95)
Mammographic	82.71 (0.64)	**82.82** (1.54)	82.68 (0.73)	81.53 (0.74)	81.85 (1.15)
Monk2	85.77 (3.59)	**91.82** (3.61)	86.66 (4.48)	76.54 (1.74)	77.36 (1.30)
Pima	75.64 (1.55)	76.02 (1.66)	75.81 (1.83)	76.13 (1.67)	**76.39** (2.14)
Sonar	80.00 (3.62)	**83.46** (3.45)	80.19 (3.63)	70.60 (3.28)	70.35 (3.71)
Thyroid	90.82 (1.09)	**95.81** (0.91)	90.84 (1.10)	88.51 (1.32)	88.87 (1.67)
Vehicle	76.13 (1.49)	**77.98** (1.56)	76.19 (1.50)	40.02 (1.87)	39.88 (1.76)
Vertebral	82.39 (2.14)	**84.32** (2.32)	82.39 (2.18)	75.44 (4.72)	76.06 (5.52)
Weaning	83.45 (1.33)	**84.32** (1.72)	83.36 (1.20)	79.00 (2.55)	78.25 (1.95)
Wine	97.14 (0.96)	**97.74** (1.14)	97.14 (0.96)	66.99 (3.09)	66.95 (3.28)
Average	82.40	**84.18**	82.49	74.09	74.33

Table 4.8: Mean and standard deviation of pool of classifiers generation time, in seconds, for the datasets from Table 4.6. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

Dataset	N_{PM}	Proposed Method	Bagging		Random Subspace		Boosting	
		N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Adult	3.10 (0.31)	**0.09** (0.01)	1.26 (0.13)	40.00 (1.50)	1.31 (0.24)	43.53 (10.79)	1.76 (0.25)	83.95 (1.93)
Blood	3.00 (0.00)	**0.10** (0.02)	2.03 (0.62)	65.85 (8.23)	1.94 (0.12)	64.69 (4.02)	2.23 (0.17)	93.41 (5.58)
Breast	3.00 (0.00)	**0.10** (0.12)	0.42 (0.09)	13.51 (2.21)	0.45 (0.07)	15.07 (2.49)	0.60 (0.16)	50.58 (17.12)
German	3.10 (0.30)	**0.13** (0.13)	2.41 (0.32)	78.44 (1.44)	2.51 (0.27)	81.37 (1.66)	2.82 (0.37)	116.88 (1.70)
Heart	3.20 (0.41)	**0.09** (0.09)	0.70 (0.24)	18.70 (1.48)	0.64 (0.10)	19.94 (1.32)	0.81 (0.17)	32.98 (1.90)
ILPD	3.80 (0.41)	**0.13** (0.11)	2.12 (0.37)	56.08 (12.22)	2.18 (0.34)	57.16 (10.35)	2.30 (0.35)	75.83 (15.39)
Ionosphere	3.70 (0.47)	**0.11** (0.12)	0.59 (0.14)	15.03 (2.71)	0.78 (0.17)	21.23 (2.98)	0.89 (0.20)	42.50 (5.15)
Laryngeal1	2.40 (0.68)	**0.05** (0.02)	0.41 (0.20)	17.48 (5.60)	0.46 (0.23)	19.14 (6.24)	0.52 (0.33)	32.46 (11.10)
Laryngeal3	4.80 (1.10)	**0.15** (0.14)	1.80 (0.64)	36.61 (7.35)	1.85 (0.69)	38.65 (7.98)	2.06 (0.79)	50.08 (10.76)
Lithuanian	3.60 (0.50)	**0.09** (0.01)	1.61 (0.41)	44.93 (6.68)	1.74 (0.70)	47.59 (14.40)	2.21 (0.45)	82.33 (15.25)
Liver	3.20 (0.41)	**0.07** (0.00)	1.18 (0.12)	37.06 (1.45)	1.21 (0.14)	38.01 (1.47)	1.25 (0.15)	43.40 (1.47)
Mammographic	2.90 (0.31)	**0.09** (0.02)	1.86 (0.69)	59.18 (5.31)	1.69 (0.20)	58.41 (2.20)	2.20 (0.32)	100.86 (2.57)
Monk2	2.50 (0.51)	**0.07** (0.02)	0.89 (0.39)	33.54 (8.28)	0.84 (0.32)	33.45 (8.18)	0.99 (0.42)	57.71 (14.08)
Pima	3.50 (0.51)	**0.13** (0.11)	2.25 (0.84)	63.33 (16.3)	2.32 (0.88)	63.54 (11.28)	2.66 (0.96)	95.72 (25.14)
Sonar	3.30 (0.66)	**0.11** (0.14)	0.29 (0.09)	8.679 (0.97)	0.71 (0.14)	21.77 (0.96)	0.34 (0.11)	14.02 (2.19)
Thyroid	3.80 (0.41)	**0.09** (0.00)	0.80 (0.10)	21.04 (0.59)	0.83 (0.10)	21.93 (0.63)	1.53 (0.37)	80.90 (0.95)
Vehicle	5.60 (0.50)	**0.18** (0.01)	4.33 (0.40)	77.47 (0.67)	4.43 (0.43)	79.26 (0.76)	5.36 (0.62)	118.06 (1.72)
Vertebral	2.50 (0.69)	**0.09** (0.14)	0.51 (0.13)	20.31 (0.63)	0.52 (0.13)	20.98 (0.64)	0.64 (0.22)	38.78 (0.69)
Weaning	3.00 (0.00)	**0.10** (0.13)	0.65 (0.08)	20.83 (0.95)	0.68 (0.05)	22.71 (1.20)	0.86 (0.15)	39.86 (0.88)
Wine	2.90 (0.31)	**0.09** (0.12)	0.13 (0.03)	4.284 (0.21)	0.13 (0.02)	4.60 (0.50)	0.14 (0.01)	3.92 (2.10)
Average	3.34	**0.10**	1.31	36.62	1.36	38.65	1.61	62.70

