12-31-2006

Random feature subspace ensemble based approaches for the analysis of data with missing features

Hussein Syed Mohammed
Rowan University

Random feature subspace ensemble based approaches for the analysis of data with missing features

by

Hussein Syed Mohammed

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Electrical and Computer Engineering
Major: Engineering (Electrical Engineering)

Approved:

In Charge of Major Work

For the Major Department

For the College

Members of the Committee

Rowan University
Glassboro, New Jersey
2006

ABSTRACT

Hussein Syed Mohammed

RANDOM FEATURE SUBSPACE ENSEMBLE BASED APPROACHES FOR THE ANALYSIS OF DATA WITH MISSING FEATURES

2006
Dr. Robi Polikar
Master of Science

Missing data in real world applications is not an uncommon occurrence. It is not unusual for training, validation or field data to have missing features in some (or even all) of their instances, as bad sensors, failed pixels, malfunctioning equipment, unexpected noise causing signal saturation, data corruption, and so on, are all familiar scenarios in many practical applications.

In this thesis, the feasibility of an ensemble of classifiers trained on a feature subset space is investigated as an effective and practical solution for the missing feature problem. Two ensemble of classifiers approach motivated by the Random Subspace Method are proposed for supervised classifiers to handle data with missing features. A sufficiently large number of classifiers are trained, each with a random subset of the features. Those instances with missing features are then classified by a majority voting of those classifiers whose training data did not include the missing features. The proposed algorithm, Learn^{++}.MF, along with a modified version of this algorithm, Learn^{++}.MFv2, are introduced in this effort. We also investigate the effect of varying the cardinality of the random feature subsets on the classification performance, discuss the conditions under which the proposed approaches are most effective, and present simulation results on several benchmark datasets.

ACKNOWLEDGEMENTS

I want to thank my family, especially my Mother and close relatives for their support, concern and their pride in me. They have given me the freedom and independence to pursue my interests and goals in life. Over the years they have amazed me with their understanding, their dedication, and most importantly their kindness and patience. Everything I am and everything I have accomplished I owe to them, and I sincerely thank them for all their love and forever indebted to them.

I would like to thank the faculty at the Rowan University Electrical and Computer Engineering Department for providing me with an opportunity to excel in my graduate studies. Particularly, I would like to thank my advisor Dr. Robi Polikar for his belief and faith in me. Additionally, I would also like to thank my committee members Dr. Shreekanth Mandayam and Dr. Christopher Lacke. I also want to thank all my colleagues and friends in the Graduate Student Office for their camaraderie and assistance. I also want to make special thanks to the engineering clinic team for their endeavor and team effort. I am indeed thankful for National Science Foundation's Grant Number ECS-0239090 and support.

Finally, I want to sincerely thank God for allowing me to complete this chapter in my life. I give praise and thanks for His everlasting mercy and grace that endures forever and without which this thesis would not have been possible. Blessed be the name of the God now and forever.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 – INTRODUCTION

1.1 The Missing Feature Problem and Motivation

Many pattern recognition professionals and statisticians often face the problem of missing or corrupt data in their field of work. There can be different levels of missing values, from a single value in a few observation vectors, to almost all features for all observations. They can occur either randomly or systematically, in certain patterns or independently from each other. Their distribution might be different in training and test data sets; the training set might be complete and missing values occur only in test data. Furthermore, the missing data values might complicate the analysis of the data, if the classification algorithm cannot cope with such values.

Many classification algorithms, including most standard neural network architectures, require that the number and nature of the features be set prior to the training phase. Since the underlying operation for most classifiers in a neural network is a matrix multiplication, instances missing even a single feature cannot be processed by such classifiers, due to the missing number(s) in the vectors/matrices to be multiplied. Hence, the field or test data to be evaluated by the classifier must contain exactly the same set and number of features as the training data used to create the neural network to make a valid classification.

Missing data in real world applications is a common occurrence. It is not unusual for training, validation or field data to have missing features in some (or even all) of their instances, as bad sensors, failed pixels, malfunctioning equipment, unexpected noise causing signal saturation, data corruption, etc. are familiar scenarios in many practical

applications. To make matters worse, different instances of the data may be missing different features, particularly if all of the data are not acquired at the same time, at the same location, or using the same equipment, etc. An example is illustrated in Figure 1.1, (for a handwritten character recognition problem), where the characters are digitized on an 8x8 grid (creating 64 features, f1 ~ f64), and about 20% of pixel values are missing.

