Université de Tunis

Institut Supérieur de Gestion

Ecole Doctorale en Sciences de Gestion

Solving data fusion problems using ensemble approaches

Mémoire en vue de l'obtention du diplôme du
Mastère de recherche en Informatique de Gestion

Elaboré par

Afef Ben Brahim

Sous la direction de

Professeur Mohamed Limam

Session : Juin 2010

Abstract

In classification problems where additional information is obtained from different data sources having heterogeneous features, the classifier trained earlier on the original data can not be updated to learn the new data. One solution is to fuse this data by using a classifier to learn from each source and to combine the outputs in order to get a more accurate final decision. Ensemble methods are well suited to solve data fusion problems, using all the data sources available while taking advantage of the complementary information. Robust techniques are proposed for combining classifiers, aimed at reducing the effect of outlier classifiers in the ensemble. The proposed techniques are shown to have better performance leading to significantly better classification results than those previously employed technique.

Résumé

Dans un problème de classification où les données sont obtenues à partir de sources différentes ayant des attributs hétérogènes, un seul classifieur ne peut pas apprendre à partir de toutes ces données à la fois. La meilleure solution est de fusionner ces données par la construction d'un modèle pour chaque source et combiner les résultats afin d'obtenir une décision finale plus correcte. Les méthodes d'ensemble sont bien adaptées à ce genre de problèmes qui nécessitent la fusion de données. Ce mémoire vise à étudier l'efficacité de ces méthodes dans le traîtement de problèmes de fusion de données, par l'exploitation de toutes les sources de données disponibles afin de tirer profit de l'information complémentaire qu'elles fournissent. Des méthodes robustes sont proposées pour la combinaison des classifieurs, visant à réduire l'influence de ceux qui sont non fiables. Les méthodes de combinaison proposées sont plus efficaces que les autres algorithmes évalués et donnent de bons résultats de classification après le processus de fusion de données.

Acknowledgements

I would like to express my utmost gratitude to my advisor, Prof. Mohamed Limam for providing me an opportunity to conduct my master s research under his supervision, for his expertise, guidance and constant support.

I am also thankful to Dr. Riadh Khanchel for his suggestions, comments and additional guidance which were valuable to the completion of this work.

Additional thanks to my professors of the master's degree in "Informatique de gestion". It was a great pleasure to be their student.

Finally, I would like to thank my family, especially my mom, for all her encouragement and support and my father who is my role model.

Contents

	Page
Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Figures	v
List of Tables	vi
Introduction	1
1 Ensemble Methods for classification	3
1.1 Introduction	3
1.2 Reasons for using ensembles of classifiers	4
1.2.1 Statistical reasons	5
1.2.2 Computational reasons	5
1.2.3 Representational reasons	5
1.2.4 Large or insufficient volume of data	6
1.2.5 Data fusion	6
1.3 Methods for constructing ensembles	7
1.3.1 Bagging and variations	7
1.3.2 Boosting and variations	9
1.3.3 Comparison of Bagging and Boosting	13
1.3.4 Stacked generalization	14
1.3.5 Mixture of experts	14
1.4 Combining ensemble classifiers	15
1.4.1 Combining class labels	15
1.4.2 Combining continuous outputs	18
1.5 Ensemble diversity	21
1.5.1 Techniques to achieve diversity	22
1.5.2 Measures of diversity	26
1.6 Taxonomies for ensemble methods	29
1.6.1 Fusion and selection	30
1.6.2 Decision optimization and coverage optimization	30
1.6.3 Trainable and non-trainable ensembles	31

1.6.4 Top-down and bottom-up ensembles . . . 31
1.6.5 Dependent and independent ensembles . . . 32
1.7 Conclusion . . . 33
2 Ensemble methods for data fusion . . . 36
2.1 Introduction . . . 36
2.2 Data fusion . . . 36
2.2.1 Definitions and applications . . . 37
2.2.2 Data fusion levels . . . 38
2.2.3 Trade-offs for different levels . . . 39
2.3 Techniques for data fusion . . . 42
2.3.1 Bayesian inference . . . 42
2.3.2 Dempster-Shafer theory of evidence (DS) . . . 43
2.3.3 Neural networks . . . 45
2.3.4 Kernel methods . . . 46
2.4 Ensemble methods for data fusion . . . 46
2.5 Evaluation of Learn++ algorithm . . . 48
2.6 Conclusion . . . 58
3 Data fusion by robust classifiers combination . . . 60
3.1 Introduction . . . 60
3.2 The robust aggregation rule . . . 62
3.2.1 Training stage . . . 62
3.2.2 Conflict resolution and decision making . . . 63
3.3 Proposed algorithms . . . 64
3.4 Experimental setup . . . 65
3.4.1 Databases . . . 65
3.4.2 Classifiers used as base learners . . . 69
3.5 Implementation and performance analysis . . . 71
3.5.1 Results summary . . . 85
3.6 Conclusion . . . 89
General Conclusion . . . 89
Bibliography . . . 92

List of Figures

1.1 The process of ensemble methods. 4
1.2 BKS method for classifiers combination. 17
1.3 Decision profile for classifiers' outputs. 18
1.4 Average fusion. 19
1.5 Decision templates for classifier combination. 22
1.6 Four levels for categorizing ensemble methods. 32

2.1 Illustration of the raw fusion process. 39
2.2 Illustration of the feature fusion process. 40
2.3 Illustration of the decision fusion process. 40
2.4 Schematic representation of Learn++ algorithm. 53
2.5 Fusion performance using Learn++ with T=10. 55
2.6 Fusion performance using Adaboost.M1 with T=10. 56

3.1 Fusion performance on Multiple feature database with T=10. 76
3.2 Fusion performance on Wine database. 78
3.3 Fusion performance on Sonar database. 81
3.4 Fusion performance on Ionosphere database. 83
3.5 Fusion performance on SPECTF database. 86

List of Tables

1.1 The relationship table with probabilities 27
1.2 Taxonomy for categorizing ensemble methods. 34

2.1 Tradeoffs for different levels of data fusion. 42
2.2 Fusion performance obtained by Learn++ with MLP and DT. 54
2.3 Fusion performance obtained by Adaboost.M1 with MLP and DT. . . . 55
2.4 Fusion performance obtained by Learn++ with MLP and DT on Wine dataset. 57
2.5 Fusion performance obtained by Adaboost.M1 with MLP and DT on Wine dataset. 57

3.1 Notation of the different variations proposed for Learn++. 65
3.2 Number of hidden layer nodes for each feature set. 73
3.3 Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on multiple feature dataset. 73
3.4 Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on multiple feature dataset. 74
3.5 Fusion performances obtained by Adaboost algorithm with MLP and DT on multiple feature dataset. 75
3.6 Number of hidden layer nodes for each feature set. 75
3.7 Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on Wine dataset. 76
3.8 Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on Wine dataset. 77
3.9 Fusion performances obtained by Adaboost algorithm with MLP and DT on Wine dataset. 77
3.10 Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on Sonar dataset. 79
3.11 Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on Sonar dataset. 79

3.12 Fusion performances obtained by Adaboost algorithm with MLP and DT on Sonar dataset. 80
3.13 Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on Ionosphere dataset. 81
3.14 Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on Ionosphere dataset. 82
3.15 Fusion performances obtained by Adaboost algorithm with MLP and DT on Ionosphere dataset. 83
3.16 Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on SPECTF dataset. 84
3.17 Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on SPECTF dataset. 84
3.18 Fusion performances obtained by Adaboost algorithm with MLP and DT on SPECTF dataset. 85
3.19 Summary of fusion performances for all databases 87

Introduction

Ensemble methods emerged as a powerful methodology for improving the prediction performance as well as the robustness of a model. Their success has been observed in multiple disciplines, including intrusion detection, anomaly detection, web applications, etc. The main idea of ensemble methods is to strengthen weak models by the combination of diversified base classifiers. Diversity is a key element for the effectiveness of ensemble methods. Different strategies are used to achieve diversity based mostly on the manipulation of the learning data by techniques such as sampling, partitioning (Dietterich (2000)).

Enormous amounts of data are continuously generated from different sources and views. For intelligent decision making, taking advantage of all the data available is important to consolidate different concepts. Fusion of heterogeneous data sources underlying the same problem presents a natural fit for ensemble systems since different classifiers could be generated using data obtained from different sources, and then combined to achieve the desired data fusion.

This dissertation addresses the issue of how to effectively use ensemble methods to optimally combine multiple sources of information in order to make a decision. Two ensemble methods are evaluated for data fusion problems, namely Adaboost.M1 (Freund and Schapire (1997)) and Learn++ (Polikar et al. (2008)). Moreover, the use of

robust combination rules, instead of those employed originally by these algorithms, is investigated. Our proposed techniques aim at reducing the effect of outlier classifiers in the ensemble. The algorithms and the variations obtained by the use of different combination techniques are evaluated experimentally on five benchmark databases.

Chapter 1 provides an overview of ensemble methods focusing on two fundamental components that are strategies used for constructing diverse ensembles and strategies used for combining them. In Chapter 2, the use of ensemble methods for the combination of heterogeneous sources of data is discussed and two ensemble approaches are evaluated for this purpose. Chapter 3 investigates the effectiveness of employing robust combination techniques with ensemble methods in the context of data fusion. The robust combination rules developed are based on conflict resolution and aim at reducing the influence of outlier models in the ensemble. A summary of the results concludes the work.

Chapter 1

Ensemble Methods for classification

1.1 Introduction

Ensemble learning consists on constructing a set of classifiers, such as decisions trees or neural networks, for the same original problem. To classify a new instance, decisions of single classifiers are combined by voting or averaging leading to a more accurate classification decision.

This process imitates human's second nature to consult several persons before making a final decision in different life domains such as medicine, finance, social problems, etc. In machine learning, this process has been extended to improve the performance of the decision making in many domains like bioinformatics, remote sensing, manufacturing, geography, information security, information retrieval and image retrieval. Ensemble methods have enjoyed success and popularity since two decades. The progress started when Schapire (1990) introduced the idea of Boosting the low accuracy of a weak learning algorithm that performs only slightly better than random guessing in the probably approximately correct (PAC) learning model. He showed that a strong classifier can be obtained by combining an ensemble of weak classifiers. At the same time,

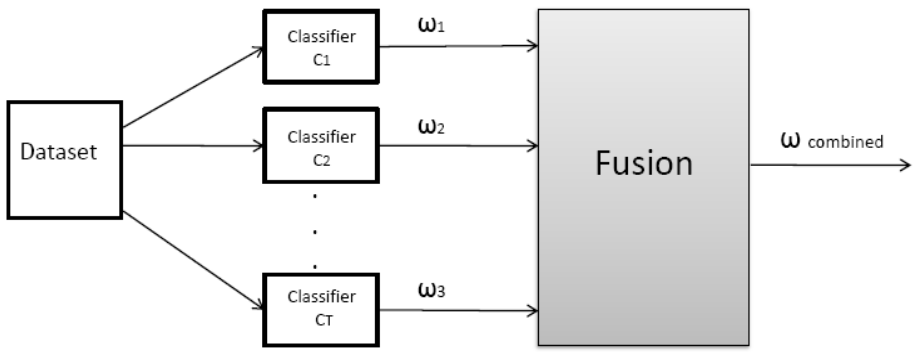

Figure 1.1: The process of ensemble methods.

Hansen and Salamon (1990) introduced the method of running an ensemble of neural networks trained on the same database and combined using a consensus scheme.

This chapter reviews ensemble methods and describes their main characteristics. It is organized as follow: Section 1 explores the reasons behind constructing ensemble methods and the cases where a classification problem can not be solved by a single classifier. Section 2 describes the most popular methods for constructing ensembles, their strengths, their drawbacks and their different characteristics. Section 3 presents the methods for combining ensemble classifiers. Ensemble diversity, its importance and the ways to achieve this principal condition to have a good ensemble are discussed. Finally, we conclude by a review of some taxonomies for characterizing ensemble methods.

1.2 Reasons for using ensembles of classifiers

There are many reasons why we need an ensemble of classifiers instead of a single one. Dietterich (2000) reports three principal reasons namely statistical, computational and representational. However, there are also other cases where ensemble methods are

preferred.

1.2.1 Statistical reasons

In many applications, the amount of training examples is not sufficient to be representative of the future data. Thus, the learning algorithm can give different hypotheses which may have similar accuracy on the training data, but different generalization performances when classifying new instances. In such cases, constructing many classifiers and combining their predictions, by voting or averaging, minimize the risk of choosing a classifier with a low generalized performance.

1.2.2 Computational reasons

Certain problems are very complex in a way it is computationally difficult for a learning algorithm to obtain the best hypothesis, even when there is not a problem concerning the amount of training data. However, constructing an ensemble of classifiers and combining them using an appropriate combination scheme can solve this difficult task. This classification system is based on a divide-and-conquer process as the data space is partitioned into smaller subspaces, and each classifier learns from a partition. The general decision is then obtained by combining the outputs of the ensemble members.

1.2.3 Representational reasons

The aim of a supervised learning process is to approximate an unknown function f which gives the classification decision. Usually, no one of the different hypotheses obtained by the learning algorithm is representative of the true function f. For example, if the optimal classifier that we want to obtain is nonlinear, and the space of possible classifiers contains only linear classifiers, then the classifier that we are looking for does not belong

to this space. Instead of training a classifier of high complexity, an ensemble of linear classifiers can be a simpler and better alternative to solve such problems.

1.2.4 Large or insufficient volume of data

Sometimes, the volume of data available for a certain application is very large such that a single classifier can not effectively solve the problem. With such amount of data, the learning process is not practical as the algorithms become very slow in learning a model or classifier. So the solution for simplifying this task, while keeping all the data, is to divide the data into smaller subsets and train a classifier with each partition of data. Partitioning the data will not only solve the problem of exceeding the memory size, but will also lead to creating an ensemble of diverse and accurate classifiers, each built from a partition. The outputs of all the classifiers are then combined using a combination rule and a highly accurate collective decision is obtained.

The exactly opposite situation can be encountered when there is only small amount of data which could not be sufficient and representative of a certain problem. The classification algorithm can not successfully learn from such little training data. In this case, resampling techniques can be employed for drawing random subsets of the available data. An ensemble is created by training a different classifier for each subset. Bagging is one of the most effective resampling techniques for constructing classifier ensembles.

1.2.5 Data fusion

In some classification problems, additional information is obtained from different data sources having heterogeneous features. The classifier, trained earlier on the original data, can not be updated to learn the new data. One solution is to fuse this data by using a classifier to learn from each source and to combine the outputs in order to get a more

accurate final decision. Ensemble methods are well suited to such applications.

1.3 Methods for constructing ensembles

Several methods have been proposed in the literature for generating ensembles of classifiers to improve ensemble's diversity. The most popular approaches are discussed in this section.

1.3.1 Bagging and variations

Bagging: This method introduced by Breiman (1996a) turns to be one of the most intuitive and simplest to implement. This method is particularly useful for small volume of data. Bagging creates an ensemble of individual classifiers and uses them to get an aggregated predictor. The multiple classifiers are built by making bootstrap replicates of the original training data. In other words, new training sets are constructed by sampling with replacement, and therefore many of the original examples may appear more than once in the resulting training set while others may be left out. This technique is called bootstrap aggregation, from which the term Bagging is derived. Instability is the most important element to get a good ensemble. Breiman demonstrates that Bagging is effective on unstable learning algorithms where small changes in the training set lead to significant changes in the model created. Neural networks and decision trees are good examples of unstable learning algorithms. To classify a new instance, predictions of individual models are combined by averaging or voting to create a single output which represents the ensemble decision. The combined model obtained often performs better than the single model built from the original data. The steps of Bagging are given in Algorithm 1.

Random Forests: This method proposed by Breiman (2001) is a variation of Bagging

Algorithm 1 Bagging

Require: A training set S with m samples, where each sample x_j is of class y_j, base classifier I, and number of bootstrapped sets T.

1: **for** $t = 1$ to T **do**
2: S_t= bootsrapped bag from S
3: $h_t = I(S_t)$
4: **end for**
5: $h_{final} = \arg\max_{y \in \Omega} \sum_{t:h_t(x)=y}$

Ensure: The final hypothesis h_{final}

algorithm. It consists of a large ensemble of decision trees constructed by randomly varying their training parameters such that each tree is based on the values of a random vector. These variations can concern the training data, as in Bagging, but they can also concern the feature space by selecting randomly a different feature subset to split each node of the tree. The most straightforward random forest with random attributes is formed by selecting randomly a small group of variables to split at each node rather than choosing the best split among all features. To classify a new instance, classifiers outputs are combined using the majority vote. In addition of beeing fast, Random forests are able to handle a very large number of input variables.

Wagging: Bauer and Kohavi (1999) present a variant of Bagging where the base learning algorithm uses training instances with different weights. To form the training data for each classifier, wagging assigns random weights to the training examples. It uses the Poisson distribution to assign these weights. Alternatively it is possible to allocate weights from the exponential distribution.

Pasting Small Votes: In data mining community, there is an increasing need for producing results accurately and fast. However, machine learning algorithms can become very slow in learning a classifier when the dataset is very large. Ensemble methods like Bagging and Boosting have limited performance on such databases. Breiman (1999)

proposes pasting small votes to handle problems with very large amount of data. This method builds a large number of classifiers on small subsets of data, called bites, and then pastes these predictions. There are two variations of pasting small votes, Rvotes which randomly creates data subsets and Ivotes that creates consecutive training sets based on the importance of the examples. These methods are computationally fast and are also applicable to on-line learning.