with $N_{100} = 100$ classifiers. The results from Table 4.9 show that, on average, the performance of the proposed method was equivalent to the performance of Bagging, Random Subspace and Boosting, for all DCS techniques.

Figure 4.14(a) summarizes the comparison between the results from Table 4.9, in terms of the mean values of the accuracy obtained for all DCS techniques by each ensemble method with the same pool size. It can be noticed that, for A Priori, the number of times the proposed method won and lost was almost equal. For the other DCS methods, the proposed method yielded an inferior mean accuracy than the other ensemble methods more frequently. Overall, the proposed method achieved a better performance, on average, in less occasions than the other three pool of classifiers generation methods with the same number of Perceptrons, as the last bar in Figure 4.14(a) shows.

The summary of the results from Table 4.9 for ensemble methods with a pool size of N_{100} in comparison with the proposed method is shown in Figure 4.14(b). The proposed method won and lost, on average, the same amount of times for the A Priori, and almost equally for OLA and MCB. For the other two ensemble methods, the performance of the proposed method was more frequently worse than the other pool of classifiers generation methods. The last bar in Figure 4.14(b) shows that, on average, the performance of the proposed method was better than the other methods with a pool of one hundred classifiers nearly half of the time.

The decision of using only DCS techniques instead of DES techniques to evaluate the performance of the ensemble methods may be the reason for the classical methods with a pool size of N_{PM} classifiers to perform better than with N_{100} classifiers more frequently, since smaller pools favours the selection of only one classifier. Moreover, since the results from Table 4.9 show a very close proximity in the accuracy rates of all methods, it can be concluded that the proposed method yields approximately the same accuracy while requiring much less processing time than the other pool of classifiers generation methods.

Table 4.9: Mean and standard deviation of accuracy for the pool of classifiers generation methods using DCS techniques for the datasets from Table 4.6. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

(a) OLA

Dataset	N_{PM}	Proposed Method	Bagging		Random Subspace		Boosting	
		N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Adult	3.10 (0.31)	**88.15** (2.93)	85.89 (2.40)	84.91 (2.79)	86.12 (2.85)	84.79 (2.43)	85.89 (2.94)	84.33 (2.46)
Blood	3.00 (0.00)	75.53 (1.14)	74.86 (3.06)	74.60 (2.24)	75.10 (3.83)	75.21 (1.81)	**75.66** (1.77)	74.44 (2.13)
Breast	3.00 (0.00)	**96.97** (1.16)	96.23 (1.14)	96.37 (1.00)	96.16 (0.53)	96.58 (0.61)	95.91 (1.03)	96.23 (1.21)
German	3.10 (0.30)	70.04 (2.35)	71.74 (2.10)	**72.70** (2.12)	71.34 (2.99)	71.40 (2.24)	71.54 (2.10)	69.62 (2.03)
Heart	3.20 (0.41)	**86.61** (2.18)	85.58 (4.09)	83.45 (3.26)	83.38 (3.99)	82.57 (2.25)	83.01 (3.86)	80.80 (3.91)
ILPD	3.80 (0.41)	64.72 (0.95)	**67.05** (3.10)	65.51 (2.75)	66.64 (2.62)	66.02 (2.95)	65.99 (3.18)	65.37 (2.77)
Ionosphere	3.70 (0.47)	87.15 (2.76)	86.98 (2.06)	87.61 (2.60)	87.21 (3.31)	**87.78** (3.25)	87.10 (2.88)	86.81 (2.84)
Laryngeal1	2.40 (0.68)	80.37 (4.25)	80.56 (5.54)	78.49 (4.79)	**81.69** (5.34)	80.66 (5.22)	80.18 (6.25)	78.96 (4.79)
Laryngeal3	4.80 (1.10)	72.24 (1.71)	70.56 (4.93)	69.94 (5.35)	71.01 (7.13)	**73.14** (3.37)	70.67 (4.09)	68.65 (4.59)
Lithuanian	3.60 (0.50)	**96.26** (1.44)	92.36 (2.89)	96.16 (1.96)	91.66 (5.61)	96.13 (1.83)	93.70 (2.74)	96.13 (1.11)
Liver	3.20 (0.41)	58.37 (3.53)	**65.93** (3.83)	65.00 (2.84)	62.26 (4.55)	64.88 (2.43)	65.23 (3.90)	63.66 (3.75)
Mammographic	2.90 (0.31)	82.59 (2.47)	82.54 (5.26)	82.35 (2.29)	82.95 (2.14)	82.40 (2.53)	**83.19** (2.14)	82.54 (2.13)
Monk2	2.50 (0.51)	86.20 (3.73)	86.66 (5.18)	**94.58** (1.53)	86.80 (5.52)	93.84 (2.37)	88.65 (3.46)	92.91 (2.44)
Pima	3.50 (0.51)	72.29 (2.38)	73.12 (3.04)	**73.77** (2.02)	72.65 (2.92)	73.38 (2.47)	73.75 (2.91)	71.40 (2.66)
Sonar	3.30 (0.66)	80.00 (3.32)	81.44 (3.59)	79.80 (4.20)	79.13 (5.62)	**82.01** (4.24)	80.67 (3.92)	81.15 (4.48)
Thyroid	3.80 (0.41)	95.83 (1.32)	96.18 (1.35)	**97.02** (0.86)	95.95 (2.53)	96.93 (1.42)	96.12 (1.17)	96.56 (0.88)
Vehicle	5.60 (0.50)	70.09 (2.57)	72.14 (1.89)	71.79 (2.27)	71.76 (2.20)	72.16 (2.47)	**72.61** (1.88)	**72.61** (2.08)
Vertebral	2.50 (0.69)	81.41 (2.05)	83.58 (4.45)	85.44 (3.36)	85.06 (3.28)	**85.76** (3.32)	85.64 (3.04)	83.52 (2.97)
Weaning	3.00 (0.00)	78.68 (3.71)	78.88 (3.75)	79.53 (3.37)	**80.59** (3.66)	80.19 (3.40)	79.67 (4.14)	76.77 (3.82)
Wine	2.90 (0.31)	**98.44** (2.05)	95.33 (4.37)	96.33 (3.55)	95.44 (3.91)	96.55 (3.18)	97.11 (1.92)	96.44 (2.63)
Average	3.34	81.10	81.38	**81.77**	81.14	82.12	81.61	80.94