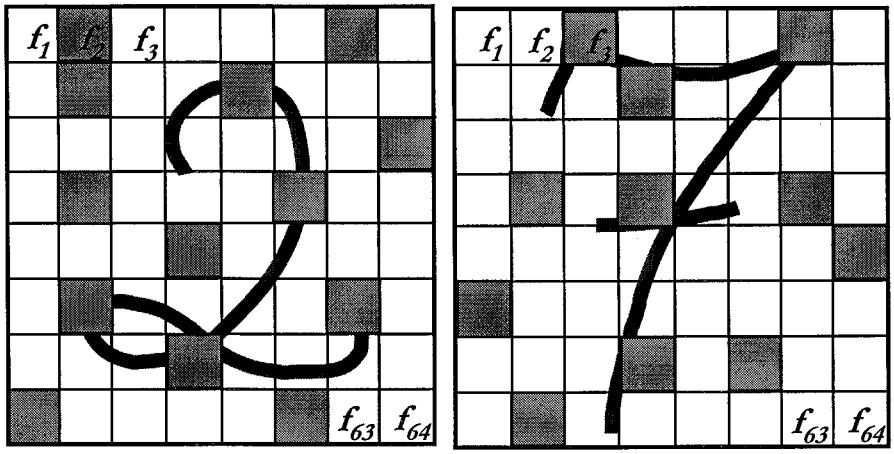

Figure 1.1: Test instance having corrupt or missing features caused by bad sensors

In most practical applications, it is often expensive, impractical or even impossible to recreate the database, demonstrating the need for a practical and robust solution to the missing feature problem.

1.2 Objectives of Thesis

The main objectives of this thesis are:

1. *To implement an ensemble approach based on the Random Subspace Method for the feasibility of the Missing Feature Problem.*
2. *To study the effects of using such a method for the Missing Feature Problem on multiple real world applications and benchmark datasets.*
3. *To investigate the effects of the algorithm's free parameters (i.e. number of features) on its performance.*
4. *To investigate modifications to the Learn^{++}.MF to attempt to boost its performance using established techniques such as weighted combination rules.*
5. *To study the effects of using the modified algorithm, along with its free parameters and compare the performance on the same datasets.*

1.3 Expected Contributions

This thesis describes two algorithms for handling the missing feature problem common in datasets. The illustration of each algorithm, along with their results on multiple datasets will be used to demonstrate their potential as possible solutions to the missing feature problem.

1.4 Scope and Organization

The scope of this thesis examines the feasibility of using two ensemble techniques based on the random selection of features for the classification of instances with missing features in datasets. The organization of the rest of this thesis is divided into the following chapters:

Chapter 2 provides a literature review of previous techniques used to solve or handle the problem of missing features and values in datasets.

Chapter 3 provides the necessary background for the techniques used in accomplishing the thesis objectives. This includes a general taxonomy on ensemble-based systems, including a detailed description of two boosting based ensemble approaches, AdaBoost.M1 and Learn^{++}, and as well as the Random Subspace Method.

Chapter 4 provides an explanation of the proposed approach, assumptions, and techniques used to accomplish the outlined objectives. The methodology for the algorithm Learn^{++}.MF is presented first, along with motivation for improving upon this algorithm. This is followed by the methodology of the second version of this algorithm, Learn^{++}.MFv2 and its necessary modifications.

Chapter 5 presents the results obtained by the algorithms Learn^{++}.MF and Learn^{++}.MFv2 on several real world applications and benchmark datasets. Finally, a study and comparison of the algorithms is performed.

Chapter 6 provides a summary of accomplishments, conclusions from this work, and recommendations for future work.

CHAPTER 2 – LITERATURE REVIEW

Missing values occur in many real-world problems, and there has been a growing library of publications dealing with missing data, indicating the increase in the seriousness and importance of the matter. However, the problem of dealing with missing values in practical applications is often dealt in an *ad hoc* basis and not reported in literature. The issue of missing values has been studied extensively in the statistical and machine learning community. Rubin further divides the case of missing values into three data mechanisms: Missing Completely At Random (MCAR), Missing At Random (MAR) and Non-ignorable. MCAR is when the probability of missing a value is the same for all variables, MAR is when the probability of missing a value is only dependent on other variables, and Non-ignorable describes the probability of missing a value is dependent on the value of the missing variable.