1.3.2 Boosting and variations

Boosting: Schapire (1990) introduces an algorithm to boost the performance of a weak learner, such as decision trees or neural networks, to the level of a strong classifier. Boosting is considered to be an important development in the history of ensemble methods and machine learning in general. This method creates an ensemble of classifiers by iteratively training a weak learner on various versions of the training data. The classifiers obtained by the learning algorithm are then combined, by majority voting, into a single composite strong model that often achieves a higher performance than that of any of the individual classifiers. The original Boosting creates three weak classifiers $\{C_i, i = 1, 2, 3\}$ which must have an error rate that is less than 0.5. The first one C_1 is trained with a random subset of the available training data. The second classifier C_2 is trained on a data subset chosen based on the performance of C_1 and the third classifier C_3 is trained with examples on which the two first classifiers C_1 and C_2 disagree. The three classifiers are then combined using majority vote. Schapire showns that the performance of the aggregated model is better than that of the best classifier in the ensemble. One main reason why Boosting improves the performance is that it generates a composed classifier whose error on the training set is small, as it is obtained by combining many classifiers whose error may be high, another reason is that Boosting reduces error by reducing the variance. However, in some cases, Boosting leads to a poor generalized performance. According to Quinlan (1996), overfitting is the cause of this deterioration

since that large number of iterations may produce a very complex aggregated model that is less accurate than a single model. So it is recommended to keep the number of iterations as small as possible in order to avoid the problem of overfitting. Also some studies indicate that the performance of Boosting methods is sensitive to the characteristics of the dataset. Boosting may overfit noisy data sets and reduce the classification accuracy.

Adaboost: Adaboost developed by Freund and Schapire (1997), is the most famous version of the Boosting algorithm, it has enjoyed much attention and popularity. Adaboost has also resulted in many variations like AdaBoost.M1 and AdaBoost.R which are able to handle multiclass and regression problems, respectively. Adaboost generates an ensemble of hypotheses by training a weak classifier, and combines their outputs using weighted majority voting. This combination rule allows giving a higher voting weight to classifiers with good performance. Iteratively, the individual classifiers are trained with instances drawn after updating the distribution of the learning data. This distribution update is based on the performance of the previous classifier on the training data forcing the algorithm to focus on instances that have been misclassified. As a consequence, difficult examples are more likely to be included in the training data of the next iteration. There are two possibilities for Adaboost to form a new training set for the base learner. Boosting by sampling, in which examples are selected with replacement from the original training data with probability proportional to their weights. The second possibility, Boosting by weighting, can be used with learning algorithms that can learn from a weighted training set directly. Both methods are very effective. Adaboost requires a base learning algorithm which have an error less than 0.5. Generally, the global error decreases rapidly in the first iterations.

Adaboost.M1 and Adaboost.M2: Adaboost algorithm deals with binary classification. Freund and Schapire have proposed two variations of Adaboost which handle multiclass

Algorithm 2 Adaboost.M1

Require: sequence of m examples $[(x_1, y_1),...,(x_m, y_m)]$
with labels $y_i \in \Omega = \{1, ..., k\}$
Base classifier I
integer T Specifying number of iterations

1: **Initialize** $D_1(i) = \frac{1}{m}$ for all $i = 1...m$.
2: **for** $t = 1, 2, ..., T$ **do**
3: Call I, providing it with the distribution D_t.
4: Get back a hypothesis $h_t : X \to \Omega$
5: Calculate the error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$
if $\varepsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
6: Set $\beta_t = \varepsilon_t/(1 - \varepsilon_t)$
7: Update distribution D_t:
$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise.} \end{cases}$
where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution)
8: **end for**

Ensure: The final hypothesis:
$h_{final}(x) = \arg\max_{y\in\Omega} \sum_{t:h_t(x)=y} log(\frac{1}{\beta_t})$

classification problems, namely Adaboost.M1 and Adaboost.M2. For these algorithms, the error of the individual classifier is not required to be less than 0.5. Adaboost.R is another extension to Adaboost which deals with regression problems.

Arc-x4: Breiman (1998) references to the family of algorithms that construct ensembles by adaptively resampling and combining, by Arcing (Adaptively resample and combine) algorithms. The most popular example of arcing algorithms is Adaboost which is also called arc-fs. One other arcing method is Arc-x4 algorithm. Breiman described it as adhoc invention which is very similar to arc-fs except for the final step, where the weighting procedure that Adaboost uses, is not employed. Arc-x builds an ensemble of classifiers successively by updating the training data distribution taking into account the number of instances incorrectly predicted by previous models. Then a voting mechanism is used to combine the classifiers outputs when a new example arrives. Arc-x4

proves that the adaptive reweighting is the main reason for the high performance of Adaboost algorithm.

AveBoost: Oza (2003) presents another variation of Boosting algorithm which achieves diversity by training each base classifier using a data distribution update that is based on the performances of all the previous models to make the error of each hypothesis as uncorrelated as possible with those of the previous ones.

Real Adaboost: Friedman et al. (2000) propose a generalized version of Adaboost which produces an alternative formulation of Boosting especially for decision trees. This algorithm is adequate for large-scale data mining problems as it is faster computationally, and it can provide interpretable descriptions of the aggregate decision rule.

Gradient Boosting: Friedman (2002) develops this method that creates an ensemble by fitting a base learner to current ”pseudo” residuals by least squares at each iteration. These pseudo-residuals are the gradient of the loss functional that is minimized, taking into account the model values at each training data point evaluated at the current iteration. The accuracy and the execution time are improved by injecting randomness. At each iteration, a subsample of the training data is drawn randomly without replacement and used to fit the base classifier.

Quadratic Boosting: Quadratic Boosting, introduced by Phama and Smeuldersb (2008), is based on quadratic combination of base classifiers in order to improve the Adaboost algorithm. This strategy constructs an intermediate classifier operating on the combined linear and quadratic terms. It first trains a classifier by randomizing the labels of the training data. Iteratively, the labels are updated and the base classifier is called, and the ensemble is formed this way.

AdaboostKL and AdaboostNorm2: Problem of overfitting can occur when Adaboost

is used with highly noisy datasets. In this context, Sun et al. (2006) propose a strategy was proposed to avoid overfitting problem by penalizing the data distribution skewness in the learning process, this will prevent the most difficult examples from spoiling decision boundaries. To do this, two penalty functions are used, based on Kullback-Leibler divergence (KL) and L2 norm and two new variations of Adaboost are derived which are AdaboostKL and AdaboostNorm2. These two methods are effective on noisy datasets.

P-Adaboost: Adaboost is known to be a sequential algorithm, and this characteristic may be an obstacle in many situations. Merler et al. (2007) present P-Adaboost for the parallelization of Adaboost. P-Adaboost starts by operating the same way as Adaboost for a little number of iterations, then the rest of the classifiers are trained in parallel based on weights obtained from the earlier steps.

1.3.3 Comparison of Bagging and Boosting

Bagging and Boosting are both techniques for generating an ensemble of classifiers aiming at improving the classification accuracy. In order to combine the classifiers outputs, both methods employ a voting mechanism. In Boosting, the resampling of the training set is dependent on the performance of the previous classifiers, while in Bagging, classifiers are independent from each other. Unlike Bagging, the probability of selecting an instance for a Boosting classifier is not equal across the training data, it is proportional to the instances weight. Breiman (1996b) proved that Bagging is performing well when the learning algorithm is unstable, while Adaboost does not assume this instability. Bagging is not very effective on large datasets, given that bootstrap replicates will be very similar to the training set itself, and thus the classifiers will not be very different. In their studies, Opitz and Maclin (1999) showed that Boostings performance depends on characteristics of the dataset rather than the learning algorithm. They deduced that Bagging is adequate for most problems, but with appropriate datasets, Boosting produces better

accuracy. In the same context, Dietterich (2000) argumented that Bagging takes advantage from noise to achieve higher diversity and improve the accuracy, while Boosting tends to overfit the data in the presence of noise. Freund and Schapire (1997) explained the sensitivity of Adaboost to noise by the large amount of weight that mislabeled instances can receive. As a conclusion, the performance of Bagging and Boosting depends essentially on both the dataset and the base classifier.

1.3.4 Stacked generalization

Stacked generalization, developed by Wolpert (1992), is an ensemble method aiming to reduce the generalization error rate of a classifier. Stacked generalization introduces the concept of a meta learner. It creates an ensemble of classifiers, whose outputs are used as inputs to a second level meta-classifier to learn the mapping between the ensemble outputs and the actual correct classes. Stacked generalization can be seen as a more sophisticated version of cross-validation, but instead of using a winner-takes-all approach, the base learners are combined.

1.3.5 Mixture of experts

This ensemble method is proposed by Jacobs et al. (1991). It enables local classification where different classifiers are used, each one is concerned by the task of classification over some region of the feature space. It constitutes an ensemble of classifiers in a first level, followed by a second level classifier, usually a neural network called the gating network, which is trained through the expectation maximization algorithm. The weights used in the combination phase are instance specific, creating a dynamic combination rule. Mixture-of-experts can be considered as a classifier selection algorithm. Individual classifiers are experts in some portion of the feature space, and the combination rule selects the most appropriate classifiers weighted according to their expertise. The algo-

rithm can choose a single classifier with the largest weight, or calculate a weighted sum of the classifier outputs for each class, and select the class that has the highest weighted sum.

1.4 Combining ensemble classifiers

In addition to constructing ensembles, the strategy employed in combining their members is the other fundamental component of any ensemble system. In this section, several rules for combining classifiers are reviewed and grouped into two categories: combination rules for class labels and combination rules for continuous outputs.

1.4.1 Combining class labels

The following aggregation rules are used if only the class labels are available from the classifier outputs. Let $\Omega = \{\omega_1, \omega_2, ..., \omega_C\}$ be the set of classes for T classifiers, $D = \{D_1, D_2, ..., D_T\}$. In the following, we consider that given an instance x to classify into one of the C classes, the decision of a classifier D_i on class ω_j is represented by

$$d_{i,j} = \begin{cases} 1 & \text{if } D_i \text{ labels } x \text{ in } \omega_j \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

Majority Voting: Majority Voting chooses the class that receives the largest number of votes among the ensemble classifiers. There are three cases of majority voting (i) unanimous voting when all classifiers agree on the class ω_j; (ii) simple majority when at least one more than half the number of classifiers in the ensemble predict the class ω_j, and (iii) plurality voting, or just majority voting, when the class selected is the one that receives the highest number of votes. Mathematically, majority voting can be written as follows

$$\sum_{i=1}^{T} d_{i,a} = \max_{j=1}^{C} \sum_{i=1}^{T} d_{i,j}, \tag{1.2}$$

where $d_{i,a}$ is the decision of a classifier D_i to choose a class ω_a. For a two class problem, majority voting is an optimal combination method under the conditions that the classifier outputs are independent and that there is an odd number of classifiers.

Weighted Majority Voting (WMV): This combination rule is proposed by Littlestone and Warmuth (1994). It is useful when certain classifiers are more qualified than others and giving them a higher weight may improve the final decision. The idea of WMV is to assign a weight w_i to each classifier in proportion to its estimated performance. Classifiers' decisions are combined through WMV leading to the choice of class ω_a if

$$\sum_{i=1}^{T} w_i d_{i,a} = \max_{j=1}^{C} \sum_{i=1}^{T} w_i d_{i,j}. \tag{1.3}$$

The selected class is the one receiving the largest total weight. A possible strategy to assign weights is to use the performance of a classifier on a validation dataset, or even on the training dataset. Adaboost combines the ensemble classifiers by WMV, it assigns a voting weight to each classifier based on its training error.

Behavior Knowledge Space (BKS): This combination method is introduced by Huang and Suen (1995). It uses a look up table, constructed by counting frequencies for each combination of the classifiers' decisions based on the training data classification. For every combination of labels obtained during testing, the predicted class is the true class for which the frequency of this particular labels combination is highest in the BKS table. This combination technique is best described in Figure (1.2). The shortcoming of this method is that the lookup table will not be adequately filled if there are not sufficient training samples. This limitation can lead to situations of a tie between labels or of no

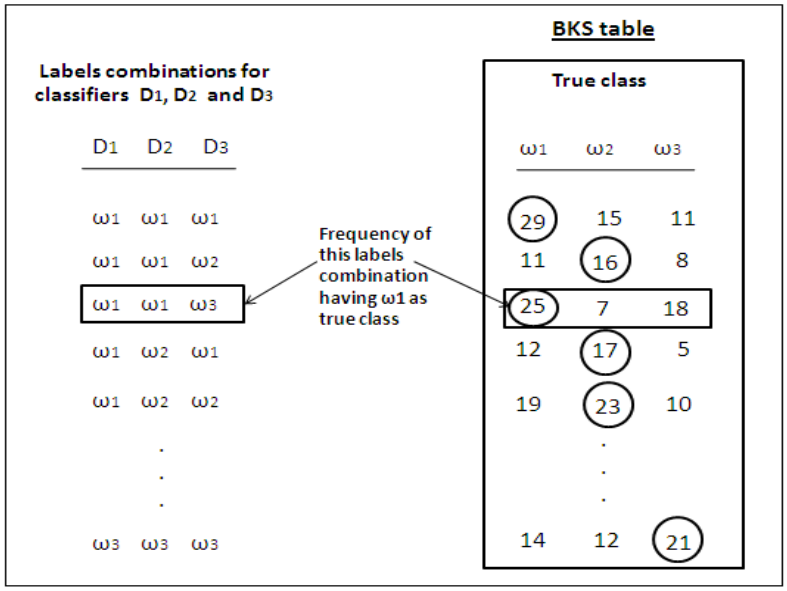

Figure 1.2: BKS method for classifiers combination.

labels in an indexed cell.

Borda Count: This method was originally introduced in 1770 by Jean-Charles de Borda, and it was adapted to classification problems. Borda count considers each classifier as a voter and the classes as the candidates. This method is always used when the classifiers can give a complete ranking of all the classes. If the outputs of the classifiers are continuous, then they can be rank ordered according to the support they have. If there are N classes, the highest ranked class gets N votes and each subsequent alternative receives one vote less. Then, for each alternative, the votes are added up across all classifiers and the alternative with the highest number of votes is chosen as the ensemble decision.

Naive Bayes combination (Pazzani and Domingos (1997)): Also called independence model since it requires the independence of individual classifiers in the ensemble. For

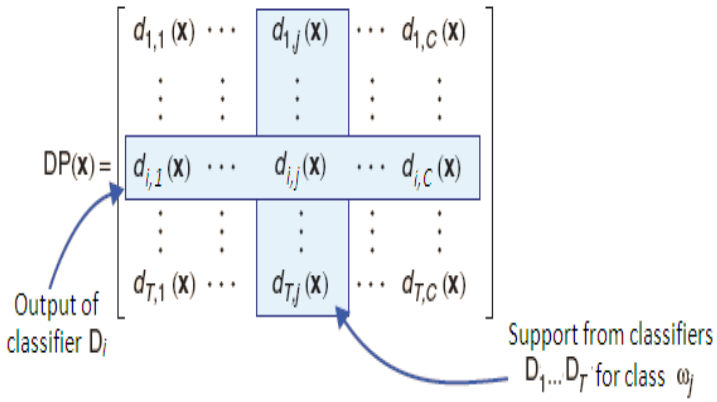

Figure 1.3: Decision profile for classifiers' outputs.

each classifier D_i, it constructs a confusion matrix for C classes by training D_i with the learning data set. When classifying a new instance, the confusion matrix is used to calculate conditional probabilities indicating which of the classes predicted by the classifiers is the ensemble decision concerning that instance.

1.4.2 Combining continuous outputs

The continuous output is the support given by a classifier to a certain class, and it is usually interpreted as an estimate of its posterior probability when the supports over all classes are normalized to sum to 1. To present the combination rules from the same perspective, Kuncheva et al. (2001) define the decision profile matrix ($DP(x)$) which organizes the T classifiers' outputs for a given instance x. The Decision profile is illustrated in Figure (1.3).

Note that $d_{i,j}$ is the support given by a classifier D_i to a class ω_j when classifying an example x. The overall support received by class ω_j, is defined using a combination

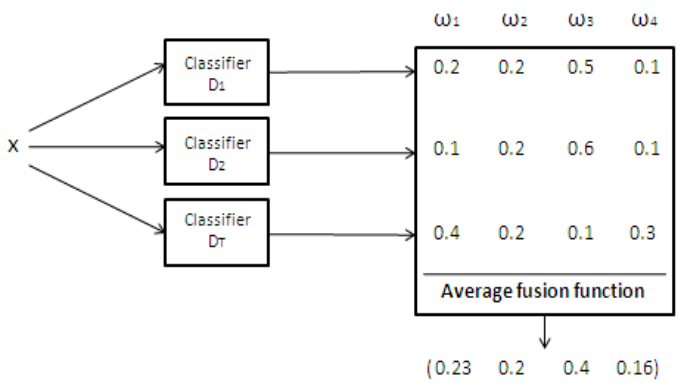

Figure 1.4: Average fusion.

function F as follows

$$\mu_j(x) = F[d_{1,j}(x), ..., d_{T,j}(x)], \tag{1.4}$$

where F is a combination function that could be one of the following :

Average: It is the simplest among the algebraic combiners. The support for ω_j is obtained as the average of all classifiers' supports for this class.

$$\mu_j(x) = \frac{1}{T}\sum_{i=1}^{T} d_{i,j}(x). \tag{1.5}$$

The class receiving the highest support is selected as the ensemble decision. An example of this combination rule is given in Figure (1.4) where ω_3 is chosen as the final decision.

Trimmed Mean: When a classifier gives unusually extreme values of support to a particular class, this will affect the final decision. To solve this problem, for a K% trimmed mean, K% of the supports is removed on each side. The mean of the remaining degrees of supports for a class ω_j presents the overall support $\mu_j(x)$.

Minimum/Maximum Rule: These rules are based on the classifiers outputs which have the minimum or the maximum support. For the maximum rule, the class selected as the ensemble decision is the one receiving the highest support value

$$\mu_j(x) = \max_{i=1}^{T}\{d_{i,j}(x)\}. \tag{1.6}$$

The minimum rule selects the class having the minimum support among the classifiers outputs, it is defined as:

$$\mu_j(x) = \min_{i=1}^{T}\{d_{i,j}(x)\}. \tag{1.7}$$

Product Rule: Product rule assumes reliable support estimates, it is adequate only if the single classifiers are independent. Such classifiers may be formed by training on different feature sets. In this combination method, the supports obtained by the classifiers are multiplied. However, it is very sensitive to supports that are very small and close to zero. The product rule is defined as

$$\mu_j(x) = \frac{1}{T}\prod_{i=1}^{T} d_{i,j}(x) \tag{1.8}$$

The sum rule: This rule assumes also independent classifiers with small differences in their outputs.The errors of the supports provided by such an ensemble of classifiers are averaged by the summation. A good example may be an ensemble of classifiers based on the same learning algorithm in the same feature space, but trained with different training sets like in Bagging where the successive training sets are bootstrap replicates from the original training data.