(b) LCA

Dataset	N_{PM}	Proposed Method	Bagging		Random Subspace		Boosting	
		N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Adult	3.10 (0.31)	**87.39** (2.81)	86.79 (2.68)	85.72 (2.00)	86.96 (2.48)	85.37 (2.04)	86.44 (2.20)	85.52 (2.54)
Blood	3.00 (0.00)	75.74 (1.04)	75.15 (3.11)	75.37 (1.97)	75.29 (3.63)	75.47 (1.77)	**76.06** (1.45)	75.39 (1.83)
Breast	3.00 (0.00)	**97.18** (1.16)	96.58 (1.00)	96.33 (0.93)	96.61 (0.67)	97.07 (0.86)	96.47 (1.18)	96.47 (0.85)
German	3.10 (0.30)	70.84 (1.86)	**72.70** (2.64)	72.24 (2.16)	72.66 (3.20)	72.32 (2.31)	72.04 (2.09)	70.12 (2.02)
Heart	3.20 (0.41)	86.47 (2.84)	**86.98** (3.38)	82.79 (4.27)	84.85 (3.27)	83.08 (2.21)	85.22 (2.40)	82.72 (3.56)
ILPD	3.80 (0.41)	64.79 (1.51)	**67.50** (2.79)	66.36 (2.65)	66.74 (2.69)	66.78 (2.87)	66.43 (3.02)	66.71 (1.95)
Ionosphere	3.70 (0.47)	87.27 (3.20)	88.57 (2.00)	88.75 (1.94)	88.75 (2.87)	88.63 (1.91)	**88.97** (2.24)	88.23 (1.92)
Laryngeal1	2.40 (0.68)	80.94 (4.69)	81.03 (4.51)	79.71 (4.81)	**81.79** (4.90)	80.47 (5.38)	81.69 (5.68)	79.90 (5.02)
Laryngeal3	4.80 (1.10)	72.58 (2.13)	72.24 (3.09)	68.53 (4.00)	72.86 (6.64)	**72.92** (3.80)	71.62 (2.86)	69.66 (3.21)
Lithuanian	3.60 (0.50)	**96.73** (1.54)	92.46 (2.97)	96.43 (1.91)	92.03 (5.57)	96.46 (1.73)	93.93 (2.75)	96.33 (1.59)
Liver	3.20 (0.41)	58.37 (2.81)	64.88 (4.17)	**65.34** (2.86)	61.74 (4.47)	63.31 (2.42)	63.89 (2.37)	64.01 (4.27)
Mammographic	2.90 (0.31)	82.54 (2.41)	82.76 (5.14)	82.30 (2.34)	**83.34** (2.87)	82.33 (2.38)	83.17 (2.41)	82.69 (2.15)
Monk2	2.50 (0.51)	90.27 (2.17)	88.42 (5.22)	**95.13** (1.46)	87.73 (5.79)	95.00 (1.05)	90.87 (2.90)	93.33 (1.93)
Pima	3.50 (0.51)	73.22 (3.38)	73.56 (2.72)	73.51 (2.00)	72.91 (2.90)	73.46 (2.10)	**73.98** (2.82)	72.18 (2.21)
Sonar	3.30 (0.66)	78.07 (5.00)	**83.75** (3.66)	83.36 (3.80)	80.28 (5.06)	82.69 (3.52)	83.36 (2.73)	82.78 (3.71)
Thyroid	3.80 (0.41)	96.82 (1.30)	97.13 (1.00)	96.90 (1.01)	96.50 (1.83)	**97.54** (1.21)	97.02 (1.09)	96.87 (0.92)
Vehicle	5.60 (0.50)	70.75 (2.21)	72.94 (1.44)	73.70 (1.54)	72.26 (1.43)	72.59 (1.44)	73.18 (1.64)	**73.82** (1.82)
Vertebral	2.50 (0.69)	82.30 (1.93)	84.67 (4.98)	84.87 (4.31)	85.32 (2.98)	85.96 (3.12)	**86.41** (2.98)	83.71 (3.05)
Weaning	3.00 (0.00)	78.81 (3.04)	80.78 (3.87)	81.11 (4.31)	**82.30** (3.78)	80.65 (3.91)	80.06 (4.56)	79.73 (3.00)
Wine	2.90 (0.31)	**98.44** (2.05)	96.55 (4.10)	97.11 (3.31)	96.22 (3.68)	97.66 (3.18)	97.55 (1.89)	97.55 (2.78)
Average	3.34	81.47	82.27	82.27	81.85	**82.49**	82.42	81.89