In this chapter, a short review of the terminology and many of the standard methods is given. The methods presented include the most prominent and established in literature, along with recent developments used to solve or counter the missing feature problem.

2.1 Filtering Method

A pragmatic approach known as *casewise* deletion is often used to handle the missing data is to ignore or delete those instances with missing features. This filtering method (see Figure 2.1) is generally the most conservative and the easiest to carry out. Its advantages are its simplicity and need for minimal computational time. It is, however, unsatisfactory for dealing with large amounts of missing data. It is often common for

many practical problems where the instances that remain after deletion are too few to effectively capture the relevant statistics (See Figure 2.2). This simple, but brute-force approach is not only suboptimal – as discarded data may still carry important information – but it may not even be practical or efficient, if all instances are missing one or more features. Variations of this method also include the *listwise* and *pairwise* deletion.

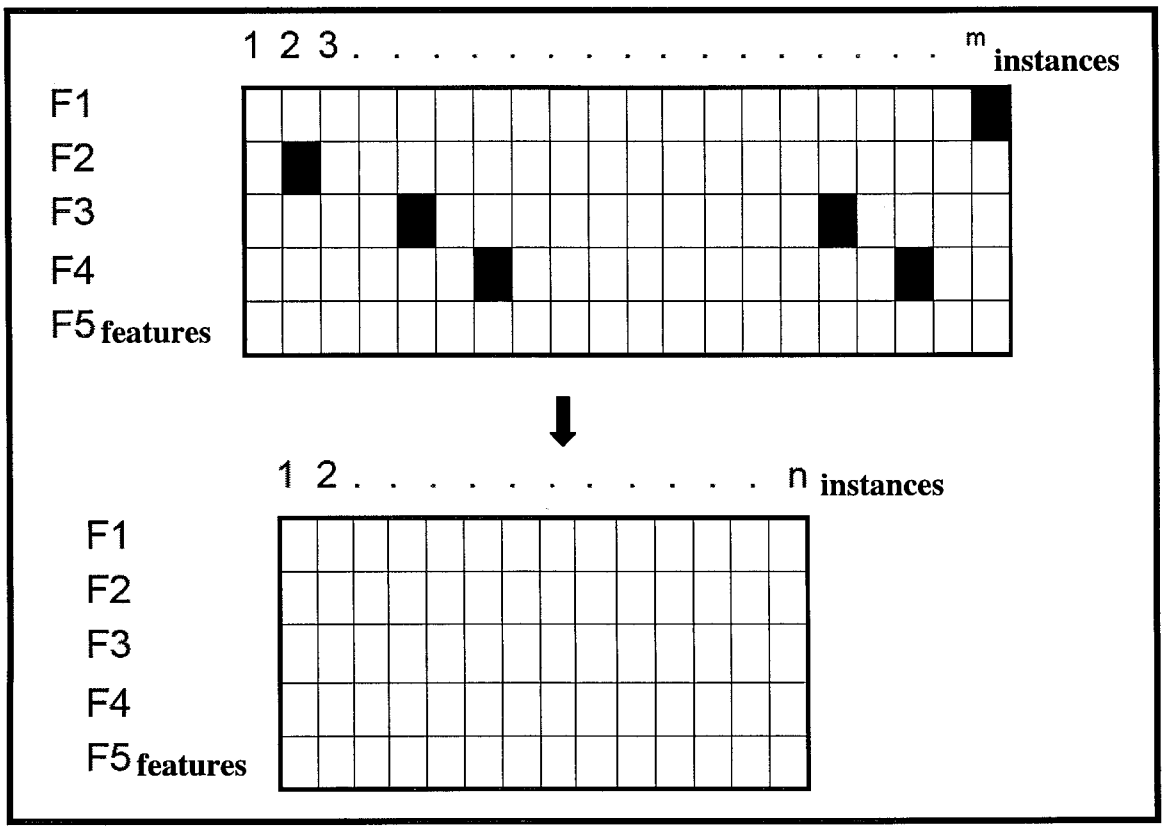

Figure 2.1: Casewise deletion resulting in n instances where $n < m$

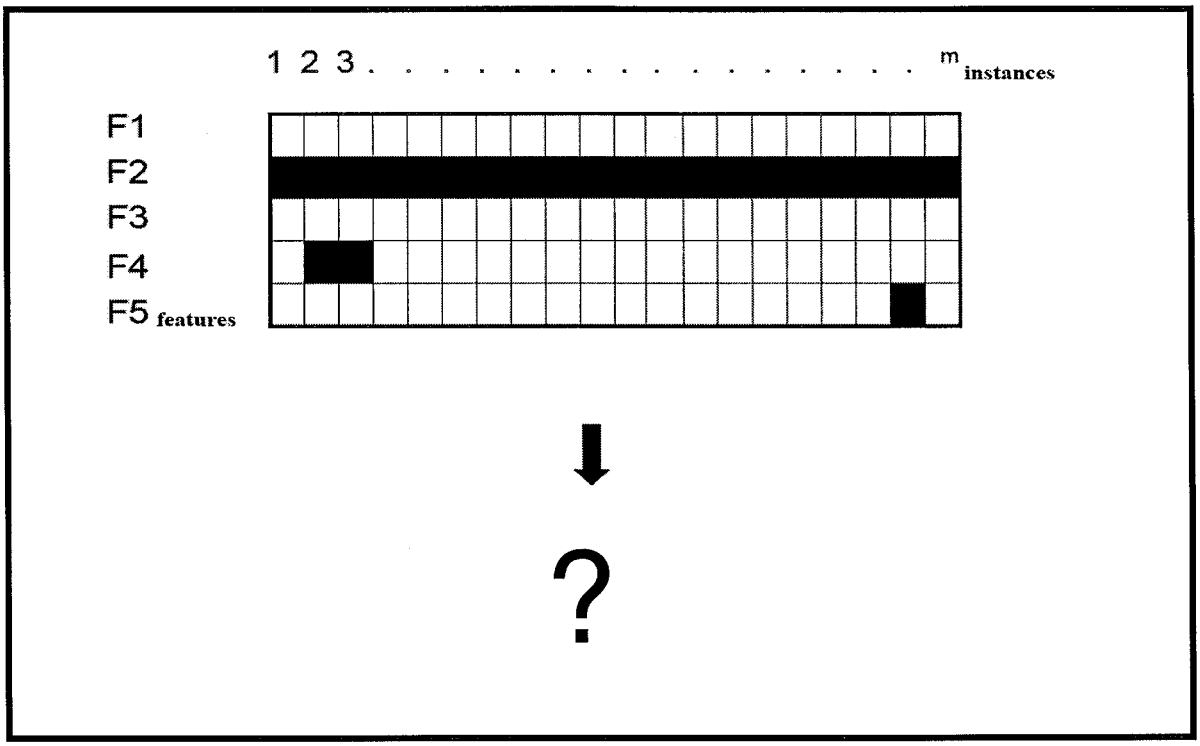

Figure 2.2: Casewise deletion resulting in no classifiable instance whereby all instances are missing one or more features missing

2.2 Bayesian Estimation

Several theoretical and heuristic methods have been proposed for the missing feature problem. Many of them rely on estimation techniques, such as Bayesian estimation [1,2] for extracting "class probabilities". We illustrate a simple case of missing features from a distribution of four classes in Figure 2.3. In Figure 2.3 below we show that feature x_1 is missing and the measured value of feature x_2 is $\hat{x}_2$. If we assume that the missing value can be substituted as the mean of all the x_1 (i.e. $\bar{x}_1$), then we might pick ω_3. However, if the prior probabilities are equal, then ω_2 would make a better decision since it has the largest likelihood $p(\hat{x}_2|\omega_2)$.