Decision Templates: Kuncheva et al. (2001) introduce decision templates, a rule for adapting the combination method to the application. For each class ω_j, a decision template (DT_j) is defined as the most presentative decision profile observed for this class

throughout training. To classify a novel instance, its decision profile is compared to the decision templates using a measure of similarity and the class whose decision template is the most similar is selected as the final decision. This combination rule can be formulated as

$$\mu_j(x) = S(DP(x), DT_j), j = 1, ..., C. \tag{1.9}$$

The similarity measure can be a squared Euclidian distance, hence the support for class ω_j will be:

$$\mu_j(x) = 1 - \frac{1}{T \times C} \sum_{i=1}^{T} \sum_{k=1}^{C} [DT_j(i,k) - d_{i,k}(x)]^2. \tag{1.10}$$

The procedure of aggregating classifiers using the decision templates is clarified in Figure (1.5) taken from Kuncheva et al. (2001).

Dempster-Shafer based combination: The Dempster-Shafer theory of evidence or theory of belief function is proposed by Dempster (1967) and extended by Shafer (1976). This method combines elements of evidence related to the classifiers of the ensemble. It is a generalization of the Bayesian approach, but it uses belief functions instead of probabilities. Dempster-Shafer theory differs from the Bayesian reasoning in that the belief in a fact and its negation are not required to sum to 1. Consequently, the selected class is the one that maximizes the value of the belief function.

1.5 Ensemble diversity

The goal of ensemble methods is to obtain a strong model by aggregating many weak classifiers. The idea is that classifiers will complement each other. This can only be achieved if they make errors on different instances, then an appropriate combination of

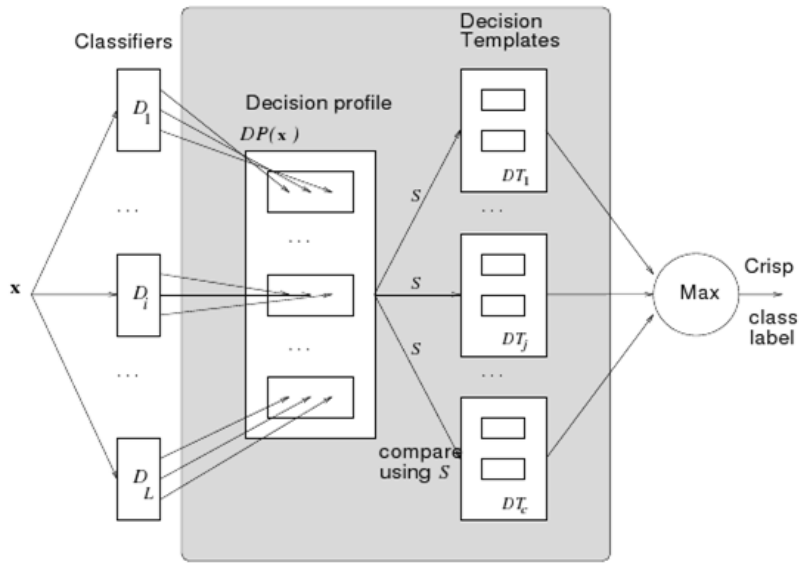

Figure 1.5: Decision templates for classifier combination.

these classifiers will reduce the global error. Diversity is an important requirement for ensemble success. To get a good ensemble, classifiers should be reasonably correct and in the mean time disagree as much as possible. Classifier diversity can be achieved in several ways. In the following, various techniques used for creating diverse ensembles are described, followed by a survey of measures proposed to quantify the ensemble diversity.

1.5.1 Techniques to achieve diversity

Manipulating the training samples

In this method, the learning algorithm is trained successively on different subsets from the original training data. This technique is used especially when the base classifiers are unstable, that means small changes in the training set leads to significant changes in the model created. Neural networks and decision trees are examples of unstable classifiers.

Bagging and Boosting are the best examples of ensemble methods using this technique. Creating samples of data for different classifiers can be done randomly like for Bagging which forms training sets for the consecutive classifiers by selecting samples of data randomly with replacement called bootstrap replicates. Sampling the data can be based on instances weight. Adaboost creates each training set by focusing on instances with large weight, those having been incorrectly classified by the classifier of the previous iteration. Cross validated committees select samples of data by leaving out a subset of the learning data in each iteration. Another possibility is the creation of additional data, Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples (DECORATE), proposed by Melville and Mooney (2005), creates training data samples that contain instances from the original learning data and additional artificially constructed training examples.

Manipulating the output targets

This method achieves diversity by manipulating the output attributes given to the base classifier. Dietterich and Bakiri (1995) develop the error correcting output codes (ECOC) which is an example of such methods. This ensemble strategy deals with problems with multiple classes. It uses a code matrix to decompose the multi class problem into multiple binary problems. ECOC can be combined with any learning algorithm.

Partitioning

In this approach, the original training set is divided into small subsets, in which different classifiers are trained. Partitioning the original training set can be horizontal or vertical. The horizontal partitioning provides different small datasets, each formed by a subset of the instances contained in the original dataset, and having the same feature space. The vertical partitioning provides different small datasets each formed by the same instances contained in the original dataset, but only a subset of the original feature space.

Horizontal partitioning: The horizontal partitioning is especially useful when the volume of the data available for a certain problem is very large that a single classifier can not effectively handle. In this case, the data can be partitioned into smaller disjoint subsets. Clustering can be used to have horizontal partitions of the learning data. The cluster-based concurrent decomposition algorithm (Rokach et al. (2005)), is a method which uses horizontal partitioning. It first applies the K-means algorithm to divide the training example into clusters. Then it forms disjoint subsets of data, each one containing instances from all clusters, in order to be representative of the original training set. A classifier is trained with each subset and when classifying a new instance, the aggregated decision is obtained by voting the classifiers outputs for this instance.

Vertical partitioning: Vertical partitions of the training data are obtained by manipulating the input features. Each classifier will be trained on a different projection of the training set. This strategy is appropriate with high dimensional datasets. In some applications, objects may be naturally represented by features extracted from different domains, e.g when objects are patients with a particular disease, characterized by features extracted from a set of answers on a medical checklist to be combined with the results of medical tests, or when the classification problem is about identification by speech and by images. To create feature subsets, three possible strategies can be used. random-based, reduct-based and collective-performance-based strategy.

Random feature selection: This is the simplest strategy. Bryll et al. (2003) present Attribute Bagging (AB), an ensemble approach that creates individual classifiers using random subsets of features. AB can be used with any learning algorithm. It starts by looking for an appropriate size for the feature subsets and then selects features randomly. Ensemble members are trained on projections of the training set and then a voting mechanism is used to classify new instances. The authors showed that AB outperforms Bagging both in accuracy and stability.

Reduct-based strategy: A reduct is the smallest feature subset having the same classification performance as the entire feature set.

Collective-Performance-based strategy: This strategy selects feature subsets based on the collective performance of the features. Oza and Tumer (2001) propose Input Decimation, a method which forms feature subsets taking into account their ability to separate the classes. Each subset is then used to build a classifier. The results indicate that input decimated ensembles outperform ensembles created by training single classifiers on randomly selected subsets of features.

Manipulating the learning algorithm

Another way to achieve diversity in creating ensemble of classifiers is to manipulate the learning algorithm. This could be done by exploiting the algorithm's parameters. For example, ensembles of diverse neural network classifiers can be obtained by using different number of hidden nodes. Some ensembles can gain diversity by injecting randomness into the learning algorithm. For example, when the ensemble is created using the back-propagation algorithm, different initial weights can be assigned randomly to the network. This is the most common way to create ensemble of neural networks, as the classifiers obtained can be quite different, however it is not the most effective method to achieve good diversity. To achieve diversity for decision trees, Dietterich (1998) proposes injecting randomness for the choice of a splitting feature at each internal node. In each step, a feature is selected randomly (with equal probability) among the best 20 features.

Hybrid ensembles

Hybrid ensembles are created by training different types of learning algorithms. Researches done on hybrid ensembles show that the classifiers obtained are different, as

they perform differently in different regions of the space. Wang et al. (2000) show that ensembles created by combining decision trees with neural networks perform well especially when the number of neural networks is larger than the number of decision trees in the ensemble. Langdon et al. (2002) create ensembles of decision trees and neural networks, then use genetic programming to find an appropriate rule to combine the classifiers outputs. Woods et al. (1997) use neural networks, k-nearest neighbors, decision trees, and Quadratic Bayes classifers to form an ensemble of classifiers then classify new instances using the predictions of a classifier that is chosen based on its local performance in the feature space. Hybrid classifier ensembles are advantageous since it is often not evident to apprehend which algorithm will yield the best performance. Each algorithm has a selective superiority, each is best for some but not all tasks and the appropriate algorithm can not be always known in advance.

1.5.2 Measures of diversity

Several measures have been proposed for quantifying diversity for classifier ensembles. Diversity measures are categorized into two types: pairwise and non-pairwise measures.

Pairwise measures

These measures quantify diversity between every pair of classifiers. If we have an ensemble of T classifiers, then $T(T-1)/2$ pairwise diversity values can be calculated. The diversity measure for the ensemble is obtained by averaging all pairwise diversity values. These measures are defined using the relationship table for two classifiers D_i and D_j,

with $a + b + c + d = 1$ and a, b, c and d are fractions of examples that are correctly or incorrectly classified by each or both classifiers. For example, a is the fraction of examples that are correctly classified by both classifiers.

Table 1.1: The relationship table with probabilities

	D_j is correct	D_j is incorrect
D_i is correct	a	b
D_i is incorrect	c	d

Disagreement and double fault measures: The disagreement measure is the simplest measure of diversity between two classifiers. It is defined as the probability that the two classifiers have different decisions:

$$D_{i,j} = b + c. \tag{1.11}$$

The double fault measure is the probability that both classifiers are wrong:

$$DF_{i,j} = d. \tag{1.12}$$

The diversity increases with both the disagreement and the double fault value.

Correlation: The correlation between two classifiers' outputs is formulated as follows

$$\rho_{i,j} = \frac{ad - bc}{\sqrt{(a+b)(c+d)(a+c)(b+d)}}, \tag{1.13}$$

where $0 \leq \rho \leq 1$. An overall measure of diversity is obtained by averaging the correlations between the outputs of every pair of classifiers. $\rho = 0$ means that the two classifiers are uncorrelated.

Q-Statistic: This measure is defined as

$$Q_{i,j} = \frac{(ad - bc)}{(ad + bc)}. \tag{1.14}$$

If the same examples are correctly classified by the two classifiers, then Q will have positive values, and $Q = 0$ means that the two classifiers are completely independent.

Margineantu and Dietterich (1997) use Kappa-statistic which is a diversity measure to compare Adaboost and Bagging. They prove that there is a trade-off between individual accuracy and Kappa measure. They show also that Adaboost produces more diverse ensembles of classiffers than Bagging.

Non-pairwise measures

These measures consider all the ensemble classifiers to obtain one diversity value.

Entropy measure: This measure is based on the assumption that the ensemble is diverse if half of the classifiers predicts correctly and others are wrong. The ensemble can not be considered as diverse if all the classifiers agree in their predictions. Based on this concept, the Entropy measure of diversity is defined as

$$E = \frac{1}{N} \sum_{u=1}^{N} \frac{1}{T - [\frac{T}{2}]} \min\{\zeta_u, (T - \zeta_u)\}, \tag{1.15}$$

where T is the total number of classifiers in the ensemble, ζ_u is the number of classifiers that classify incorrectly instance x_u and N is the cardinality of the dataset. E varies between 0 and 1, 1 indicates maximum diversity and 0 that there is no difference between classifiers.

Kohavi-Wolpert variance: This measure of classifiers diversity is calculated as follows:

$$KW = \frac{1}{NT^2} \sum_{u=1}^{N} \zeta_u (T - \zeta_u). \tag{1.16}$$

KW is equivalent to the disagreement measure of diversity averaged over all the en-

semble classifiers.

Measure of difficulty: This measure proposed by Hansen and Salamon (1990) employs a random variable Z taking values in $\{0/T, 1/T, ..., 1\}$ and defining the fraction of classifiers that correctly classifies an instance x. The variance of the variable Z is then calculated to give the measure of difficulty

$$\theta = Var(Z). \tag{1.17}$$

Hansen and Salamon (1990) show that diversity is higher when θ is small because the classifiers are negatively correlated, meaning that they are more likely to make incorrect classifications of different instances. However, if θ is large, the classifiers are positively correlated and tend to correctly predict the same instances. In this case, the ensemble is not diverse.

In spite of all the measures defined to quantify ensemble diversity, researchers deduced that there is no yet a diversity measure that is constantly associated with the classification error or with higher accuracy. Kuncheva (2004) suggests the Q statistic measure when there is no additional information, because of its simplicity.

1.6 Taxonomies for ensemble methods

Given the success of ensemble techniques, a high number of these methods exists. This has led to the emergence of several taxonomies for categorizing ensemble methods. This section describes the most known approaches for grouping ensembles.

1.6.1 Fusion and selection

Ensemble methods literature distinguishes two main strategies in combining classifiers: fusion and selection. The classifiers' fusion assumes that each member in the group has knowledge of the entire feature space and solve the same original task. The aim here is to get a model that is more robust than the individual classifiers forming the ensemble. Hence, in the fusion strategy, combination rules such as the average and weighted majority vote are employed. However, the classifier selection strategy simplifies a complex problem by dividing it into multiple sub-problems. It assumes that each member in the group is specialized in a partition of the feature space and is concerned by objects in this partition. For this reason the selection approach classifies a new instance by selecting only one classifier. Researchers are interested by classifiers' fusion much more than classifiers' selection. However, this situation may change given that the selection strategy is presumably the best, if correctly trained. The categorization of classifiers combination into fusion or selection strategy has many synonyms in the literature as ensemble approach-modular approach, competitive approach-cooperative approach, multiple topology-hybrid topology. There are combination strategies that are categorized between selection and fusion. The mixture of experts architecture is an example of such a strategy.

1.6.2 Decision optimization and coverage optimization

Methods that are categorized as decision optimization methods concentrate on the choice and the optimization of the combiner for a fixed ensemble of classifiers. Methods categorized as coverage optimization methods are those that fix a combiner than create different base classifiers. Ensembles lying under this categorization can also be referenced as non-generative and generative ensembles.

1.6.3 Trainable and non-trainable ensembles

Non trainable ensembles are those with combiners that do not need training after all ensemble members have been trained. Majority vote is an example of such combiner. However, some other combiners require additional training, like the weighted average combiner. Ensembles like Adaboost are categorized in a third class that develops the combiner in parallel with the training of the individual classifiers.

1.6.4 Top-down and bottom-up ensembles

This categorization indicates if the ensemble is created top-down or bottom-up. Top-down strategy refers to ensembles with a combination mechanism that is based on something else than the classifiers outputs. Ensembles where the classifiers are combined taking into account their outputs are referenced as Bottom-up ensembles. These methods are divided into fixed methods, such as voting, and dynamic methods, such as stacking.

Brown et al. (2005) categorize the ensemble methods depending on the way they achieve diversity, implicitly by randomization methods or explicitly via some metric.

Kuncheva (2004) suggests organizing ensembles according to four elements that characterize the ways these ensembles are created.

1. Data level: this level refers to ensembles that use different subsets of data to train the ensemble classifiers. This approach has shown to be very effective especially for creating Bagging and Boosting ensembles.

2. Feature level: designate ensembles that built classifiers on different feature subsets.

3. Classifier level: this level is associated to ensembles that train different base classifiers on the learning data.

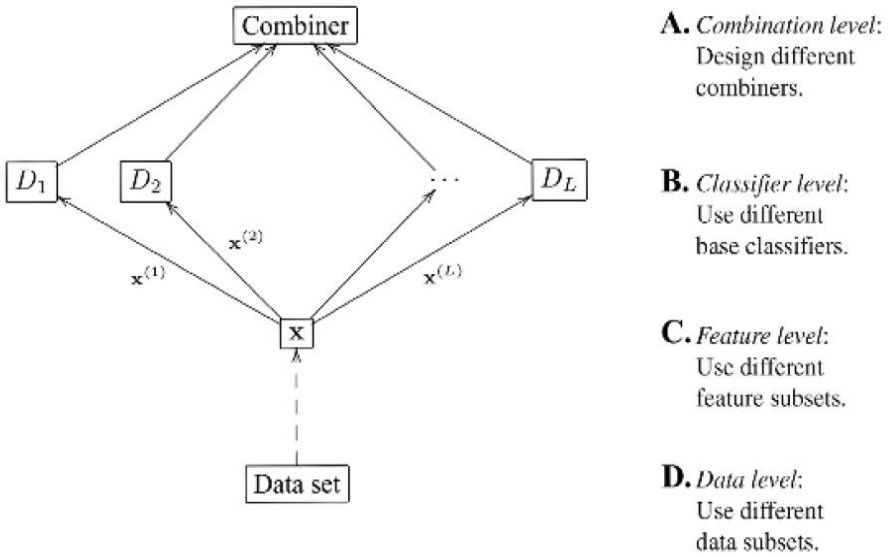

Figure 1.6: Four levels for categorizing ensemble methods.

4. Combination level: This level is invoked when different combination rules are used for fusing classifiers' outputs.

This taxonomy of ensemble methods is illustrated in Figure (1.6) taken from (Kuncheva (2004)).