Table 4.9: Mean and standard deviation of accuracy for the pool of classifiers generation methods using DCS techniques for the datasets from Table 4.6. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

(c) MCB

Dataset	N_{PM}	Proposed Method	Bagging		Random Subspace		Boosting	
		N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Adult	3.10 (0.31)	**88.15** (2.93)	85.98 (2.36)	84.91 (2.79)	86.06 (2.70)	84.79 (2.43)	86.06 (2.86)	84.36 (2.50)
Blood	3.00 (0.00)	75.53 (1.14)	74.86 (3.06)	74.60 (2.24)	75.10 (3.83)	75.21 (1.81)	**75.66** (1.77)	74.44 (2.13)
Breast	3.00 (0.00)	**97.04** (1.28)	96.19 (1.19)	96.37 (1.00)	96.16 (0.53)	96.58 (0.61)	95.95 (1.06)	96.23 (1.21)
German	3.10 (0.30)	70.52 (2.08)	71.70 (2.25)	**72.62** (2.08)	71.50 (3.18)	71.36 (2.31)	71.62 (2.09)	69.60 (2.03)
Heart	3.20 (0.41)	**86.17** (2.35)	85.66 (3.90)	83.45 (3.26)	83.45 (4.18)	82.50 (2.38)	83.38 (3.99)	80.80 (3.91)
ILPD	3.80 (0.41)	64.72 (0.95)	**67.08** (3.07)	65.51 (2.75)	66.57 (2.72)	65.99 (2.99)	65.95 (3.22)	65.37 (2.77)
Ionosphere	3.70 (0.47)	87.15 (2.71)	87.10 (2.15)	**87.67** (2.64)	87.55 (3.36)	87.61 (3.37)	87.27 (2.94)	86.81 (2.84)
Laryngeal1	2.40 (0.68)	80.56 (4.58)	**80.84** (5.41)	78.49 (4.79)	81.69 (5.34)	80.66 (5.22)	80.37 (6.43)	78.96 (4.79)
Laryngeal3	4.80 (1.10)	71.79 (1.58)	70.67 (4.77)	69.94 (5.35)	71.06 (7.18)	**73.37** (3.28)	70.67 (4.18)	68.65 (4.59)
Lithuanian	3.60 (0.50)	96.26 (1.44)	92.36 (2.89)	96.16 (1.96)	**91.66** (5.61)	96.13 (1.83)	93.70 (2.74)	96.13 (1.11)
Liver	3.20 (0.41)	58.37 (3.49)	**65.98** (3.93)	65.00 (2.84)	62.61 (4.71)	65.69 (2.24)	65.23 (4.14)	63.66 (3.75)
Mammographic	2.90 (0.31)	82.59 (2.47)	82.54 (5.28)	82.35 (2.29)	83.00 (2.12)	82.40 (2.53)	**83.19** (2.14)	82.54 (2.13)
Monk2	2.50 (0.51)	87.96 (3.79)	86.75 (5.16)	**94.58** (1.53)	86.85 (5.55)	93.79 (2.36)	88.70 (3.49)	92.91 (2.44)
Pima	3.50 (0.51)	72.70 (2.67)	73.15 (3.04)	**73.77** (2.02)	72.55 (3.16)	73.30 (2.52)	**73.77** (2.92)	71.35 (2.65)
Sonar	3.30 (0.66)	79.80 (3.08)	81.53 (3.55)	79.71 (4.29)	79.03 (5.68)	**82.01** (4.24)	80.86 (4.20)	81.15 (4.48)
Thyroid	3.80 (0.41)	95.95 (1.32)	96.18 (1.35)	**97.02** (0.86)	95.95 (2.53)	96.93 (1.42)	96.12 (1.17)	96.56 (0.88)
Vehicle	5.60 (0.50)	70.14 (2.51)	72.24 (1.71)	71.74 (2.24)	72.07 (2.59)	72.28 (2.65)	72.64 (1.81)	**72.66** (2.15)
Vertebral	2.50 (0.69)	82.69 (2.22)	83.58 (4.45)	85.44 (3.36)	85.06 (3.28)	**85.76** (3.32)	85.70 (2.97)	83.52 (2.97)
Weaning	3.00 (0.00)	79.21 (3.29)	79.01 (3.78)	79.53 (3.37)	**80.85** (3.75)	80.32 (3.48)	79.86 (3.96)	76.77 (3.82)
Wine	2.90 (0.31)	**98.44** (2.05)	95.33 (4.37)	96.33 (3.55)	95.44 (3.91)	96.55 (3.18)	97.11 (1.92)	96.44 (2.63)
Average	3.34	81.28	81.43	81.76	81.21	**82.16**	81.69	80.94