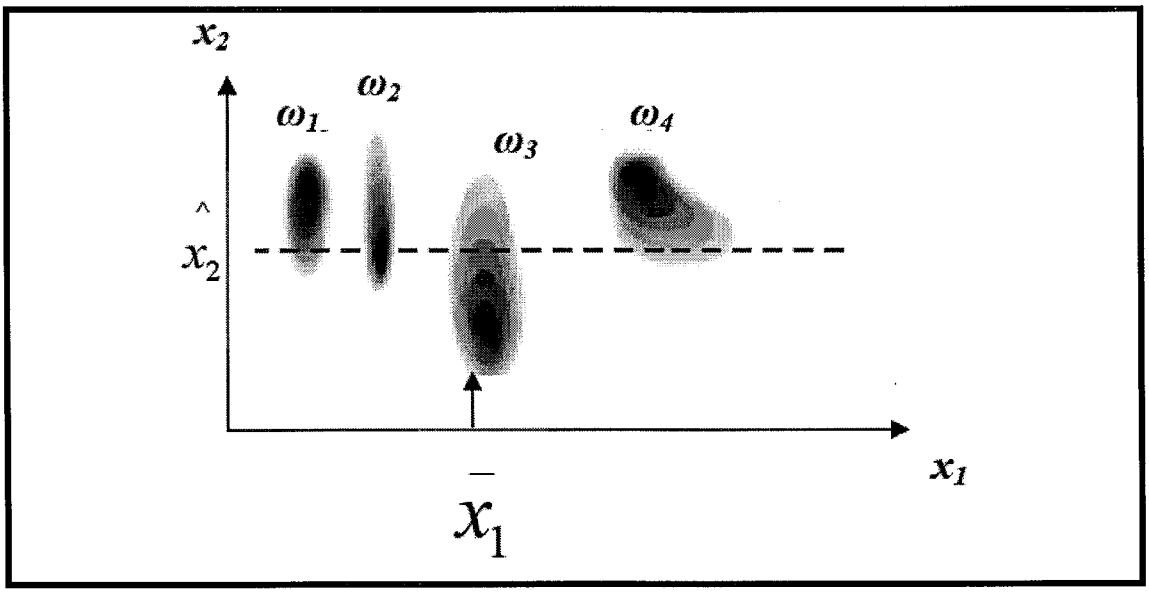

Figure 2.3: Class conditional distributions for a four class problem [7]

In terms of a set of existing features, the posteriors are

$$P(\omega_i \mid x_g) = \frac{\int g_i(x)p(x)dx_b}{\int p(x)dx_b} \tag{2.1}$$

where x_b indicates the bad or missing features, x_g indicates the good or present features, and g_i(x) is the discriminant function. In short, equation (2.1) presents the integrated, marginalization of the posterior probability over the missing features. Hence, if one knows the full probability structure of the problem, one can construct a *Bayes* decision rule. Equation (2.2) below shows the *Bayes* descision rule being used on the resulting posterior probabilities, which is to choose the class with the largest posterior probability.

$$P(\omega_i|x_g) > P(\omega_j|x_g) \text{ for all i and j} \tag{2.2}$$

A major setback for this method is that it requires access to a dense data distribution for it to be able to approximate the missing values. Furthermore, it also requires specifying a prior distribution for all unknown parameters. But, in many cases,

prior knowledge is either vague, or non-existent, which often makes it very difficult to specify a useful prior distribution.

2.3 Expectation Maximization

Another alternative strategy for computing incomplete data problems is the Expectation Maximization (EM) algorithm [3,4,5]. The EM Algorithm is an iterative procedure to find the maximum likelihood estimates of parameters or the posterior mode of parameters in a model. The algorithm augments the observed data with latent data, which can be either missing data or parameter values, so that the likelihood function conditioned on the data and the latent data has a form that is easy to analyze.

A main advantage of the EM algorithm lies in its theoretical simplicity. It consists of two iterative steps, the E step (expectation step) and M step (maximization step), which update class conditional probabilities such that they are maximized over existing data. These steps are often easy to construct conceptually.

1. *Begin Initialize* θ^0, δ, $k \leftarrow 0$
2. *Do* $k \leftarrow k+1$
3. *E step:* $Q(\theta; \theta^k)$
4. *M step:* $\theta^{k+1} \leftarrow \arg\max_{\theta} Q(\theta; \theta^k)$
5. *Until* $Q(\theta^{k+1}; \theta^k) - Q(\theta; \theta^{k-1}) \leq \delta$
6. *Return* $\hat{\theta} \leftarrow \theta^{k+1}$
7. *End*

Figure 2.4: Pseudocode of EM algorithm [7]

Figure 2.4 shows the pseudocode of the EM algorithm. We define θ as the model parameter, δ as our preset convergence criterion, and k as the number of intended iterations until convergence is met.