1.6.5 Dependent and independent ensembles

This category groups ensembles according to whether they are formed by dependent or independent classifiers. In a dependent ensemble, the generation of a certain classifier depends on the outputs of the previous one. Such classifiers can also be described as successive. On the other hand, independent ensembles are constructed by classifiers that are built separately without regard to each other.

Dependent Methods: Dependent approaches take into account the results of the previ-

ous classifier to train the classifier of the next iteration. The information extracted from the outcome of the previous classifiers is used to exploit the training set for the following iteration. These methods usually train classifiers by focusing on instances that were misclassified at the previous step, for example by increasing their weight. Adaboost is a very good example of a dependent ensemble method.

Independent Methods: In this approach, the individual classifiers are usually generated using different subsets of the training data. These subsets may be disjointed or overlapping. To classify a new instance, the classifiers' outputs are aggregated using an appropriate combination rule. Bagging is an example of a method that creates an ensemble of independent classifiers by using samples of data drawn randomly with replacement from the original learning set. Generating an ensemble of independent classifiers improves the performance of the classification and decreases the execution time as this methodology can be easily parallelized.

Categorization of ensemble methods by Jain et al. (2000) is shown in Table (1.2).

1.7 Conclusion

This chapter has provided an overview of ensemble systems. Two fundamental components for ensemble methods have been presented. The first one is the strategy used for constructing diverse ensemble, focusing on the most popular approaches namely Bagging, Boosting and their multiple variations. The second key component is the strategy needed for combining ensemble members, grouping the combination rules into two groups, those applied to class labels and those specific to continuous outputs. This chapter has also shown the importance of diversity for the ensemble success presenting the most used techniques to get diverse ensemble of classifiers, and the measures proposed in the literature for quantifying this diversity. Furthermore, several categorizations of

Table 1.2: Taxonomy for categorizing ensemble methods.

Scheme	Architecture	Trainable	Adaptive	Info-levels	Comments
Voting	Parallel	No	No	Abstract	Assumes independent classifiers
Sum, mean, median	Parallel	No	No	Confidence	Robust, assumes independent confidence estimators
Product, min, max	Parallel	No	No	Confidence	Assumes independent features
Generalized ensemble	Parallel	Yes	No	Confidence	Considers error correlation
Adaptive weighting	Parallel	Yes	Yes	Confidence	Explores local expertise
Stacking	Parallel	Yes	No	Confidence	Good utilization of training data
Borda count	Parallel	Yes	No	Rank	Converts ranks into confidences
Logistic regression	Parallel	Yes	No	Rank confidence	Converts ranks into confidences
Class set reduction	Parallel cascading	Yes / No	No	Rank confidence	Efficient
Dempster-Shafer	Parallel	Yes	No	Rank confidence	Fuses non-probabilistic confidences
Fuzzy integrals	Parallel	Yes	No	Confidence	Fuses non-probabilistic confidences
Mixture of local experts (MLE)	Gated parallel	Yes	Yes	Confidence	Explores local expertise, joint optimization
Hierarchical MLE	Gated parallel hierarchical	Yes	Yes	Confidence	Same as MLE, hierarchical
Associative switch	Parallel	Yes	Yes	Abstract	Same as MLE, but no joint optimization
Bagging	Parallel	Yes	No	Confidence	Needs many comparable classifiers
Boosting	Parallel hierarchical	Yes	No	Abstract	Improves margins, unlikely to overtrain, sensitive to mislabels, needs many comparable classifiers
Random subspace	Parallel	Yes	No	Confidence	Needs many comparable classifiers
Neural tree	Hierarchical	Yes	No	Confidence	Handles large numbers of classes

ensemble techniques have been discussed. The next chapters will focus on the application of ensemble methods on data fusion problems, one of the different domains in which ensemble techniques have shown their effectiveness.

Chapter 2

Ensemble methods for data fusion

2.1 Introduction

Diversity is an important requirement for constructing good ensemble of classifiers. It can be achieved by using many sources of data presenting different related views of the same phenomenon. The interactions and dependencies between the views could be interesting for a better understanding of the problem and for the performance improvement of the classification task. This results in challenging machine learning problems where data sources are combined in order to benefit of the complementary information brought by each source. This process is known by data fusion.

2.2 Data fusion

Data fusion is a formal framework in which are expressed the means and tools for the alliance of data originating from different sources. This approach is based upon the synergy offered by the various sources. The general idea is to increase training data for a given problem with data obtained from other associated learning problems. Learning from multiple sources is useful in situations where the information contained in the sin-

gle sources is partially relevant to the problem in interest. Therefore, it is advantageous to extract complementary information from a large number of data sources. Also, data fusion is profitable when different data sources available for a certain problem are incomplete and noisy. Data fusion is however a vast domain which is not only restricted to machine learning applications.

2.2.1 Definitions and applications

Hall and Llinas (1997) define data fusion as combination of data obtained from multiple sensors and related information from associated databases, in order to improve accuracy and achieve more specific inferences than that could be achieved by the use of single sensor alone. This definition focuses on information quality and fusion methods.

In (Wald (1999)), data fusion is defined as a formal framework where means and tools are expressed for the association of data obtained from different sources aiming at obtaining information of greater quality. This definition highlights fundamentals of data fusion focusing on the quality with a general meaning that can refer to an increase in accuracy of a classification. This quality can also be related to the production of more pertinent information or to a better allocation of human or financial resources for a given project.

The main reason for the development of data fusion is the increasing requirement for more relevant information, more robust procedures to manage data efficiently, and more reliability and performance of the system.

There are different domains where data fusion is applied, due to the increasing availability of multiple different but associated sources of data. Data fusion principles are at the core of diverse modern applications.

Medicine: An example would be combining several medical test results obtained from an MRI scan (an image), an electroencephalogram (a time series), and several blood tests (scalar numbers) for the diagnosis of a neurological disorder. Each medical test generates data with heterogeneous attributes, and data fusion can be applied to get a more accurate diagnosis of the disease.

Remote sensing and geographic data: In combination of multisource remote sensing and geographic data for classification purposes, different types of information from several data sources, e.g., landsat thematic mapper data, radar data, elevation data, and slope data, are used in order to improve the classification accuracy as compared to the accuracy achieved by single-source classification.

We should mention other areas such as bioinformatics, weather forecasting, robot navigation and land mine detection.

2.2.2 Data fusion levels

Data fusion processes are categorized into three main levels, low, intermediate and high, depending on the stage at which fusion is performed.

Low level fusion (raw or data fusion): Is the combination of raw data from multiple sources into new raw data that should be more informative and synthetic. This fusion level is illustrated in Figure (2.1).

Intermediate level fusion (feature fusion): This fusion level combines different data sources at the intermediate level which requires the extraction of different features from the sources of raw data to be aggregated into a composite feature. Note that a feature is a property of an object, it is equivalent to attribute. Figure (2.2) illustrates this fusion level.

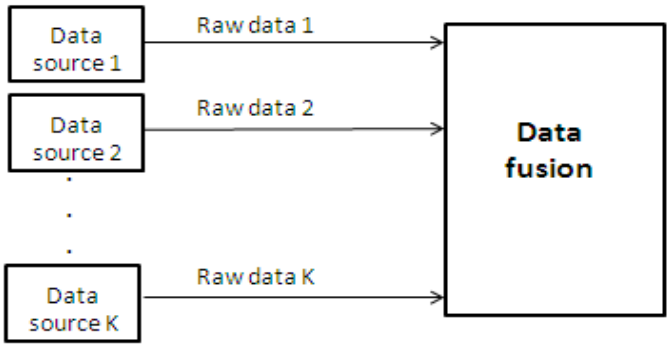

Figure 2.1: Illustration of the raw fusion process.

High level fusion (decision fusion): As shown in Figure (2.3), decision fusion process combines results from multiple sources or algorithms to yield a final fused decision. Several methods of decision fusion exist, such as statistical methods, voting methods, ensemble methods, etc.

2.2.3 Trade-offs for different levels

Some problems are to be solved prior to the fusion process. The most important is the choice of the most appropriate fusion level for the data available. To choose among the different levels we must know strengths and limitations of each one.

Data level fusion is intended to realize the highest fusion and classification performance as all the information collected from different sources is conserved. However, the raw fusion level has many limitations. First, it requires conversion of the data into a common frame before concatenation. This is called the alignment problem since it is difficult to provide a general frame for heterogeneous data. The alignment problem supports that data fusion is more complex than data concatenation. Data concatenation

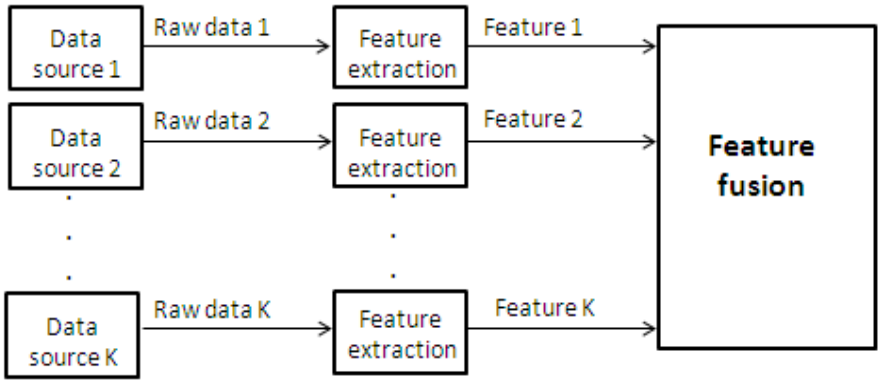

Figure 2.2: Illustration of the feature fusion process.

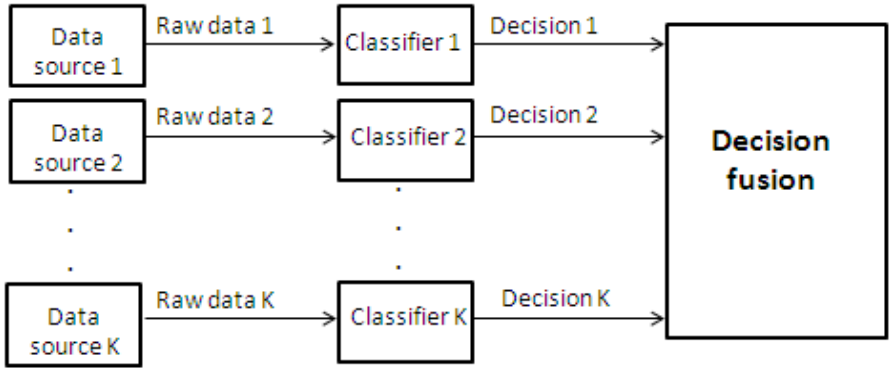

Figure 2.3: Illustration of the decision fusion process.

is performed simply by juxtaposing all the data. These data should be homogeneous. Also, data level fusion is a centralized processing. It requires high bandwidth and data storage, accurate data registration and increased learning time. This low level suffers from another major problem which is the elimination of data variation. It is important to maintain the independence of the multiple sources to be associated, as well as allow an easy integration of the information they contain.

Feature level fusion requires lower bandwidth and data storage, and less strict data and features registration. However, the reduction of the original data to a small set of features could affect the performance of the model.

Decision level fusion is the simplest fusion approach. Data from each source is reduced down to a binary value. Certainly, this reduction lowers fusion performance, but it greatly increases the efficiency and reliability of fusion. Decision level fusion is also advantageous because of its feasibility, lower computational complexity and robustness to the removal or addition of individual data sources. It is able to generate a solution given a set of features, a learning algorithm and a combination technique. The decision level deals with:

- Multiple sources of data
- Multiple heterogeneous features
- Multiple learning algorithms
- Multiple combination techniques

The main characteristics of the three levels of data fusion are summarized in Table (2.1).

As a conclusion from the different levels' characteristics, it is possible to say that the decision fusion level is the most appropriate for multiple sources data merging. Fusion

Table 2.1: Tradeoffs for different levels of data fusion.

	data	**feature**	**decision**
Bandwidth	possibly very large	medium	very small
Information loss	no loss	some	medium
Performance loss	no loss	some	some
Operational complexity	high	medium	low

at the raw data level allows for better performance. However, data fusion at this level increases the noise and the complexity of the data set. In this case, the learning algorithm will tend to overfit the data rather than to capture the underlying data trends. At the other hand, fusion at the decision level is more robust to noise. Decision fusion has the ability to capture general trends in data while remaining robust to noise effects.

2.3 Techniques for data fusion

Several approaches have been developed for data fusion. In multi-source or more specifically multi-sensor systems, the objective is to extract information from several available sources using the most appropriate techniques. However, the data or signals obtained are often corrupted by noise and uncertainty. In order to deal with these problems, methods from the probability theory like Bayesian inference, Kalman filtering, but also other techniques like Dempster-Shafer evidence theory, Fuzzy logic and Neural networks, have been widely used for data fusion. The basic principles of some of these techniques will be presented.

2.3.1 Bayesian inference

Bayesian inference allows information obtained from multiple sources to be combined using the rules of probability theory. This technique is based on statistical inference

which relies on collecting evidence to adjust the belief on a given hypothesis. New information is used to update the a priori probability of a hypothesis. This process is called Bayesian inference because it uses Bayes' theorem for the probabilities update. Given new evidence E, a set of mutually exclusive and exhaustive hypotheses, $\{H_i, i = 1...T\}$, the Bayesian inference process is as follows

$$P(H_i/E) = \frac{P(E/H_i)P(H_i)}{P(E)}, \tag{2.1}$$

where $P(H_i/E)$ is the posterior probability of H_i given the evidence E, $P(H_i)$ is the prior probability of H_i inferred before the new evidence E became available, $P(E/H_i)$ is the conditional probability of observing the evidence E given that H_i is true, and $P(E)$ is the marginal probability of E or the prior probability of observing the new evidence E under all possible hypotheses. The Bayesian posterior probability reflects the belief in the hypothesis based on the prior information and current observations. Bayesian methods have been widely used in probabilistic image fusion. Williams (2009) uses a Bayesian data fusion approach for seabed classification using multiview synthetic aperture sonar imagery. The proposed approach exploits all available information and gives probabilistic predictions.

However, Bayesian inference has some limits. It requires the knowledge of a priori and conditional probabilities that are very rarely known. Thus, the implementation of probabilistic methods often relies on simplified assumptions, for instance events independency, or arbitrary choices of Gaussian or uniform distributions.

2.3.2 Dempster-Shafer theory of evidence (DS)

DS theory of evidence proposed by Dempster (1967); Shafer (1976) is also known as the theory of belief functions. This technique is a generalization of the Bayesian infer-

ence as it allows for distributing support not only to single hypothesis but also to the union of hypotheses. It allows presenting and combining measures of evidence for a more general interpretation of uncertainty and even quantification of ignorance. In this theory, the belief and the uncertainty concerning a fact could be formulated independently. The Bayesian inference and DS approaches are both based on the concept of assigning weights to the states of the problem of interest. But, while the weights in Bayesian theory are probabilities, DS theory deals with measures of belief based on the idea of ”mass” as opposed to probability. This theory assigns its masses to all entities that comprise a system. DS method gives a rule for calculating the confidence measure of each state, based on data from both new and old evidence.

In DS evidence theory, all hypotheses that are mutually exclusive are collectively called ”The frame of discernment”, $\Theta = \{\theta_1, ..., \theta_T\}$, and 2^Θ presents all subsets of Θ.
$m : 2^\Theta \rightarrow [0, 1]$ is an elementary mass function which is called a basic probability assignment (bpa) if

$$m(\emptyset) = 0, \sum_{A \subseteq \Theta} m(A) = 1. \tag{2.2}$$

In DS theory, belief is not only associated to an atomic hypothesis, but also to some set $A = \{a_1, ..., a_n\} \subseteq \Theta$.
For each element B, with $m(B) \geq 0$, the belief function $bel : 2^\Theta \rightarrow [0, 1]$ is defined as:

$$bel(B) = \sum_{A \subseteq B} m(A), \tag{2.3}$$

where B is called a focal element, and the belief in B presents the amount of evidence which implies the observation of B based on all elements A that support B. Note that if A is an atomic hypothesis, $bel(A) = m(A)$.
Bayesian inference and DS produce identical results when all the hypotheses are singletons and mutually exclusive.

DS evidence theory allows the formulation of both imprecision and uncertainty and has shown to be effective for data fusion applications. Gong (2009) proposes a data fusion approach using DS evidence theory to overcome the low accurate identification. It gives an alternative to situations where the general ways of data fusion can not identify the multiple sensors uncertainty.

However, in DS evidence theory, the definition of mass functions is a crucial step and it is the most difficult part in the implementation of DS evidence theory.

2.3.3 Neural networks

The advantage of neural network classifiers over supervised statistical approaches is that they need no prior statistical information about the input data. This is especially important for multisource data, since the whole multiple source dataset is usually very difficult to model by statistical methods.

Neural networks are learning algorithms inspired by the architecture and the behavior of the brain as a network of units called neurons. Basically, an artificial neural network is built from simple elements called neurons or nodes, which are linked by weighted connections called synapses. These neurons are always grouped into several layers. The information is given to the neurons of the first layer and then propagated to the neurons of the second. The neurons are considered as computational units. Each of these units receives an input information and transforms it into an output response. This transformation involves two steps: First, the activation of the neuron is computed as the weighted sum of its inputs, and second this activation is transformed into a response by using a transfer function. The objective of the neural network is to transform the inputs into meaningful outputs.

Neural networks approach exhibits properties analogous to adaptive biological learn-

ing. It has good pattern recognition capabilities, and once learned, information remain resistant to hardware or data failure. It has some advantages over statistical approaches, it is distribution-free and the degree of belief in each data source is represented by the weights of the network and determined by the training process.

2.3.4 Kernel methods

Some researchers use kernel methods to deal with data fusion problems. These methods provide a framework for representing many different types of data structures, such as vectors, strings, and graphs, etc.

A generic framework is proposed by Biessmann et al. (2010) for the analysis of simultaneously acquired signals of different dimensionalities and temporal resolutions. Their algorithm is based on Kernel Canonical Correlation Analysis.