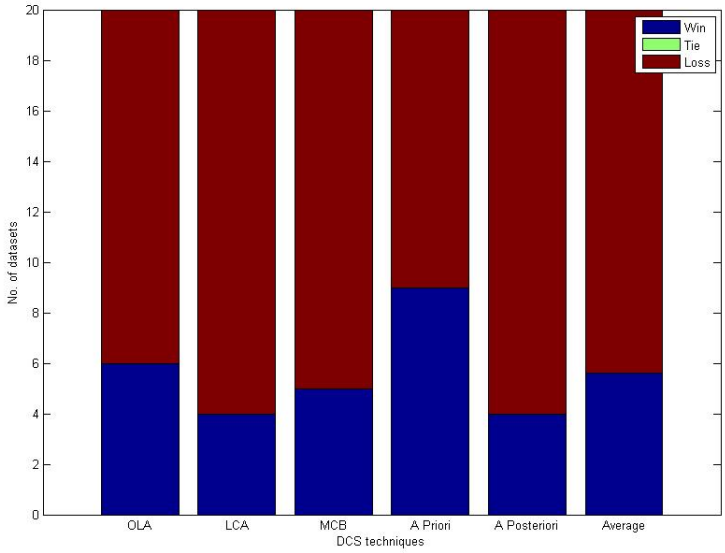
(d) A Priori

Dataset	N_{PM}	Proposed Method	Bagging		Random Subspace		Boosting	
		N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Adult	3.10 (0.31)	**86.99** (2.46)	85.80 (2.62)	84.36 (3.13)	85.95 (2.61)	83.98 (1.84)	85.34 (2.49)	82.28 (2.36)
Blood	3.00 (0.00)	**73.53** (1.55)	72.18 (4.88)	71.59 (4.50	70.74 (6.30)	71.94 (2.98)	72.50 (4.16)	70.00 (5.34)
Breast	3.00 (0.00)	**97.04** (1.10)	96.23 (1.14)	96.37 (1.00)	96.16 (0.53)	96.58 (0.61)	95.84 (1.06)	96.23 (1.21)
German	3.10 (0.30)	69.80 (1.24)	71.62 (2.32)	71.74 (2.13)	**72.14** (3.55)	71.46 (2.31)	71.20 (2.21)	68.50 (1.78)
Heart	3.20 (0.41)	**87.20** (2.61)	85.00 (3.77)	82.79 (3.30)	83.16 (3.67)	81.32 (3.09)	82.72 (3.69)	80.44 (4.16)
ILPD	3.80 (0.41)	64.72 (2.01)	**67.43** (2.84)	65.13 (3.21)	67.15 (2.68)	66.43 (3.19)	67.12 (3.63)	65.68 (3.75)
Ionosphere	3.70 (0.47)	86.93 (3.00)	86.98 (2.06)	87.61 (2.60)	87.55 (3.24)	**87.78** (3.39)	87.44 (2.98)	86.87 (2.89)
Laryngeal1	2.40 (0.68)	81.32 (4.45)	81.50 (5.49)	78.11 (5.13)	**82.16** (5.31)	80.94 (5.00)	80.66 (6.23)	78.96 (4.75)
Laryngeal3	4.80 (1.10)	71.79 (3.02)	70.78 (4.62)	68.76 (5.88)	71.06 (7.35)	**72.07** (4.45)	70.44 (3.82)	68.48 (4.42)
Lithuanian	3.60 (0.50)	**96.40** (1.89)	92.26 (2.96)	96.13 (2.14)	91.60 (5.44)	96.20 (2.24)	93.96 (3.00)	95.90 (1.33)
Liver	3.20 (0.41)	61.39 (3.87)	**66.39** (3.45)	64.76 (3.60)	63.08 (4.85)	66.04 (1.71)	65.29 (3.35)	62.90 (4.02)
Mammographic	2.90 (0.31)	**81.58** (2.83)	76.99 (11.25)	80.72 (2.97)	77.73 (10.81)	79.92 (3.06)	77.92 (8.78)	80.31 (2.69)
Monk2	2.50 (0.51)	86.66 (3.96)	86.20 (5.10)	**94.53** (1.61)	86.06 (4.92)	93.61 (2.42)	88.14 (3.44)	92.45 (2.88)
Pima	3.50 (0.51)	72.50 (2.12)	72.96 (3.17)	**73.69** (2.04)	72.81 (2.85)	73.33 (2.00)	73.41 (3.08)	70.83 (2.20)
Sonar	3.30 (0.66)	**82.11** (2.34)	81.34 (3.58)	79.80 (4.20)	79.80 (5.34)	81.82 (4.16)	80.86 (3.71)	81.15 (4.48)
Thyroid	3.80 (0.41)	96.06 (1.26)	96.50 (1.42)	96.61 (0.75)	96.38 (2.49)	**97.13** (1.17)	96.21 (1.23)	96.50 (0.92)
Vehicle	5.60 (0.50)	69.24 (2.00)	71.48 (1.90)	71.39 (2.39)	71.22 (1.98)	71.17 (1.98)	71.43 (1.97)	**72.00** (2.65)
Vertebral	2.50 (0.69)	80.76 (1.55)	83.91 (4.27)	84.61 (3.37)	85.70 (3.33)	**86.53** (3.51)	85.51 (3.19)	83.07 (3.36)
Weaning	3.00 (0.00)	78.02 (3.47)	78.81 (3.86)	79.67 (3.37)	**80.78** (3.77)	80.19 (3.65)	79.34 (4.11)	76.77 (3.89)
Wine	2.90 (0.31)	**98.44** (2.05)	95.55 (4.50)	96.33 (3.55)	95.44 (3.91)	96.55 (3.18)	96.88 (1.96)	96.44 (2.63)
Average	3.34	81.12	80.99	81.23	80.83	**81.75**	81.11	80.28