The EM algorithm consists of choosing an initial $\theta^{[k]}$. The E step finds the conditional expectation of the missing data, given the observed data y and the current estimated parameter θ and substitutes this value for the missing data. The M step then performs the maximum likelihood of θ. The EM algorithm alternates between performing the E step, which computes the expected value of the latent variables, and the M step, which computes the maximum likelihood estimates of the parameters given the data and setting the latent variables equal to their expectations. In this iterative two step procedure, the algorithm alternates between the E step and the M step until the parameter has converged or there is no change in the estimate.

Meng and Pedlow have shown that the range of problems that can be handled by the EM algorithm is very broad [6]. However, there are two major drawbacks to the EM algorithm. First, large fractions of missing information can dramatically lower the convergence rate. Next, the M step can be difficult (i.e. has no closed form), which prohibits the theoretical simplicity of the algorithm from converting to practical simplicity [18]. Despite its optimality, this technique, like the former, requires either prior knowledge that is often unavailable, or the estimation of underlying distributions, which can be computationally prohibitive, particularly for large dimensional datasets.

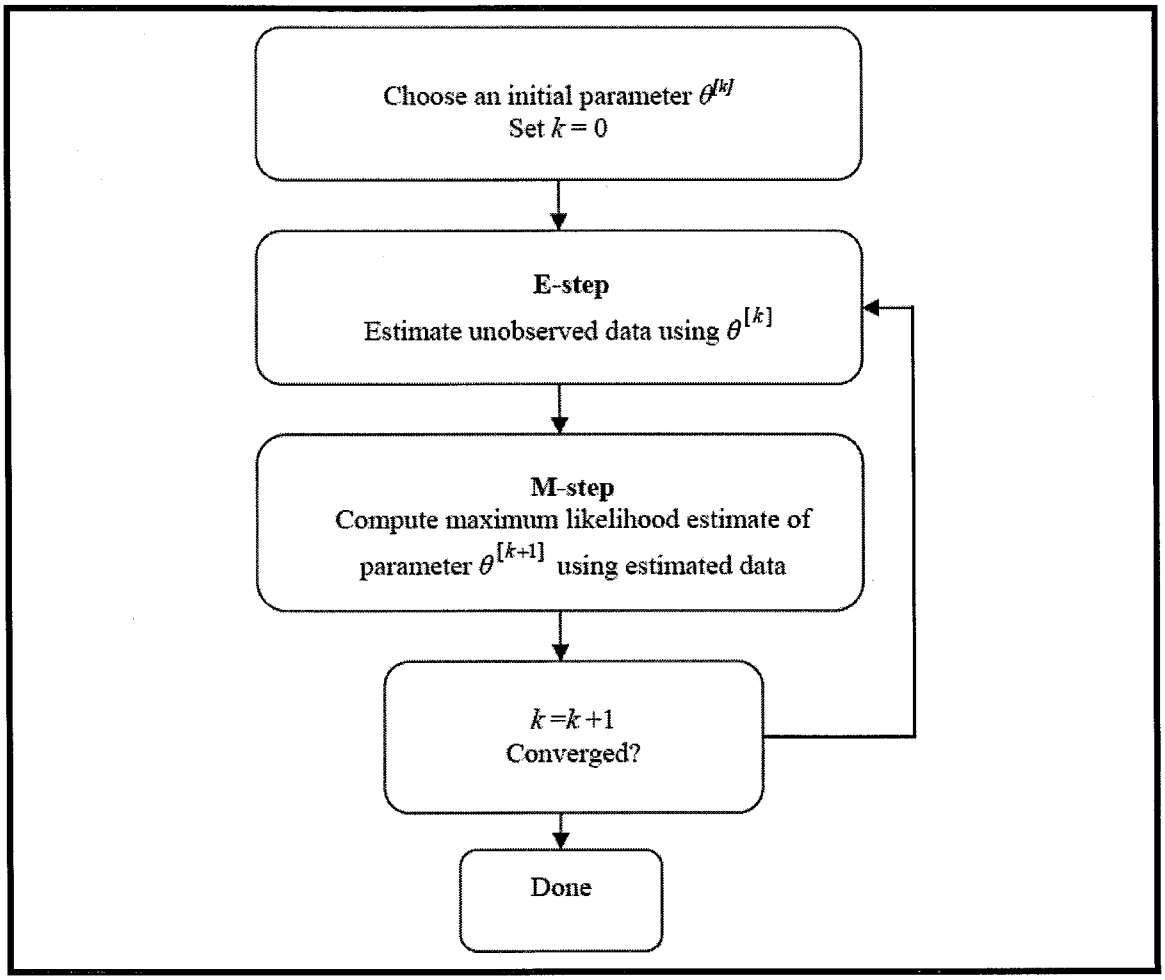

Figure 2.5: An overview of the EM algorithm

2.4 Feature Reduction

The problem with missing features deals with having a lack of features. However, Duda *et al.* has also shown that addition of features above a certain point may lead to a higher probability of error [7]. By selecting only discriminative features, one lowers the likelihood of having to deal with more missing features as the set of features under consideration is reduced. Therefore, an effective feature selection strategy is needed to reduce the dimensionality of the feature space and to identify the relevant features to be used for classification.

Another alternative for reducing the dimensionality of the dataset requires searching for an optimal subset of features so that fewer features are required [8]. This method may include evaluating every subset of features by training a classifier with a subset of the total available features and to select the subset that provides the best performance. While it is conceptually simple, optimal feature selection can be difficult, primarily because of its computational complexity. The optimization space of all the subsets of a given cardinality is subject to combinatorial explosion. An exhaustive search is often computationally intensive, making it impractical even for a relatively small number of features.

Search algorithms that avoid an exhaustive search include the depth-first search [9], breadth-first search [10], as well as the hill climb search [11]. Each algorithm comes with its own limitations. The Branch and Bound algorithm strives to reduce to the computational burden. However, the problem still remains if any of these optimal features are missing. Since the rest of the features (which may still be discriminative) are ignored, useful information hidden in these features is not taken into consideration. This may result in a poor performance on the selected feature subset.