Sa et al. (2010) use a kernel combination method to aggregate multiple data sources for a clustering problem. The kernels are induced by a graph constructed by exploiting co-occurrences of patterns among the different sources.

Szafranski et al. (2010) develop the composite kernel learning approach dedicated to learning the kernel when there is a known group structure among a set of candidate kernels. This method applies to single data source problems and it is also particularly appropriate to multiple sources problems.

2.4 Ensemble methods for data fusion

Data fusion is a natural fit for ensemble systems, since different classifiers can be generated using data obtained from different sources, and then combined to achieve the desired data fusion. Each source of data captures different characteristics of the ob-

jects to be classified and the ensemble obtained may give better prediction performances through the diversity and independence of the classifiers' decisions.

Several ensemble approaches have been proposed for this purpose. These approaches include simple combination techniques such as majority vote, simple average and maximum/minimum rules. More complex data fusion techniques are also commonly used such as DS, decision templates and stacked generalization.

Briem et al. (2002) experiment Bagging, Boosting, and consensus-theoretic classifiers for the classification of multisource remote sensing and geographic data. Several base classifiers are employed such as the conjugate-gradient back-propagation, decision table and C4.5 decision tree.

Wang et al. (2006) present a classifier fusion approach for poorly-differentiated tumor classification using both mRNA and miRNA expression profiles. They use Bagging with fuzzy k-nearest-neighbors as base classifier. Outputs from classifiers are combined by a simple average.

The performance of several ensembles of SVM classifiers in gene function prediction is tested in (Re and Valentini (2009)). Classifiers are trained on different types of data, and then combined using different aggregation techniques like linear weighted combination, the logarithmic weighted combination and the similarity based decision templates approach.

Another ensemble approach having shown its effectiveness in data fusion applications is Learn++ algorithm. This algorithm was originally developed for incremental learning of novel information from new data and adopted for data fusion (Polikar et al. (2008)).

This algorithm is used for several data fusion applications like Nondestructive evalua-

tion of pipelines database which is concerned with detection and identification of flaws in various types of materials.
Learn++ algorithm is also employed in medical applications, like early diagnosis of Alzheimer's disease. Data is collected from different sources, namely different channels of electroencephalogram (EEG) and different frequency bands. Event related potentials recorded from different electrodes of the EEG signals are decomposed into different frequency bands using multi-resolution wavelet analysis, to form multiple datasets. The algorithm allows the identification of the most discriminating information from each data source, for an improved performance in the diagnosis of Alzheimer's disease.
Learn++ algorithm is described and evaluated in the following section.

2.5 Evaluation of Learn++ algorithm

Learn++ is based on the well known Adaboost algorithm. As Adaboost, it generates an ensemble of classifiers iteratively, each one is trained on an updated distribution of the training data. Adaboost trains the next base learner by increasing the weights of data points which are wrongly classified by the previously generated classifier. However, Learn++ updates the data distribution based on the existing ensemble performance, not just the previous classifier. Another difference is that Adaboost aims to improve the performance of a classifier for a given dataset, while Learn++ deals with multiple datasets by constructing an ensemble for each dataset and then combining all the ensembles to obtain a meta-ensemble of classifiers.

In the context of data fusion, we have K sources, each introducing a new dataset DS_k. Iteratively, a weak classifier is trained with a training subset obtained from DS_k. This classifier can be any supervised algorithm performing at least 50% on its training dataset, so that a minimum performance can be expected. A hypothesis is obtained and

assigned a weight proportional to its error. In a first combination phase, all hypotheses generated thus far are fused using the weighted majority voting. The composite hypothesis obtained represents the current ensemble decision for the given feature set. For the data fusion process, the weights of all hypotheses that have been generated from data sources are adjusted based on observed training performance on each data source. Then, a final hypothesis is obtained by combining all hypotheses through weighted majority vote, and this is the second phase where a combination rule is employed.
The inputs to this algorithm are :

- K feature sets DS_k, $(k = 1, ..., K)$ obtained from K sources.
- A training data $S_k = [(x_i, y_i)]$, $(i = 1, ..., m_k)$ composed of m_K instances with their associated labels.
- A supervised learning algorithm to generate the individual classifiers.
- An integer T_k, specifying the number of classifiers for dataset S_k.

The different steps of Learn++ are given in Algorithm 3.

An ensemble of classifiers is obtained by an iterative process. Learn ++ starts by setting a uniform distribution so that all the training instances have equal probability to be selected into training subset TR_1. At the tth iteration, the distribution D_t is updated by normalizing weights w_t assigned to instances of the training set S_k as follows

$$D_t = \frac{w_t}{\sum_{i=1}^{m_k} w_t(i)}. \tag{2.4}$$

According to this distribution, a learning subset TR_t is selected from the entire dataset in order to train the base classifier. A hypothesis h_t^k is obtained and its error

Algorithm 3 Learn++

Require: DS_k, S_k, T_k.

Ensure: $H_{final}(x)$

1: **for** $k = 1$ to $k = K$ **do**

2: Initialize $w_1(i) = D_1(i) = \frac{1}{m_k} \forall i, i = 1, 2 \ldots m_k$

3: **for** $t = 1$ to $t = T_k$ **do**

4: Set $D_t = \frac{w_t}{\sum_{i=1}^{m_k} w_t(i)}$ so that D_t is a distribution

5: Draw training TR_t subset from D_t

6: Obtain h_t^k by training with data TR_t and calculate its error on S_k
$\varepsilon_t^k = \sum_{i:h_t^k(x_i)\neq y_i} D_t(i)$. If $\varepsilon_t^k > \frac{1}{2}$, discard h_t^k and $\rightarrow$ step 5

7: Set $\beta_t^k = \frac{\varepsilon_t^k}{(1-\varepsilon_t^k)}$ and obtain the composite hypothesis
$H_t^k = \arg\max_{y\in\Omega} \sum_{t:h_t^k(x)=y} \log(\frac{1}{\beta_t^k})$

8: Compute error of H_t^k: $E_t^k = \sum_{i:H_t^k(x_i)\neq y_i} D_t(i)$

9: Set $B_t^k = \frac{E_t^k}{(1-E_t^k)}$, and update the weights
$$w_{t+1}(i) = w_t(i) \times \left\{ \begin{array}{ll} B_t^k, & \text{if } H_t^k(x_i) = y_i \\ 1, & \text{otherwise} \end{array} \right\}$$

10: **end for**

11: Compute voting weights adjustment factor : $\alpha_k = \frac{(\sum_{i=1}^{m_k} [|H_{T_k}^k(x_i)\neq y_i|])}{m_k}$

12: **end for**

13: Output the final hypothesis: $H_{final}(x) = \arg\max_{y\in\Omega} \sum_{k=1}^{K} \frac{1}{\alpha_k} (\sum_{t:h_t^k(x)=y} \log(\frac{1}{\beta_t^k}))$.

ε_t^k given by,

$$\varepsilon_t^k = \sum_{i:h_t^k(x_i)\neq y_i} D_t(i), \tag{2.5}$$

is calculated on the dataset S_k as the sum of the distribution weights associated to misclassified instances.

The hypothesis h_t^k is accepted only if its error is less than $1/2$. In this case, the error is normalized to give β_t^k which is used to formulate a voting weight for the hypothesis h_t^k. In order to obtain the composite hypothesis H_t^k of the current iteration defined by,

$$H_t^k = \arg\max_{y\in\Omega} \sum_{t:h_t^k(x)=y} \log(\frac{1}{\beta_t^k}), \tag{2.6}$$

WMV is employed as a combination rule for all generated hypotheses.

Hypotheses with smaller training error have more influence in the final classification decision, since they are assigned a higher voting weight. The error of the composite hypothesis H_t^k is calculated by summing the distribution weights of instances that are incorrectly classified by the current ensemble decision as follows

$$E_t^k = \sum_{i:H_t^k(x_i)\neq y_i} D_t(i). \tag{2.7}$$

Since all hypotheses composing H_t^k have individual errors less than $1/2$, then the composite error E_t^k will also be less than $1/2$. Then, this error is normalized giving B_t^k, used for updating the training set distribution for the next step as follows

$$w_{t+1}(i) = w_t(i) \times \left\{ \begin{array}{ll} B_t^k, & \text{if } H_t^k(x_i) = y_i \\ 1, & \text{otherwise} \end{array} \right\}. \tag{2.8}$$

The distribution update is done in a way that highlights instances incorrectly classi-

fied by the current ensemble, since the distribution weights w_t of the correct instances are reduced by a factor of B_t^k. Thus, the learning algorithm will focus on the misclassified instances as they are more likely to be selected into the training subset of the next iteration. The process described is repeated until T_k classifiers are generated for the dataset S_k. Once this is done, the weighted performance associated to each classifier, $\log(\frac{1}{\beta_t^k})$, is adjusted by multiplying it by a reliability factor which represents the observed training performance on the feature set S_k and more generally the reliability of the data source in interest. This adjustment factor is given by

$$\alpha_k = \frac{(\sum_{i=1}^{m_k}[\mid H_{T_k}^k(x_i) \neq y_i \mid])}{m_k}, \tag{2.9}$$

where $H_{T_k}^k$ is the composite hypothesis obtained in the final step from the training set S_k of data source DS_k. This reliability factor may also be based on prior information indicating if a certain data source is more reliable than another. Hence, a higher voting weight is assigned to classifiers trained with such data. When the construction of ensembles for all feature sets is achieved, the final hypothesis H_{final} is obtained by combining all hypotheses generated from all K data sources. WMV is employed for this purpose using the adjusted weights of all the hypotheses. H_{final} is calculated as follows

$$H_{final}(x) = \arg\max_{y \in \Omega} \sum_{k=1}^{K} \frac{1}{\alpha_k} (\sum_{t:h_t^k(x)=y} \log(\frac{1}{\beta_t^k})). \tag{2.10}$$

There are similarities between Learn++ and Adaboost algorithm. Both generate an ensemble of classifiers in order to improve the generalization performance of a weak learner, by using an iterative process, where classifiers are trained by updating the distribution of the training data. However, unlike Adaboost which uses a single dataset, Learn++ learns from multiple feature sets. Adaboost focuses on instances that have been misclassified by the previous hypothesis h_t, while Learn++ relies on the performance of the composite hypothesis H_t. Learn++ not only focuses on misclassified data,

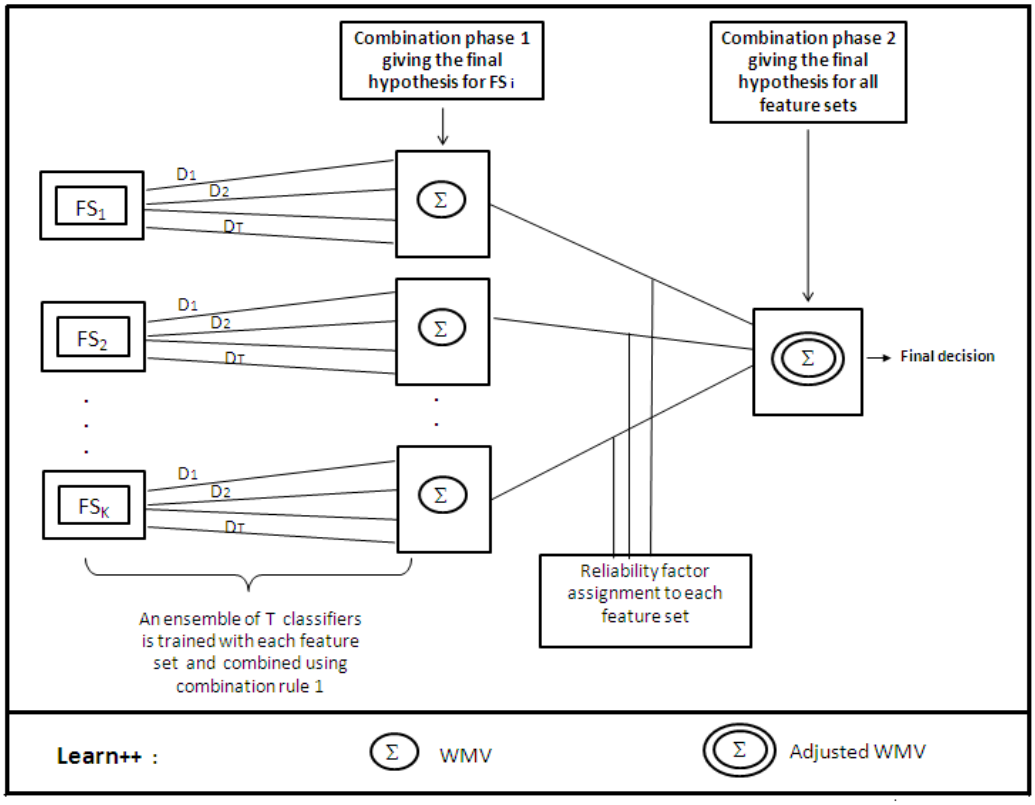

Figure 2.4: Schematic representation of Learn++ algorithm.

but also on instances not yet learned by the ensemble, those carrying novel information. Figure (2.4) illustrates the overall algorithm as structured for data fusion applications.

In order to evaluate Learn++ algorithm for data fusion applications, two benchmark databases obtained from the UCI Repository of Machine Learning are used. One of them, the multiple feature database, which deals with handwritten digit recognition task, is naturally in data fusion format, allowing a true real world data fusion setting. The second, Wine database, is randomly partitioned into subsets, where each partition uses only a portion of the features to simulate a data fusion setting.

Multilayer perceptrons (MLP) and decision trees (DT) are used as base classifiers and T

is the number of classifiers trained. The databases and the learning algorithms used are described in Chapter 3. Results are compared with Adaboost.M1 algorithm.

Table 2.2: Fusion performance obtained by Learn++ with MLP and DT.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	**90.53**	90.49	75.71	81.71
FS2	58.71	71.31	71.29	72.00
FS3	84.12	86.46	72.53	74.53
FS4	76.03	84.62	76.59	80.47
FS5	68.35	72.93	64.29	67.43
FS6	72.53	72.49	68.71	69.58
Fusion	94.41	**94.43**	93.88	**94.01**

We notice that for most of the single feature sets, the individual performance increases as the number of classifiers increases. Both with MLP and DT as base learners, the classification performance obtained by fusing the feature sets is better than that of any individual feature set. We can notice that the best fusion result, 94.43%, is obtained when using MLP as base learner, training ten classifiers for each feature set. The improvement is about 4% over the performance of the best single dataset, that is 90.53% obtained by the first feature set.

DT ensembles give similar fusion performance, 94.01%, to MLP ensembles, also with ten classifiers for each feature set. However, 94.01% significantly exceeds 81.71%, which is the best individual performance obtained by the first feature set when training with DT. This shows the advantage of extracting complementary information from the different feature sets. The interaction between the single datasets is the main reason for the performance improvement, even when these datasets give not excellent individual classification results.

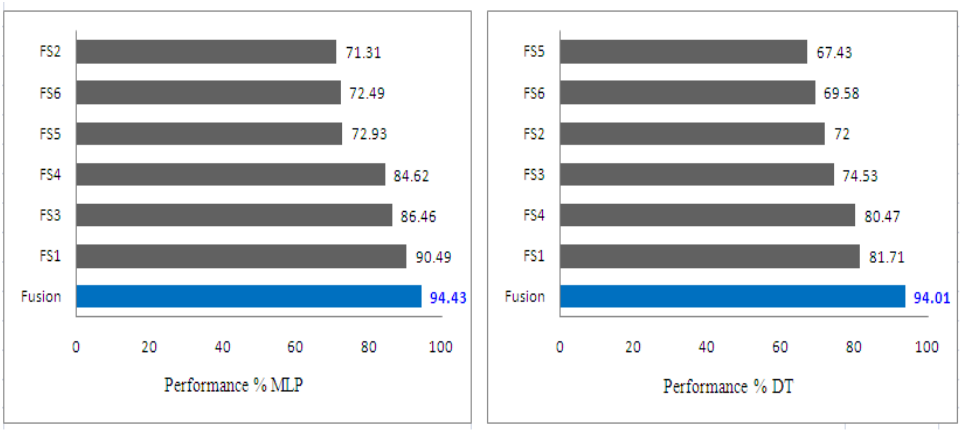

Figure 2.5: Fusion performance using Learn++ with T=10.

Diagrams in Figure (2.5) present histograms with the individual and fusion performances achieved by MLP and DT with Learn++ algorithm.

Adaboost algorithm for data fusion is also evaluated on the multiple feature dataset giving good fusion results as it is shown in Table (2.3).

Table 2.3: Fusion performance obtained by Adaboost.M1 with MLP and DT.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	90.21	**90.53**	78.22	80.35
FS2	69.85	71.47	70.03	70.18
FS3	86.33	86.71	71.65	73.29
FS4	84.12	85.00	77.43	81.62
FS5	73.97	74.06	70.84	71.15
FS6	72.48	73.94	67.45	69.89
Adaboost.M1	96.55	**96.83**	96.16	96.09

As for Learn++, Adaboost gives high classification accuracy when all the feature

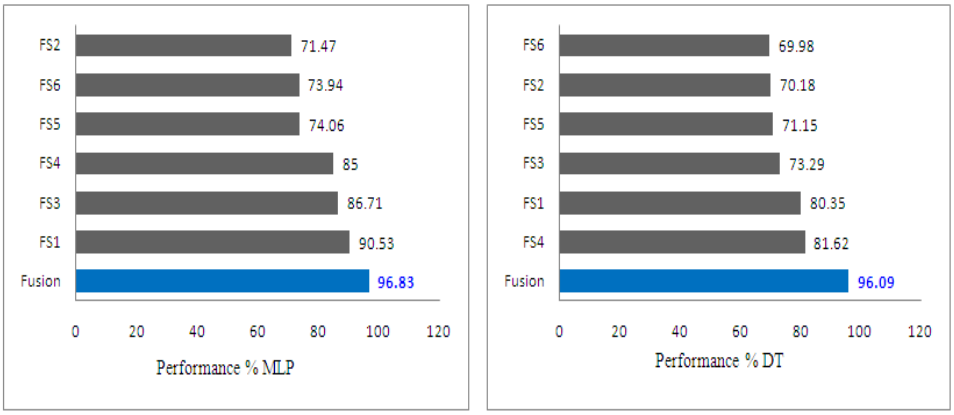

Figure 2.6: Fusion performance using Adaboost.M1 with T=10.

sets are combined, exceeding the best individual classification performance. With MLP as base learner, the best individual performance, 90.53%, is achieved by the first feature set using 10 classifiers.
Individual and fusion results of MLP and DT with Adaboost.M1 algorithm are illustrated in the histograms of Figure (2.6).