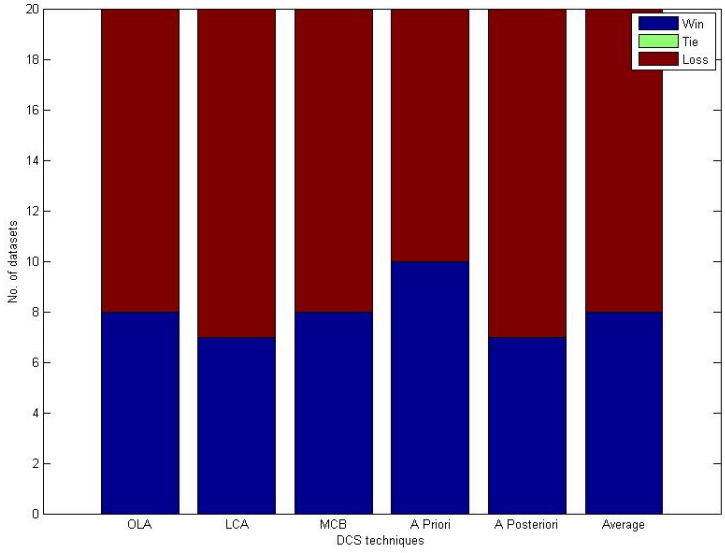
Table 4.9: Mean and standard deviation of accuracy for the pool of classifiers generation methods using DCS techniques for the datasets from Table 4.6. N_{PM} is the proposed method's pool size, while N_{100} is the pool size of 100 Perceptrons. Best results are in bold.

(e) A Posteriori

Dataset	N_{PM}	Proposed Method	Bagging		Random Subspace		Boosting	
		N_{PM}	N_{PM}	N_{100}	N_{PM}	N_{100}	N_{PM}	N_{100}
Adult	3.10 (0.31)	86.47 (2.53)	86.70 (1.94)	85.80 (2.25)	**86.82** (2.77)	85.78 (2.23)	86.73 (2.43)	85.78 (2.23)
Blood	3.00 (0.00)	**73.75** (1.63)	72.52 (4.84)	73.24 (4.20)	71.01 (6.30)	73.13 (3.63)	72.60 (4.26)	71.25 (4.55)
Breast	3.00 (0.00)	97.32 (1.01)	97.18 (1.07)	97.11 (1.42)	96.83 (0.87)	**97.92** (0.95)	97.71 (1.04)	97.25 (1.18)
German	3.10 (0.30)	71.32 (1.89)	72.86 (2.54)	71.82 (2.94)	**73.18** (3.58)	72.96 (2.68)	72.06 (1.76)	71.92 (3.30)
Heart	3.20 (0.41)	86.32 (4.27)	**87.57** (2.99)	86.17 (3.51)	85.66 (3.53)	86.25 (3.44)	86.61 (2.97)	86.17 (3.51)
ILPD	3.80 (0.41)	65.89 (2.31)	68.49 (2.87)	66.91 (2.39)	68.32 (2.59)	**68.76** (2.61)	67.53 (3.44)	66.26 (2.73)
Ionosphere	3.70 (0.47)	86.47 (3.59)	**89.65** (1.80)	87.78 (2.49)	89.31 (2.59)	87.15 (3.27)	89.43 (1.84)	86.64 (3.51)
Laryngeal1	2.40 (0.68)	82.26 (4.51)	82.35 (4.75)	81.98 (4.47)	**82.73** (5.20)	81.79 (4.29)	81.98 (6.04)	82.07 (4.60)
Laryngeal3	4.80 (1.10)	72.47 (1.65)	73.98 (2.13)	73.31 (1.45)	**75.00** (1.95)	73.37 (6.79)	73.31 (1.58)	72.58 (1.38)
Lithuanian	3.20 (0.41)	**96.80** (2.07)	92.46 (2.98)	96.73 (2.19)	92.16 (5.49)	96.63 (2.15)	94.03 (2.93)	96.73 (2.19)
Liver	3.20 (0.41)	61.27 (3.92)	**65.52** (4.37)	60.81 (3.81)	62.55 (4.99)	64.47 (2.70)	63.02 (3.93)	60.17 (4.41)
Mammographic	2.90 (0.31)	**82.04** (3.62)	76.75 (11.35)	81.20 (3.89)	78.58 (11.02)	81.22 (3.01)	77.92 (9.29)	81.45 (3.48)
Monk2	2.50 (0.51)	90.55 (2.64)	87.31 (4.78)	**92.96** (1.48)	86.89 (5.50)	92.63 (2.15)	90.27 (3.14)	92.87 (1.53)
Pima	3.50 (0.51)	73.54 (2.85)	73.75 (2.59)	73.46 (2.58)	73.88 (2.47)	**74.50** (2.48)	74.14 (2.61)	73.09 (2.57)
Sonar	3.30 (0.66)	79.42 (4.41)	82.69 (4.13)	79.80 (4.16)	80.00 (4.47)	77.40 (4.40)	**82.78** (4.81)	77.30 (4.39)
Thyroid	3.80 (0.41)	96.93 (0.99)	96.99 (1.02)	97.05 (1.04)	96.41 (1.72)	97.05 (1.04)	**97.10** (1.10)	97.05 (1.04)
Vehicle	5.60 (0.50)	71.08 (2.20)	**73.18** (1.71)	71.88 (1.83)	73.06 (1.68)	73.01 (1.38)	72.33 (2.05)	70.99 (2.15)
Vertebral	2.50 (0.69)	81.66 (1.56)	84.48 (4.32)	83.46 (1.70)	85.89 (2.35)	**87.05** (2.59)	86.53 (2.37)	82.43 (1.02)
Weaning	3.00 (0.00)	77.36 (3.76)	80.26 (4.16)	76.84 (2.74)	**81.71** (2.92)	79.21 (3.21)	79.40 (4.48)	76.97 (2.84)
Wine	2.90 (0.31)	97.33 (2.23)	96.88 (3.48)	97.55 (1.89)	96.55 (2.92)	97.55 (1.89)	**97.66** (2.21)	97.33 (2.34)
Average	3.34	81.51	81.30	81.79	81.83	**82.39**	82.16	81.31