2.5 Imputation Techniques

Many missing data methods fall under the general heading of imputation. The simplest method includes substituting the missing value with an educated guess and then to proceed as if there were no missing data. The assumption that one can impute a dummy or default value oversimplifies the problem and does not work well in general. Imputation often requires prior knowledge about the data distribution which is not readily or easily available.

A simple method used to replace missing values is the Nearest Neighbor Estimator. This method will find the nearest neighbor instance that includes no missing features. Secondly, it fills up missing values of this instance by corresponding values of the nearest neighbor instance. To estimate the distance between an instance containing missing values and other instances, one can use the Euclidean distance. A simple implementation of this can be found in below in Figure 2.6.

For each instance $\mathbf{x}_i$ *that has missing values*

For each instance $\mathbf{x}_j$ *that has no missing values*

Calculate Euclidean distance d_{ij} *between* $\mathbf{x}_i$ *and* $\mathbf{x}_j$

$d_{ij} = \sqrt{\sum_{k=1}^{f}(x_{ik} - x_{jk})^2}$, *where f is the number of non-missing features*

Decide the nearest instance $\mathbf{x}_n$ *and replace each missing value in* $\mathbf{x}_i$ *by the corresponding value in* $\mathbf{x}_n$

Figure 2.6: Pseudocode for Nearest Neighbor Algorithm

There are variations of the above mentioned method. Morin [12] proposed to replace the missing features by values of these features from their k nearest neighbors (KNN). Another KNN-based method selects instances with profiles similar to the instance(s) of interest to impute missing values, taking into account the similarity of the instances [13,14]. One attraction of these KNN-based techniques is that they consider the correlation structure of the data. However, a major disadvantage of this data imputation method is the time required for searching through the entire dataset looking for the most similar instances. The choice of k (i.e. the number of neighbors), is critical, and small

values of k have been known to produce a deterioration in the performance of the classifier. On the other hand, a large k may include instances that are significantly different from the instance containing the missing features. Hence, a large k can also severely affect the classifier performance.

Variations of imputation method also include the mean and mode substitution [15]. Like many of the imputation methods mentioned above, these also require that the available training data be sufficiently dense, a requirement often not satisfied for large dimensional data. Normally, mean imputation is used for attributes whose values are numeric and mode imputation is used for values that are symbolic. However, in a dense numeric distribution, one could also use the mode imputation technique for missing values. The substitution method artificially reduces the variance of the dataset. It also diminishes relationships with other variables. Hence, as the proportion of missing data increases, this method tends to produce biased estimates, so it generally should be avoided. We show the pseudocode for both the mean and mode replacement algorithms below.