Compared to Learn++, the best fusion performance (96.83%) is achieved by Adaboost.M1 algorithm when training 10 successive MLP classifiers for each feature set. Also, for both algorithms individual and fusion performances obtained by training with MLP are better than those obtained by training with DT. As for Learn++, Adaboost.M1 results show the interest of merging complementary data to improve the overall classification performance.

Learn++ and Ababoost.M1 algorithms are also evaluated for Wine recognition database. This database is randomly partitioned into three subsets, where each partition uses only a portion of the features to simulate a data fusion setting. Results are given in Tables (2.4) and (2.5).

Table 2.4: Fusion performance obtained by Learn++ with MLP and DT on Wine dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	78.38	81.62	**89.05**	86.89
FS2	63.11	67.57	73.78	74.33
FS3	74.73	70.13	67.97	68.10
Fusion	77.45	73.25	81.81	70.40

According to results provided in Table (2.4), Learn++ shows a decrease in classification performance compared to that of the best single feature set, and this performance decreases even more as the number of classifiers for each feature set increases. This is true for both MLP and DT ensembles.

Table 2.5: Fusion performance obtained by Adaboost.M1 with MLP and DT on Wine dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	83.78	83.78	87.43	**87.55**
FS2	68.11	67.03	77.02	74.06
FS3	74.59	75.27	71.35	69.73
Adaboost.M1	83.98	65.94	69.46	52.43

The same behavior as Learn++ concerning the poor fusion performance is noticed when using Adaboost algorithm for the wine database.This is contradictory to the previous deduction supporting that the fusion of several feature sets for the same classification problem leads to better accuracy than that of the best individual feature set. The reason for the failure of the fusion process may be due to an inappropriate method for merging the multiple datasets' outputs.

Both Learn++ and Adaboost.M1 use WMV as a combination rule in order to get the final hypothesis for a single dataset, and also to get the final classification decision by fusing all the outputs that the different feature sets give. Resulting fusion performance, observed on Wine database, induces to think that WMV is not a robust technique insuring good fusion performance for all data fusion applications.

Given the advantages that fusing several data sources underlying the same classification problem could present for numerous important domains, there is a need to search for more robust aggregation rules allowing to achieve the desired data fusion and so getting better classification performance.

2.6 Conclusion

This chapter discussed the application of ensemble methods to data fusion problems. Different definitions of data fusion process have been presented, focusing on information quality and fusion techniques. Many domains of application like medicine and remote sensing have been cited, and different levels of combining data from multiple sources have been described. Decision fusion level can be considered appropriate for most data fusion applications. This chapter has also presented several techniques employed for this purpose, like Bayesian inference, DS theory of evidence and others. The use of ensemble methods shows that ensemble structure provides a natural mechanism to combine multiple heterogeneous sources of data. Several applications of ensemble approaches for data fusion have been reviewed. The final part of this chapter focused on the evaluation of some ensemble techniques for data fusion applications. We showed the importance of the choice of a robust combination rule in order to achieve better classification performance when fusing multiple sources of data. Outliers are responsible for the overall performance deterioration in some data fusion applications. In the next

chapter, robust techniques are proposed for combining classifiers, to reduce the effect of outlier classifiers in the ensemble.

Chapter 3

Data fusion by robust classifiers combination

3.1 Introduction

Fusion can be applied to various sources of knowledge but also to multiple classifiers. Many different approaches have been reviewed for classifiers' decisions fusion, including min/max rule, Bayesian fusion, majority vote, models coming from incertainty reasoning such as DS theory, etc. WMV is one of these approaches that is used in Adaboost and Learn++ algorithms in data fusion context. In a first phase, WMV is used to combine classifiers generated iteratively for each data source. Then, it is employed to get the final hypothesis after the data fusion process.

The idea of WMV is to assign a weight w_i to each classifier proportionally to its estimated performance. Given an instance x to classify by H hypotheses into one of C

classes, the decision of a hypothesis h_i on class ω_j is represented in the following way

$$d_{i,j} = \begin{cases} 1 & \text{if } h_i \text{ labels } x \text{ in } \omega_j \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The classifiers' decisions are combined through WMV leading to the choice of class ω_a if we have

$$\sum_{i=1}^{H} w_i d_{i,a} = \max_{j=1}^{C} \sum_{i=1}^{H} w_i d_{i,j} \tag{3.2}$$

The selected class is the one receiving the largest total weight. In WMV, the final decision is influenced by classifiers having a high estimated performance. So, the quality of the classifier is important in this group decision making process, and it depends only on the model's own estimated performance. But what if the classifier having the largest weight turns out to be unreliable?

The experiments done in the previous chapter in order to evaluate ensemble methods for data fusion have shown that WMV is not always effective. Therefore, a need to search for more robust aggregation rules allowing to achieve the desired data fusion and obtain better classification performance has been expressed.

We propose to use a robust combination rule for aggregating classifiers, aimed at reducing the effect of outlier classifiers in the ensemble. The proposed solution takes into account the conflict level of a classifier with the other classifiers in the group. This level is used to determine the reliability associated to the classifier. The detection and resolution of the conflicts between the models leads to a reduction of the influence of outlier models in the final decision making.

3.2 The robust aggregation rule

This technique aims at reducing the influence of conflicting classifiers in the ensemble (Garcia and Puig (2004)). The opinions given by an ensemble of classifiers are represented as probabilities. As for WMV, these opinions are associated with a confidence level presenting the classifier's belief on its own decision. The robust approach determines the conflict level of each classifier by measuring the similarity between its opinion and confidence, and those of the other classifiers in the ensemble. Based on those conflict levels, a reliability rate is associated to each classifier, such as a reliable classifier is the one which is confident and non-conflicting at the same time. The final decision is obtained by multiplying these reliability factors by the original classifier opinions. The process involves two steps. The first one is the training stage where an ensemble of classifiers is trained on the learning dataset, the second one is the conflict resolution and decision making.

3.2.1 Training stage

Let $O_j = \{o_{tj}, t = 1 \dots T\}$ be the opinions of an ensemble of T classifiers, regarding a set of J classes, $\Omega = \{\omega_j, j = 1 \dots J\}$, corresponding to a classification problem. A confidence level w_{tj} is assigned to each classifier about each opinion o_{tj} it expresses. The confidence is a weight based on the Kullback J-divergence (KJ) (Kittler (1986)) measuring the separability between two classes ω_a and ω_b as follows

$$KJ_t(\omega_a, \omega_b) = \int_0^1 (A - B) \log(\frac{A}{B}) du, \tag{3.3}$$

where A and B are obtained from classifier's opinions regarding respectively the two classes ω_a and ω_b. This method measures the classifier's confidence. A classifier with a low KJ measure will have a low confidence, as it separates slightly the different classes.

A classifier with a high KJ differentiates properly among the different classes. Such classifier will have a high confidence. These confidences are computed as the normalized average of the KJ between ω_j and the other classes. This is a possible way to define expert's confidence.

Another confidence formulation can be applied, the same as that used for the WMV as the weighted performance of each classifier, the weights are given by

$$w_{tj} = \log(\frac{1}{\beta_t}), \tag{3.4}$$

with β_t being the normalized error of the tth classifier in the ensemble for class ω_j.

3.2.2 Conflict resolution and decision making

Given $O_j(x)$ the opinions of T classifiers about the belonging of an instance x to the class ω_j, and given $W_j = \{w_{tj}, t = 1 \dots T\}$, the confidences associated with those opinions, the conflict of each classifier is formulated by first measuring the similarity between its opinions, and those of the other classifiers in the ensemble, as follows

$$Sim_t(O_j(x)) = Sim(o_{tj}, O_j \backslash \{o_{tj}\}) = 1 - \frac{1}{(T-1)} \sum_{k=1, k \neq t}^{T} \mid o_{tj}(x) - o_{kj}(x) \mid . \tag{3.5}$$

Then, the expert's confidence similarity with the rest of the confidences, $Sim_t(W_j)$, is calculated the same way. Based on these calculations, the conflict raised by a classifier is defined as

$$Conflict_{tj}(x) = Sim_t(W_j)[1 - Sim_t(O_j(x))]. \tag{3.6}$$

Conflicting classifiers are those with similar confidences to the agreeing classifiers but completely different opinions from theirs. The conflict measure will affect classi-

fier's reliability which is calculated as follows

$$r_{tj}(x) = w_{tj}(1 - Conflict_{tj}(x)). \tag{3.7}$$

Finally, the original opinions of the experts are adjusted by multiplying them by the associated reliability factors after being normalized. The selected class is the one having the maximum adjusted opinion. We can also get the final decision by using the maximum posterior probability, obtained by applying the Bayes rule.

Given that for each classifier two possible confidence formulations are possible, there are two versions of the robust combination method: robust combination based KJ divergence confidence denoted by RKJ, and robust combination based $\log(\frac{1}{\beta})$ confidence denoted by RL.

3.3 Proposed algorithms

In Learn++ ensemble algorithm described in Chapter 2, there are two phases where a combination rule is employed. The first phase is when generating, for a single dataset, a composite hypothesis in each iteration in order to update the training set distribution based on the current ensemble decision. The second phase is when combining the ensembles that have been generated for all the feature sets in order to obtain the final hypothesis. On both phases, the algorithm employs the WMV. Here, we replace this technique by the robust combination method in each of the two phases alternately and in both phases together to have another version of the algorithm based only robust combination.

In total, Learn++ algorithm and five of its variations will be evaluated. According to the combination technique used for each phase, these variations are referred as shown on Table (3.1).

Table 3.1: Notation of the different variations proposed for Learn++.

	Combination rule	
Notation	phase 1	phase 2
Learn++	WMV	WMV
Learn.RKJ1	WMV	RKJ
Learn.RL1	WMV	RL
Learn.WMV	RKJ	WMV
Learn.RKJ2	RKJ	RKJ
Learn.RL2	RKJ	RL

The robust aggregation rule is evaluated on another ensemble method for data fusion, namely Adaboost.M1 algorithm. Robust aggregation rule is used in the data fusion process to integrate the classifier ensembles of all feature sets. Two variations of Adaboost.M1, namely Adaboost.RKJ and Adaboost.RL, which use respectively RKJ and RL as combination techniques, are obtained.

In order to compare the proposed variations to the original algorithms, experiments are run on five benchmark databases from the UCI Machine Learning Repository. Two of them, namely Multiple feature and Wine databases, were used in chapter 2.

3.4 Experimental setup

3.4.1 Databases

Multiple features dataset

This database is obtained from the UCI Machine Learning Repository. It contains 649 features for 2000 handwritten digits ('0'-'9'). The target class has 10 states, each one has 200 samples. Three hundred out of the 2000 instances are used for training, and the rest for testing. The dataset is presented by 6 feature sets:

- Feature Set 1 (FS1): Profile correlations - 216 features
- Feature Set 2 (FS2): Fourier coefficients - 76 features
- Feature Set 3 (FS3): Karhunen Love coefficients - 64 features
- Feature Set 4 (FS4): Pixel averages - 240 features
- Feature Set 5 (FS5): Zernike moments - 47 features
- Feature Set 6 (FS6): Morphological features - 6 features

The multiplicity of feature sets specific to data fusion problems, in addition to the relatively high dimensionality of some datasets, increase the execution time. In our approach, to select most informative features from each of the large datasets, we apply a feature selection algorithm to the training data.

Feature selection

In order to decrease the execution time for some of the feature sets containing a relatively high number of features, only 40 relevant features are selected for the first four feature sets, according to the minimal-redundancy-maximal-relevance criterion (mRMR) based on mutual information (Peng et al. (2005)). In addition to optimizing time execution, feature selection leads also to improve the classification accuracy. The mRMR method selects a feature subset that has the highest relevance with the target class, subject to the constraint that selected features are mutually as dissimilar to each other as possible.

Given a_i, representing the attribute i, and the class label ω, their mutual information is defined in terms of their frequencies of appearances $p(a_i)$, $p(\omega)$, and $p(a_i, \omega)$ as follows

$$I(a_i, \omega) = \int p(a_i, \omega) \log \frac{p(a_i, \omega)}{p(a_i)p(\omega)} da_i d\omega. \tag{3.8}$$

The Maximum-Relevance method selects the best individual features correlated to the class labels by finding a feature set S with m features, which jointly has the largest dependency, $D(S, \omega)$, on the target class ω

$$\max D(S, \omega), D = \frac{1}{|S|} \sum_{a_i \in S} I(a_i, \omega). \tag{3.9}$$

However, the correlations among those top features may be high. In order to remove the redundancy among features, a Minimum-Redundancy criterion, $minR(S)$, is introduced where mutual information between each pair of attributes is taken into consideration. This criterion is given by

$$\min R(S), R = \frac{1}{|S|^2} \sum_{a_i, a_j \in S} I(a_i, a_j). \tag{3.10}$$

By combining optimization criteria of Eqs. (3.9) and (3.10), mRMR improves the generalization properties of the features in the subset and the classification performance. An incremental process is used to select features satisfying optimization criteria of Eqs. (3.9) and (3.10). Suppose that A represents the whole feature set and we already selected S_{m-1}, the feature set with $m-1$ features. In order to choose the mth feature from the set $\{A - S_{m-1}\}$, the two constraints D and R are combined and the feature maximizing this combination is selected as follows

$$\max_{a_i \in A - S_{m-1}} [I(a_i, \omega) - \frac{1}{m-1} \sum_{a_i \in S_{m-1}} I(a_i, a_j)]. \tag{3.11}$$

The mth feature can also be selected as follows:

$$\max_{a_i \in A - S_{m-1}} [I(a_i, \omega) / \frac{1}{m-1} \sum_{a_i \in S_{m-1}} I(a_i, a_j)]. \tag{3.12}$$

Wine dataset

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The dataset is composed of 178 instances presenting three types of wines, or three classes, described by 13 features which are constituents found in each of the three types of wines, 30 out of the 178 instances are used for training, and the rest, 148 instances are used for testing. The dataset is randomly divided into 3 feature sets to simulate data fusion.

Sonar Dataset

The task of the Sonar dataset is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Thus, the dataset consists of two data classes for 208 objects. Each object is described by 60 features, each one representing the energy within a particular frequency band, integrated over a certain period of time. Fifty out of the 208 instances are used for training, 158 instances are used for testing. The dataset is randomly divided into 5 feature sets to simulate data fusion.

Ionosphere dataset

The Ionosphere dataset contains 351 instances and 34 attributes describing a binary classification task where radar signals target two types of electrons in the ionosphere: those that show some structure in the ionosphere (good) and those that do not (bad). These radar data are collected by a system consisted of 16 high-frequency antennas. 200 out of the 351 instances are used for training, and the rest, 151 instances are used for testing. The dataset is randomly divided into 4 feature sets to simulate data fusion.

Spectf dataset

This dataset describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patients is classified into two categories: normal and abnormal. The database consists of of 267 SPECT image sets (patients) that are described by 44 continuous features for a binary classification task. 80 out of the 267 instances are used for training, and the rest, 187 instances are used for testing. The dataset is randomly divided into 4 feature sets to simulate data fusion.

3.4.2 Classifiers used as base learners

Multilayer perceptron networks

Also known as multilayer feed-forward networks (Hornik et al. (1989)), multilayer perceptron (MLP) networks are the most widely used types of neural networks. MLP consists of an input layer followed by one or more hidden layers whose nodes are neurons and an output layer that also contains neurons. The input layer presents a vector of predictor variable values. Arriving at a neuron in the hidden layer, the value from each input neuron is multiplied by a weight (w_i), and the resulting weighted values are added together producing a combined value. The weighted sum is fed into a transfer function F, which produces an output response

$$F(\sum_{i=1}^{m} w_i x_i), \tag{3.13}$$

where $w = (w_i, , w_m)$ is the weight vector and $x = (x_1, , x_m)$ is the input vector.

The selection of the transfer function typically depends on the nature of the input and the objective of the network. Most used transfer functions are the Logistic function given by

$$f(x) = \frac{1}{1 - e^{-x}}, \tag{3.14}$$

and the Hyperbolic tangent function represented by

$$f(x) = tanh(\frac{x}{2}) = \frac{1 - e^{-x}}{1 + e^{-x}}. \tag{3.15}$$

The network aims at learning some relationship between input and output patterns. The learning process relies on the modification of the connection weights between nodes. The most common method used is back-propagation.

Classification trees by CART method

CART (Breiman et al. (1984)) is an alternative method for classification that is used to split the data into different classes by building binary decisions. There are two types of trees, classification tree and regression tree. Classification or decision tree (DT) is used when there is a need to classify the data into classes. The regression tree is used when the predicted outcome is continuous. Since in this dissertation we deal with classification problems, classification trees are used.

DT contain a binary question (yes/no answer) about some feature at each node in the tree that leads to binary partitioning by dividing the sample into two sub-samples for each node. The leaves of the tree contain the best prediction based on the training data.

First, all observations of an independent variable are placed in the root node and it is assumed that each observation of that independent variable divides the sample into two sub samples. All observations of that independent variable are called splits.

The choice of the best split depends on the purity of nodes. The most frequently

used splitting rule is the Gini index. It uses the following impurity function $i(t)$

$$i(t) = \sum_{i \neq j} p(j|t)p(i|t), \tag{3.16}$$

where i, j are indexes of the class, $p(j|t)$ is the probability of class j observation in node t and $p(i|t)$ is the probability of class i observation in node t.