(a) N_{PM}



(b) N_{100}

Figure 4.14: Win-tie-loss graph of the proposed method in terms of mean accuracy achieved in comparison with the classical ensemble methods with a pool size of (a) N_{PM} and (b) N_{100}.

5

Conclusion

In this work, a general view of MCSs was introduced. It was explained that, in order to yield a better solution than a single strong classifier, the pool of a MCS should contain accurate and diverse classifiers. For DS techniques, which selects, for each test instance, the most suitable classifiers in the pool to label it, it was shown the importance of having local experts in the pool.

Furthermore, the Bagging, Random Subspace and Boosting methods and their main characteristics were presented in more detail. As previously shown, the number of classifiers in the pool is an input parameter to those methods, even though it is reasonable that the amount of classifiers in a solution should vary with the problem. Also, the training of the entire pool usually takes too much time.

The pool of classifiers generation method proposed in this work was designed to tackle both issues. A step-by-step example showed how the proposed method generates a pool of classifiers with a very small number of Perceptrons. The proposed method works so that the oracle for the training dataset is 100%, and the generated classifiers are not trained, in order to reduce the process time.

An analysis of the proposed method using synthetic data showed that it is less sensitive to variations on test data distribution than the other ensemble methods discussed in this work, despite the high dependency on data disposition the pool generation process has. Also, the proposed method was analysed using the P2 Problem, which is a problem with complex boundaries, and it was concluded that the proposed method is capable of generating a very small pool of classifiers as accurate as a large pool generated by the classical ensemble methods for this kind of problem. Furthermore, an analysis regarding multi-class problems showed that, although each Perceptron generated by the proposed method is only able to choose between two classes, the proposed method can achieve at least the same accuracy rate as other ensemble methods presented in this work.

The performance of the proposed method, both in time and accuracy, was evaluated using five DCS techniques over 20 datasets. The experimental results showed that the proposed method generates very few classifiers for each classification problem, and the classical ensemble methods were ten times slower in generating the same amount of classifiers than the proposed

method. Moreover, the proposed method was able to yield a similar performance in terms of accuracy than the other ensemble methods for the same and for a greater pool size. Thus, it can be concluded that the proposed method yields approximately the same accuracy while requiring much less processing time than the other pool of classifiers generation methods.

Improvements on the proposed method may involve the definition of more sophisticated heuristics to obtain the hyperplanes, and thus enhance the performance. Also, since the proposed method generates very few and diverse classifiers, the pool generated is generally not fit for DES techniques, so other possible future works may concern adaptations of the algorithm in order to make it suitable for ensemble selection.

References

[1] A. BERTONI, R. F.; VALENTINI, G. Random subspace ensembles for the biomolecular diagnosis of tumors. In: MODELS AND METAPHORS FROM BIOLOGY TO BIOINFORMATICS TOOLS, NETTAB. **Anais...** [S.l.: s.n.], 2004.

[2] A. H. R. KO R. SABOURIN, u. S. B. J. From dynamic classifier selection to dynamic ensemble selection. **Pattern Recognition**, [S.l.], v.41, p.1718 – 1731, 2008.

[3] BREIMAN, L. Bagging predictors. **Machine Learning**, [S.l.], v.24, p.123 – 140, 1996.

[4] BREIMAN, L. Arcing classifiers. **The Annals of Statistics**, [S.l.], v.26, p.801 – 849, 1998.

[5] BRITTO A., S. R.; OLIVEIRA, L. Dynamic selection of classifiers - A comprehensive review. **Pattern Recognition**, [S.l.], v.47, p.3665 – 3680, 2014.

[6] CHAWLA, N. V.; BOWYER, K. W. Random subspaces and subsampling for 2-d face recognition. In: IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, (CVPR), VOLUME 2. **Proceedings...** [S.l.: s.n.], 2005. p.582 – 589.