The splitting rule has to maximize the change in impurity given by

$$\Delta i(t) = i(t) - p_L i(t_L) - p_R i(t_R), \tag{3.17}$$

where p_L is the proportion of observations in the left child node and p_R is the proportion of observations in the right child node.

After determining the splitting rule, the nodes termination rule should be assessed to determine whether to terminate or continue to split. A splitting process stops when it is impossible to decrease the change in impurity.

For terminal nodes, a class should be assigned as the one that has the largest amount of observations in the node in interest.

CART may have unstable DT. Insignificant modification of learning sample, such as eliminating several observations, could lead to radical changes in DT such as an increase or decrease of tree complexity or changes in splitting variables and values.

3.5 Implementation and performance analysis

All experiments are conducted using Matlab 7.8 computer software and all results are based on the holdout method which divides the dataset horizontally into two mutually

exclusive subsets. The first is used for training the classifiers and the second is used for testing the performance of the models and estimating the classification accuracy. Training and test sets for each dataset have been presented in the previous section.
For all experiments, ensembles of 5 and 10 classifiers, using MLP then DT as base learner, are evaluated for each feature set, repeating this process 10 times in order to get a good estimate of the average performance.

Performances achieved by individual feature sets are reported, in order to compare them with the fusion results. Only one combination technique is used for a single dataset, as the second one is used only in the fusion process.

For each database, results are provided in three tables. The first one gives individual and fusion performances when WMV is employed as first combination method. Therefore, only Learn++, Learn.RKJ1 and Learn.RL1 can be tested as they use WMV as a first combination method.

The second table gives individual and fusion results when the robust combination technique (RKJ), is used at first. Thus, results of Learn.WMV, Learn.RKJ2 and Learn.RL2 are provided.

Finally, individual and fusion performances of Adaboost.M1 and its two variations, Adaboost.RKJ and Adaboost.RL which use respectively RKJ and RL as combination techniques, are shown in the third table. Note again that Adaboost.M1 employs only once a combination method in order to obtain the final hypothesis either for single or multiple datasets.

Results on multiple feature dataset

For this database, MLP classifier is configured with an error goal of 0.01 for all feature sets and a number of nodes at each hidden layer as shown in Table (3.2).

Table 3.2: Number of hidden layer nodes for each feature set.

Feature set	1	2	3	4	5	6
No. hidden layer nodes	20	10	15	10	25	25

Individual and fusion performances obtained by training ensembles of MLP and DT classifiers on each dataset are shown in Table (3.3) where, as explained before, results reported are obtained by using WMV as a first combination rule. Note that FSi is the ith feature set.

Table 3.3: Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on multiple feature dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	**90.53**	90.49	75.71	81.71
FS2	58.71	71.31	71.29	72.00
FS3	84.12	86.46	72.53	74.53
FS4	76.00	84.62	76.59	80.47
FS5	68.35	72.93	64.29	67.43
FS6	72.53	72.49	68.71	69.58
Learn++	94.41	94.43	93.88	94.01
Learn.RKJ1	97.00	97.23	96.33	96.74
Learn.RL1	96.94	**97.29**	89.41	90.15

Both for MLP and DT, the first feature set (FS1) has the best classification accuracy, with highest value obtained by training an ensemble of 5 MLP classifiers, but as expected

the best fusion performance, given by Learn.RL1 with an ensemble of 10 MLP, exceeds this value leading to an improvement of about 7%.

Table 3.4: Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on multiple feature dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	91.76	**92.53**	79.59	82.06
FS2	63.59	74.06	68.47	67.73
FS3	81.71	89.82	67.59	76.06
FS4	82.53	88.65	78.76	79.47
FS5	71.82	76.18	61.71	65.7
FS6	70.35	72.00	70.29	69.11
Learn.WMV	92.35	94.41	94.59	93.82
Learn.RKJ2	96.35	97.53	96.71	96.73
Learn.RL2	96.65	**97.71**	74.18	90.47

Table (3.4) shows that the best individual classification performance is achieved by the first feature set (FS1) when an ensemble of 10 MLP classifiers is trained on this dataset. The same feature set performs the best individual classification accuracy when trained with an ensemble of 10 DT, but MLP outperforms significantly DT with a difference of 10%.

It is clear that data fusion process improves the overall classification performance as there is an improvement of 5%, obtained by Learn.RL2 with 10 MLP classifiers for each feature set, compared to the best individual performance. However, this variation of Learn++ is less effective with ensembles of DT.

As shown in Table (3.5), the best fusion result, which is achieved by Adaboost.RKJ with ensembles of 10 DT classifiers for each feature set, is slightly inferior to the best fusion results obtained previously.

Table 3.5: Fusion performances obtained by Adaboost algorithm with MLP and DT on multiple feature dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	90.21	**90.53**	78.22	80.35
FS2	69.85	71.47	70.03	70.18
FS3	86.33	86.71	71.65	73.29
FS4	84.12	85.00	77.43	81.62
FS5	73.97	74.06	70.84	71.15
FS6	72.48	73.94	67.45	69.89
Adaboost.M1	96.55	96.83	96.16	96.09
Adaboost.RKJ	96.79	96.95	97.01	**97.14**
Adaboost.RL	96.79	97.02	90.99	91.14

Figure (3.1) allows defining algorithms that perform well on multiple feature dataset, independently of the base classifier employed. Learn.RKJ1, Learn.RKJ2 and Adaboost.RKJ give the best fusion performances both with MLP and DT. Also, Adaboost.M1 performance is still good. However, Learn.RL1, Learn.RL2 and Adaboost.RL performances decrease with DT as base classifiers.

Results on Wine dataset

For Wine database, MLP classifier is configured as presented in Table (3.6).

Table 3.6: Number of hidden layer nodes for each feature set.

Feature set	1	2	3
Error goal	0.005	0.01	0.01
No. hidden layer nodes	30	25	15

Table (3.7) shows that the best fusion performance is achieved by Learn.RKJ1 with

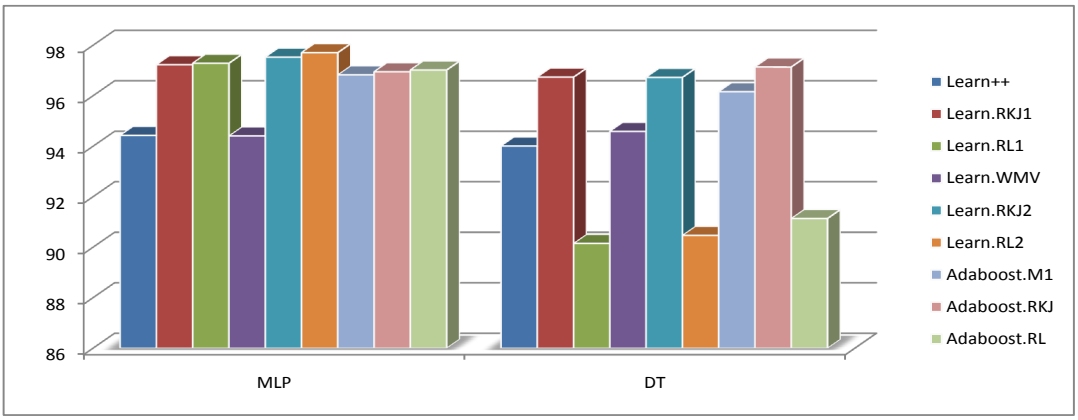

Figure 3.1: Fusion performance on Multiple feature database with T=10.

Table 3.7: Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on Wine dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	78.38	81.62	**89.05**	86.89
FS2	63.11	67.57	73.78	74.33
FS3	74.73	70.13	67.97	68.10
Learn++	77.45	73.25	81.81	70.40
Learn.RKJ1	81.22	83.65	88.78	**90.27**
Learn.RL1	83.52	86.63	75.97	71.89

ensembles of 10 DT classifiers. It is better than the best individual performance obtained by training an ensemble of 5 DT on the first feature set.

Table 3.8: Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on Wine dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	77.93	76.23	88.69	**89.46**
FS2	71.39	67.81	75.98	80.78
FS3	62.83	71.5	69.4	68.78
Learn.WMV	77.36	77.21	83.29	67.20
Learn.RKJ2	80.06	84.03	91.03	**91.83**
Learn.RL2	84.01	85.44	56.51	76.08

Table (3.8) shows that Learn.RKJ2 with ensembles of 10 DT classifiers give the highest classification accuracy, outperforming the best single performance realized by the first feature set, also with an ensemble of 10 DT.

Table 3.9: Fusion performances obtained by Adaboost algorithm with MLP and DT on Wine dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	83.78	83.78	87.43	**87.55**
FS2	68.11	67.03	77.02	74.06
FS3	74.59	75.27	71.35	69.73
Adaboost.M1	83.98	65.94	69.46	52.43
Adaboost.RKJ	83.78	85.34	90.67	**90.68**
Adaboost.RL	85.27	87.57	58.24	53.58

Adaboost.RKJ with ensembles of 10 DT leads to an improvement of about 3% over the best individual performance also obtained by an ensemble of 10 DT trained on the

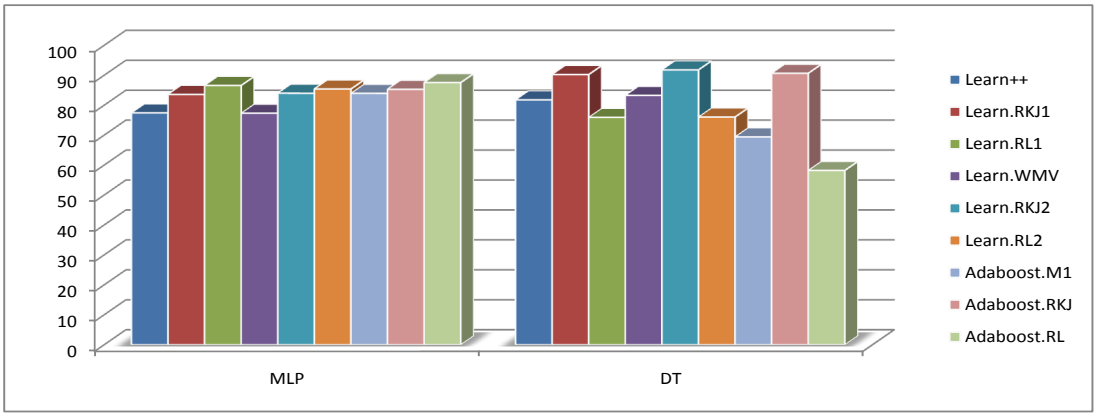

Figure 3.2: Fusion performance on Wine database.

first feature set.

Figure (3.2) shows that, as for Multiple features dataset, Learn.RKJ1, Learn.RKJ2 and Adaboost.RKJ give good fusion performances both with MLP and DT on Wine database. Also, it is shown that Adaboost.RL with DT don't performs as well as with MLP.

Results on Sonar dataset

For Sonar database, MLP classifier is configured with an error goal of 0,1 and a number of hidden layer nodes equal to 24 for all feature sets.

Table (3.10) shows that the best fusion performance is realized using Learn.RL1 with ensembles of 10 MLP classifiers. This result is better than the highest individual performance obtained also with an ensemble of 10 MLP classifiers trained iteratively on the third feature set.

Table (3.11) shows that the feature sets fusion leads to an improvement of more than

Table 3.10: Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on Sonar dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	57.85	57.84	62.16	62.15
FS2	62.28	65.70	57.72	58.86
FS3	68.23	**69.75**	59.37	58.35
FS4	60.51	64.3	62.53	65.57
FS5	59.75	59.24	55.44	58.61
Learn++	67.34	54.56	54.3	65.18
Learn.RKJ1	70.30	73.04	71.39	70.51
Learn.RL1	71.01	**73.42**	70.89	71.14

Table 3.11: Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on Sonar dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	64.43	63.04	55.32	50.25
FS2	58.73	64.30	64.18	63.04
FS3	60.50	59.49	**68.1**	**68.1**
FS4	65.44	63.92	62.03	65.19
FS5	58.1	60.00	55.69	55.44
Learn.WMV	66.03	67.47	70.63	72.53
Learn.RKJ2	73.04	72.53	69.74	72.03
Learn.RL2	**73.67**	72.66	69.75	73.42

5% over the best single dataset which is FS3. While the highest classification rate is obtained using ensembles of DT, the best fusion performance is obtained by Learn.RL2 with ensembles of MLP classifiers.

Table 3.12: Fusion performances obtained by Adaboost algorithm with MLP and DT on Sonar dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	56.96	63.03	61.65	63.04
FS2	62.40	65.57	54.68	60.38
FS3	69.49	**70.13**	57.97	59.62
FS4	63.04	64.43	62.15	65.57
FS5	60.38	59.62	55.19	58.23
Adaboost.M1	70.25	72.59	67.40	69.68
Adaboost.RKJ	68.48	71.07	69.94	73.42
Adaboost.RL	69.62	71.44	69.24	**73.73**

For Adaboost, FS3 gives the highest individual performance with an ensemble of 10 MLP classifiers. Concerning the best fusion result, it is obtained using Adaboost.RL with ensembles of 10 DT for each feature set.

Figure (3.3) shows that both with MLP and DT classifiers, Learn.RKJ2 and Learn.RL2 have good fusion performances. However, Learn.WMV is not as effective with MLP as with DT.

Results on Ionosphere dataset

For Ionosphere database, MLP classifier is configured with an error goal of 0,01 and 25 nodes at each hidden layer for all feature sets.

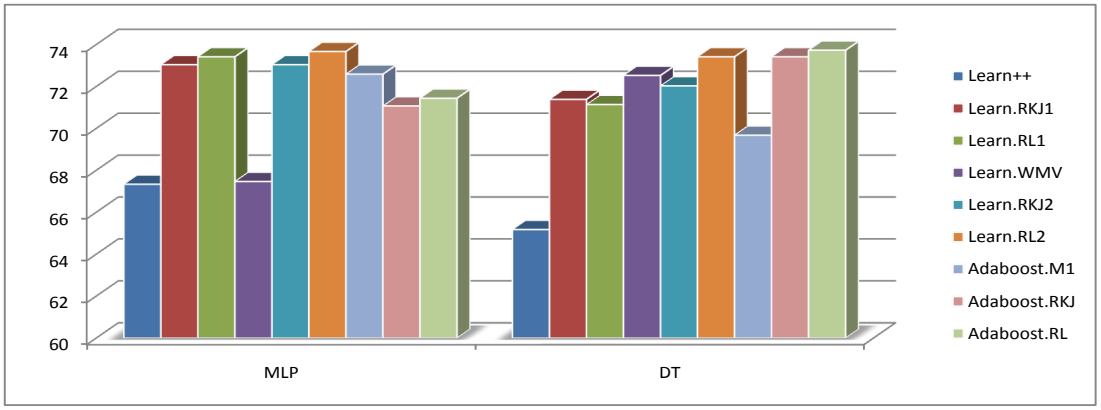

Figure 3.3: Fusion performance on Sonar database.

Table 3.13: Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on Ionosphere dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	63.18	66.76	90.2	90.59
FS2	70.33	71.25	90.86	**92.85**
FS3	71.39	64.5	87.58	88.87
FS4	57.22	71.26	88.61	88.87
Learn++	69.00	77.35	94.17	94.30
Learn.RKJ1	67.01	73.91	91.92	92.45
Learn.RL1	69.00	73.91	93.51	**94.97**

Table (3.13) shows that the best individual performance is obtained by training 10 DT classifiers iteratively on the second feature set (FS2). The best fusion result exceeds this performance, with the same ensemble configuration for each feature set and using Learn.RL1 algorithm.

Table 3.14: Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on Ionosphere dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	60.26	58.4	**87.28**	87.08
FS2	66.22	68.74	80.00	81.85
FS3	59.2	61.32	87.02	84.93
FS4	66.75	72.18	86.62	86.75
Learn.WMV	77.35	58.8	**95.23**	94.96
Learn.RKJ2	67.95	72.18	93.13	92.18
Learn.RL2	66.35	68.34	94.70	94.83

Table (3.14) shows that when using the robust method in the first combination phase, the highest individual classification accuracy is obtained by training an ensemble of 5 classifiers, with DT as base learner, on the first feature set (FS1). Learn.WMV gives the best fusion result, also with ensembles of 5 DT for each feature set.

As presented in Table (3.15), Adaboost.RL gives the best fusion performance, when ensembles of 10 DT are constructed for each feature set. This performance exceeds the highest individual classification accuracy achieved by an ensemble of DT trained on the second feature set.

As shown in Figure (3.4), Learn.WMV and Learn++ give good results both for the two base classifiers. However, Learn.RL2 while performing well when constructing DT classifiers, it is the less effective algorithm when evaluated with MLP classifiers.

Table 3.15: Fusion performances obtained by Adaboost algorithm with MLP and DT on Ionosphere dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	62.65	60.13	90.73	91.26
FS2	59.73	65.03	**93.24**	**93.24**
FS3	57.62	61.85	88.08	89.40
FS4	63.58	69.93	87.41	87.42
Adaboost.M1	66.62	69.93	94.37	94.57
Adaboost.RKJ	69.00	71.06	92.12	92.72
Adaboost.RL	65.76	69.54	94.90	**94.97**

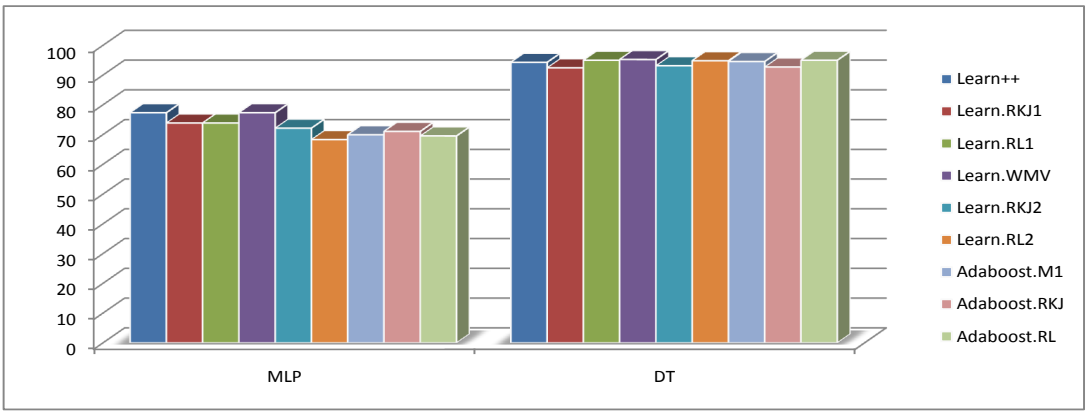

Figure 3.4: Fusion performance on Ionosphere database.

Results on Spectf dataset

For Spectf database, MLP classifier is configured with an error goal of 0,1 and 25 nodes at each hidden layer for all feature sets.

Table 3.16: Fusion performances obtained by Learn++ based WMV as first combination rule with MLP and DT on SPECTF dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	59.14	69.41	73.53	**75.05**
FS2	66.84	66.73	69.07	68.13
FS3	60.10	65.56	67.64	66.09
FS4	66.42	68.88	67.11	68.66
Learn++	69.84	71.34	71.84	67.91
Learn.RKJ1	69.30	73.58	74.87	74.55
Learn.RL1	70.48	73.15	72.28	72.94

Table (3.16) shows that the classification accuracy decreases slightly with the fusion process if we compare the best individual and fusion performances.

Table 3.17: Fusion performances obtained by Learn++ based RKJ as first combination rule with MLP and DT on SPECTF dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	61.28	63.15	71.65	68.98
FS2	63.74	69.30	67.92	70.80
FS3	56.74	67.38	65.35	70.27
FS4	68.13	71.01	71.87	**72.3**
Learn.WMV	68.23	71.47	71.98	57.09
Learn.RKJ2	69.30	**75.19**	74.33	**75.19**
Learn.RL2	70.29	74.00	73.26	73.48

Ensemble of 10 DT classifiers trained iteratively on FS4 gives the best individual performance. This performance is lower than the best fusion result which is obtained by Learn.RKJ2 with both ensembles of 10 MLP and DT classifiers.

Table 3.18: Fusion performances obtained by Adaboost algorithm with MLP and DT on SPECTF dataset.

	MLP		DT	
Feature set/N classifiers	T=5	T=10	T=5	T=10
FS1	60.96	66.20	70.91	70.37
FS2	70.59	71.66	67.59	69.73
FS3	64.39	68.02	65.66	68.55
FS4	**74.03**	73.37	69.52	68.34
Adaboost.M1	65.13	65.24	72.19	69.52
Adaboost.RKJ	71.39	75.03	**75.51**	74.49
Adaboost.RL	70.69	75.08	73.26	73.32

It is noticed in Table (3.18) that there is a little improvement realized by Adaboost.RKJ constructing ensembles of 5 DT classifiers, over the best individual classification performance obtained by FS4 with an ensemble of 5 MLP classifiers.

For this database, Figure (3.5) shows that Learn.RKJ2 and Adaboost.RKJ are the best algorithms, with both base classifiers. In the other hand, Adaboost.M1 performance differs the most when modifying the base classifier.

3.5.1 Results summary

This section summarizes evaluation results for Learn++, Adaboost.M1 and their proposed variations for fusing multiple feature sets. Each empty box in Table (3.19) indicates that for the given dataset, the corresponding algorithm gives a fusion performance that is lower than the best performance obtained by a single feature set with that al-

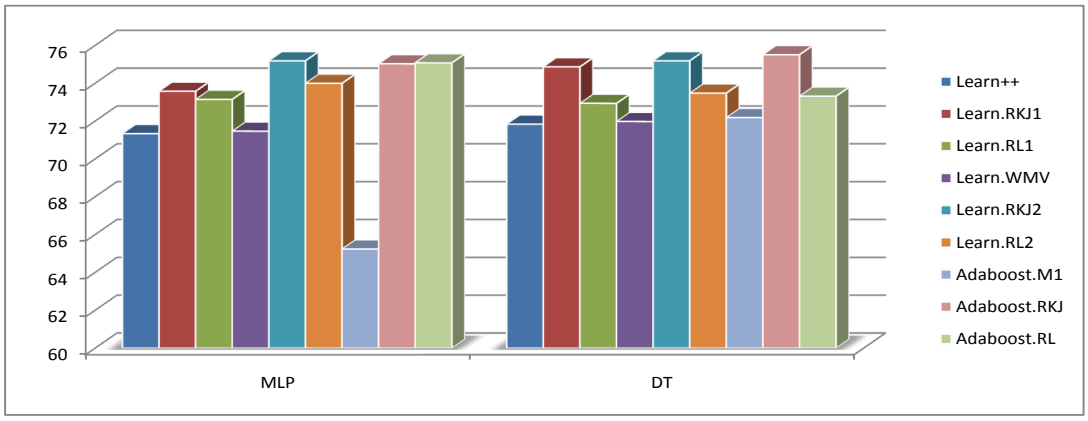

Figure 3.5: Fusion performance on SPECTF database.

gorithm. Thus, it is not useful to show this fusion performance. Results provided are the best individual and fusion results obtained by each algorithm for each dataset when comparing MLP and DT results.

Note that for the first category, the individual feature set performance is obtained by using WMV as a unique combination rule. It is used to combine the existing ensemble of classifiers and update the training set distribution for the next step based on the composite hypothesis obtained. In the last step, the composite hypothesis is the final hypothesis for the feature set. This process is the application of the original Learn++ for a single dataset.

In the second category, the individual performance is obtained by applying the same procedure. However, instead of WMV, RKJ is used as unique combination rule.

In the third category, the individual feature set performance is the result of the application of Adaboost.M1 on that feature set, that is for each training step updating the data distribution for the next iteration based on the previous classifier's performance and getting the final hypothesis by combining all classifiers obtained by WMV.

Table 3.19: Summary of fusion performances for all databases

	Algorithm / Database	Multiple feature	Wine	Sonar	Ionosphere	Spectf
1st group	Best feature set result	90.53	89.05	69.75	92.85	75.05
	Learn++	94.43	-	-	94.30	-
	Learn.RKJ1	97.23	90.27 *	73.04	-	-
	Learn.RL1	97.29 *	-	73.42 *	94.97 *	-
2nd group	Best feature set result	92.53	89.46	68.1	87.28	72.3
	Learn.WMV	94.59	-	72.53	**95.23****	-
	Learn.RKJ2	97.53	**91.83****	73.04	93.13	75.19 *
	Learn.RL2	**97.71****	-	73.67 *	94.83	74.00
3rd group	Best feature set result	90.53	87.55	70.13	93.24	74.03
	Adaboost.M1	96.83	-	72.59	94.57	-
	Adaboost.RKJ	97.14 *	90.68 *	73.42	-	**75.51****
	Adaboost.RL	97.02	87.57	**73.73****	94.97 *	75.08

*Best of the group
**Best of all algorithms

Results show that Learn++ is not always effective in data fusion applications as for three out of five databases, empty boxes indicate that it gives fusion results that are lower than those of the best individual feature set. Also, for the two other applications, it leads to moderate results compared with Learn++ variations and Adaboost algorithms.

It is noticed that the first group of algorithms in Table (3.19), containing variations of Learn++ which use WMV as a first combination rule, gives lower fusion results and even deteriorates the classification accuracy compared to the best individual feature set performance.

Most of the best fusion results are obtained by the second category which regroups variations of Learn++ that use RKJ as first combination technique. In this category Learn.RKJ2, which uses RKJ also for the fusion process, outperforms the best individual performance for all databases. Learn.WMV, while giving the best result for Ionosphere database, is not always effective. This shows again that WMV is not always appropriate

for data fusion applications.

For Multiple feature dataset, Learn.RL2 gives the best performance and also gives good results for three other databases.

In the third category that regroups Adaboost.M1 and its proposed variations, fusion classification results are often better than individual results. Adaboost.RKJ gives often the best results in this category, however Adaboost.RL can be considered more robust as it outperforms the best individual classification accuracy for all databases.

Also, agreeing with what Table (3.19) presents, previous results have shown that Learn.RKJ2 gives always good fusion performance and this performance remains often stable if the base classifier is changed. Learn.RL2 performance, while sometimes not stable depending on the base classifier used, is also often good for data fusion applications. Finally, Adaboost.RKJ and Adaboost.RL complete this list of the most effective algorithms evaluated for the data fusion applications proposed.

The common remark for these four algorithms is that they all use only robust combination rules in the two combination phases. A slight advantage is noticed for RKJ compared to RL as a robust combination technique.

All results and conclusions obtained agree with our hypothesis that WMV is not always appropriate for data fusion applications and that more robust combination methods, as those proposed in this chapter, take advantage of the additional information available leading to better classification performance.

3.6 Conclusion

This chapter investigates the effectiveness of employing robust combination techniques for ensemble methods in the context of data fusion problems. The robust combination rules developed are based on conflict resolution to reduce the influence of outlier models in the ensemble. From these combination rules, several variations of Adaboost.M1 and Learn++ algorithms are obtained. These variations are Learn.RKJ1, Learn.RL1, Learn.RKJ2 Learn.RL2 Learn.WMV, Adaboost.RKJ and Adaboost.RL. They differ from each other on the choice of the methods used for the two phases when a combination rule is needed.

This study shows that fusing multiple sources of data is effective only when appropriate combination methods are used. In this context, we show that WVM is not always adequate for this kind of problems. However, four algorithms, namely Learn.RKJ2, Learn.RL2, Adaboost.RKJ and Adaboost.RL, give good fusion performance for most applications. These methods do not only give good fusion results, but also for most of them the performance remains stable when the base classifier is changed.

Future research should focus on studying the effectiveness of employing other robust classifiers combination strategies, based mainly on the detection and resolution of the conflict between the ensemble classifiers. It should be of interest to test other ensemble methods with robust combination rules in the context of data fusion applications.

General Conclusion

Given that ensemble methods are effective in a wide variety of applications, this study addressed the issue of applying them in order to solve data fusion problems. The goal of fusing information obtained from multiple sources is to get a more informed decision and to improve the classification performance. The ensemble structure provides a natural mechanism to combine heterogeneous features and allows efficient learning of complementary information from each dataset.

To have a clear idea about the basic principles of ensemble systems Chapter 1 provided an overview of these methods. Two fundamental components have been presented, namely the strategy used for constructing diverse ensembles like Bagging and Boosting, and the strategy for combining them. We have shown the importance of diversity for the ensemble success by presenting the most widely used techniques. Furthermore, categorizations of ensemble techniques have been discussed.

In order to evaluate ensemble systems, Chapter 2 focused on the application of these approaches on data fusion problems, one of the different domains in which ensemble techniques have shown their effectiveness. Different levels for combining data from multiple sources have been described, showing that the decision fusion level can be considered appropriate for most data fusion applications. This chapter has also reviewed several techniques employed for this purpose, especially ensemble methods whose struc-

ture provides a natural mechanism to combine multiple heterogeneous sources of data and take advantage from complementary information each source brings. Adaboost.M1 and Learn++ ensemble algorithms have been evaluated, with a general conclusion that a robust combination rule is required to achieve better classification performance when fusing multiple sources of data.

Chapter 3 investigated the effectiveness, in the context of data fusion problems, of robust combination techniques for ensemble methods. Robust combination rules developed were based on conflict resolution and aimed at reducing the influence of outlier models in the ensemble. From these combination rules, seven variations of Adaboost.M1 and Learn++ have been proposed. Four out of these variations, namely Learn.RKJ2, Learn.RL2, Adaboost.RKJ and Adaboost.RL, gave good fusion performance for most of the applications. Also, it has been shown that WVM is not always appropriate in the context of data fusion.

Some questions need further investigations. The first, would be to study the effectiveness of employing other robust classifiers combination strategies, based mainly on the detection and resolution of the conflict between the ensemble classifiers. Another direction consists on testing other ensemble methods with robust combination rules in the context of data fusion applications. Since ensemble methods evaluated for data fusion can use any base classifier, different types of learning algorithms particularly suited to a specific set of features could be combined to construct hybrid ensembles for improved data fusion performance.

Bibliography

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1-2):105–139.

Biessmann, F., Meinecke, F., Gretton, A., Rauch, A., Rainer, G., Logothetis, N., and Mueller, K. (2010). Temporal kernel cca and its application in multimodal neuronal data analysis. *Machine Learning*, 79(1-2):5–27.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996b). Stacked regressions. *Machine Learning*, 24(1):49–64.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics*, 26(3):801–849.

Breiman, L. (1999). Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1-2):85–103.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth Publishing Company, Belmont, California, U.S.A.

Briem, G., Benediktsson, J., and Sveinsson, J. (2002). Multiple classifiers applied to multisource remote sensing data. *IEEE transactions on geoscience and remote sensing*, 40(10):2291–2300.

Brown, G., Wyatt, J., Harris, R., and Yao, X. (2005). Diversity creation methods: A survey and categorisation. *Information Fusion*, 6(1):5–20.

Bryll, R., Gutierrez-Osuna, R., and Quek, F. (2003). Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36:1291–1302.

Dempster, A. (1967). Upper and lower probabilities induced by multivalued mappings. *Annals of Mathematical Statistics*, 38:325–339.

Dietterich, T. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. In *Bagging, boosting, and randomization. Machine Learning*, pages 139–157.

Dietterich, T. (2000). Ensemble methods in machine learning. In Kittler, J. and Roli, F., editors, *First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag.

Dietterich, T. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2(1):263–286.

Freund, Y. and Schapire, R. (1997). Decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, J. (2002). Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407.

Garcia, M. and Puig, D. (2004). Robust aggregation of expert opinions based on conflict analysis and resolution. In *Current Topics in Artificial Intelligence*, pages 488–497. Springer.

Gong, B. (2009). An algorithm of data fusion using artificial neural network and dempster-shafer evidence theory. In *CASE '09: Proceedings of the 2009 IITA International Conference on Control, Automation and Systems Engineering (case 2009)*, pages 407–410.

Hall, D. and Llinas, J. (1997). An introduction to multisensor data fusion. In *IEEE*, volume 85, pages 6–23. Springer.

Hansen, L. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Huang, Y. and Suen, C. (1995). A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):90–94.

Jacobs, R., Jordan, M., Nowlan, S., and Hinton, G. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Jain, A., Duin, P., and Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37.

Kittler, J. (1986). Feature selection and extraction. In *HPRIP86*, pages 59–83.

Kuncheva, L. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley and Sons, Inc.

Kuncheva, L., Bezdek, J., and Duin, R. (2001). Decision templates for multiple classifier fusion: An experimental comparison. *Pattern Recognition*, 34:299–314.

Langdon, W., Barrett, S., and Buxton, B. (2002). Combining decision trees and neural networks for drug discovery. In *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002, volume 2278 of LNCS*, pages 60–70. Springer-Verlag.

Littlestone, N. and Warmuth, M. (1994). Weighted majority algorithm. *Information and Computation*, 108(2):212–261.

Margineantu, D. and Dietterich, T. (1997). Pruning adaptive boosting. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 211–218, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Melville, P. and Mooney, R. (2005). Creating diversity in ensembles using artificial data. *Information Fusion*, 6:99–111.

Merler, M., Caprile, B., and Furlanello, C. (2007). Parallelizing AdaBoost by weights dynamics. *Computational Statistics and Data Analysis*, 51(5):2487–2498.

Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198.

Oza, N. (2003). Boosting with averaged weight vectors. In Roli, F., Kittler, J., and Windeatt, T., editors, *Fourth International Workshop on Multiple Classifier Systems*, pages 15–24. Springer-Verlag.

Oza, N. and Tumer, K. (2001). Input decimation ensembles: Decorrelation through dimensionality reduction. In *LNCS*, pages 238–247. Springer.

Pazzani, M. and Domingos, P. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130.

Peng, H., Long, F., and Ding, C. (2005). Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1226–1238.

Phama, T. and Smeuldersb, A. (2008). Quadratic boosting. *Pattern Recognition*, 41(1):331–341.

Polikar, R., Topalis, A., Green, D., Kounios, J., and Clark, C. (2008). Ensemble based data fusion for early diagnosis of alzheimer disease. *Information Fusion*, 9(1):83–95.

Quinlan, J. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 725–730.

Re, M. and Valentini, G. (2009). Prediction of gene function using ensembles of svms and heterogeneous data sources. In *Applications of Supervised and Unsupervised Ensemble Methods*, pages 79–91.

Rokach, L., Maimon, O., and Arad, O. (2005). Improving supervised learning by sample decomposition. *International Journal of Computational Intelligence and Applications*, 5(1):37–54.

Sa, V., Gallagher, P., Lewis, J., and Malave, V. (2010). Multi-view kernel construction. *Machine Learning*, 79(1-2):47–71.

Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press, Princeton.

Sun, Y., Todorovic, S., and Li, L. (2006). Reducing the overfitting of adaboost by controlling its data distribution skewness. *International Journal of Pattern Recognition and Artificial Intelligence*, 20(7):1093–1116.

Szafranski, M., Grandvalet, Y., and Rakotomamonjy, A. (2010). Composite kernel learning. *Machine Learning*, 79(1-2):73–103.

Wald, L. (1999). Some terms of reference in data fusion. *IEEE Transactions on Geosciences and Remote Sensing*, 37(3):1190.

Wang, W., Jones, P., and Partridge, D. (2000). Diversity between neural networks and decision trees for building multiple classifier systems. In *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 240–249, London, UK. Springer-Verlag.

Wang, Y., Dunham, M., Waddle, J., and Mcgee, M. (2006). Classifier fusion for poorly differentiated tumor classification using both messenger rna and micro-rna expression profiles. In *CBS 2006*.

Williams, D. (2009). Bayesian data fusion of multiview synthetic aperture sonar iagery for seabed classification. *IEEE Transactions on Image Processing*, 18(6):1239–1254.

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, 5(2):241–259.

Woods, K., Kegelmeyer, W., and Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410.