[7] DIETTERICH, T. G. Ensemble methods in machine learning. In: MULTIPLE CLASSIFIER SYSTEMS, LECTURE NOTES IN COMPUTER SCIENCE, VOL. 1857, Berlin, Heidelberg. **Anais...** Springer, 2000. p.1 – 15.

[8] EFRON B, T. R. **An Introduction to the Bootstrap.** [S.l.]: Chapman & Hall, 1993.

[9] FREUND, Y. An adaptive version of the boost by majority algorithm. **Machine Learning**, [S.l.], v.43, p.293 – 318, 2001.

[10] FREUND, Y. A more robust boosting algorithm. **Statistics – Machine Learning**, [S.l.], 2009.

[11] FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. **Journal of Computer and System Sciences**, [S.l.], v.55, p.119 – 139, 1997.

[12] HO, T. K. The random subspace method for constructing decision forests. **IEEE Transactions on Pattern Analysis and Machine Intelligence.**, [S.l.], v.20, p.832 – 844, 1998.

[13] J. ALCALá-FDEZ A. FERNáNDEZ, J. L. J. D. S. G. KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. **Multiple-Valued Logic Soft Comput.**, [S.l.], v.17, p.255 – 287, 2011.

[14] J. KITTLER M. HATEF, R. P. W. D. J. M. On combining classifiers. **IEEE Transactions on Pattern Analysis and Machine Intelligence.**, [S.l.], v.20, p.226 – 239, 1998.

[15] K. BACHE, M. L. **UCI machine learning repository**. 2013.

[16] KUNCHEVA, L. **Combining pattern classifiers.** [S.l.]: J. Wiley., 2004.

[17] KUNCHEVA, L. **Ludmila kuncheva collection**. 2004.

[18] L. I. KUNCHEVA J. J. RODRÍGUEZ, C. O. P. D. E. J. L.; JOHNSTON., S. J. Random subspace ensembles for fMRI classification. **IEEE Transactions on Medical Imaging**, [S.l.], v.29, p.531 – 542, 2010.

[19] M. O. CRUZ R., S. R.; D. C. CAVALCANTI, G. **A DEEP analysis of the META-DES framework for dynamic selection of ensemble of classifiers.** 2015.

[20] R. M. O. CRUZ R. SABOURIN, G. D. C. C. T. I. R. META-DES: a dynamic ensemble selection framework using meta-learning. **Pattern Recognition**, [S.l.], v.48, p.1925 – 1935, 2015.

[21] R. P. W. DUIN P. JUSZCZAK, D. d. R. P. P. E. P. D. M. T. **Prtools, a matlab toolbox for pattern recognition**. 2004.

[22] RAJ D. IYER DAVID D. LEWIS, R. E. S. Y. S.; SINGHAL, A. Boosting for document routing. In: IN PROCEEDINGS OF THE NINTH INTERNATIONALCONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT. **Anais...** [S.l.: s.n.], 2000.

[23] R.D. KING C. FENG, A. S. **Statlog: comparison of classification algorithms on large real-world problems**. 1995.

[24] SCHAPIRE R. E., F. Y. B. P.; LEE, W. S. Boosting the margin: a new explanation for the effectiveness of voting methods. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING., 14., Nashville, USA. **Proceedings...** [S.l.: s.n.], 1997. p.322 – 330.

[25] SINGER, Y.; SCHAPIRE, R. E. BoosTexter: a boosting-based system for text categorization. **Machine Learning**, [S.l.], v.39, p.135 – 168, 2000.

[26] SKURICHINA, M.; DUIN, R. P. W. Bagging, Boosting and the Random Subspace Method for Linear Classifiers. **Pattern Analysis & Applications**, [S.l.], v.5, p.121 – 135, 2002.

[27] STEFANO MERLER CESARE FURLANELLO, B. L.; SBONER, A. Tuning costsensitive boosting and its application to melanoma diagnosis. In: IN MULTIPLE CLASSIFIER SYSTEMS: PROCEEDINGS OF THE 2ND INTERNATIONAL WORKSHOP. **Anais...** [S.l.: s.n.], 2001. p.32 – 42.

[28] SUN, S.; ZHANG, C. Using a Random Subspace Predictor to Integrate Spatial and Temporal Information for Traffic Flow Forecasting. In: **Advances in Natural Computation**. [S.l.]: Springer Berlin Heidelberg, 2005. p.652–655. (Lecture Notes in Computer Science, v.3611).

[29] VALENTINI, G. An experimental bias-variance analysis of svm ensembles based on resampling techniques. **IEEE Transactions on Systems, Man, and Cybernetics**, [S.l.], v.Part B 35, p.1252 – 1271, 2005.

[30] VIOLA, P.; JONES, M. J. Robust real-time face detection. **International Journal of Computer Vision**, [S.l.], v.57, p.137 – 154, 2004.

[31] WOLPERT, D.; MACREADY, W. No free lunch theorems for optimization. **IEEE Transactions on Evolutionary Computation**, [S.l.], v.1, p.67 – 82, 1997.

[32] WOŹNIAK M., G. M.; CORCHADO. A survey of multiple classifier systems as hybrid systems. **Information Fusion**, [S.l.], v.16, p.3 – 17, 2014.

[33] ZHOU, Z. **Ensemble methods.** [S.l.]: Taylor & Francis., 2012.