For each numeric attribute x_i that has missing values

1. *Calculate the mean of the non-missing values*

   $$\mu = \frac{1}{m} \sum_{i=1}^{m} x_j$$

   where m is the number of non-missing values for that particular attribute
2. *Replace the missing value by the calculated value, μ*

Figure 2.7: Pseudocode for Mean Replacement Algorithm

For each symbolic attribute x_i that has missing values

a. Count the number of times each value appears
b. Replace the missing value by the value that appears the most (mode)

Figure 2.8: Pseudocode for Mode Replacement Algorithm

There are also implicit modeling techniques that can be used to handle missing data. These include hot deck [16] and cold deck [17] imputation methods. In hot deck imputation, the missing values are replaced by values from similar units in the sample, whereas in the latter, the missing value is replaced by a constant value from an external source, normally a previous data collection. The methods are attractive due to their simplicity. However, these methods may introduce bias and in the case of the latter method, a large variance.

2.6 Maximum Likelihood Estimators

There are also more classical methods that use linear regression to estimate substitutes for missing feature values [18,19]. Maximum Likelihood (ML) is a general approach to statistical estimation that is widely used to handle many otherwise difficult estimation problems. Maximum likelihood based methods assume that the value to be estimated comes from a distribution with a known form, but unknown parameters. ML estimators seek solutions that best explain the observed data. The problem then reduces to a function minimization/maximization problem, whose solution techniques are well known.

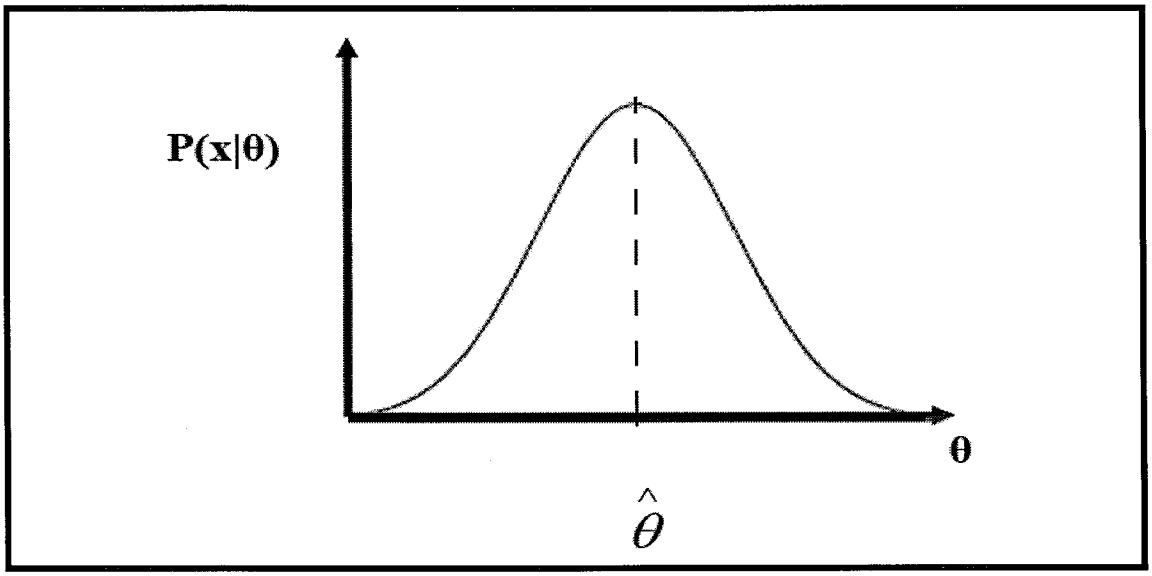

Figure 2.9: Maximum Likelihood Estimation

ML estimators have a number of desirable properties. Maximum likelihood methods are often preferred because they are computationally less intensive, and require merely differential calculus techniques (e.g. first and second order moments) or a simple gradient search. Under a fairly wide range of conditions, they are known to be asymptotically efficient. Efficiency implies that the true standard errors are at least as small as the standard errors for any other consistent estimator [20]. However, the former is only approximately true as approximation gets better with a denser distribution. Problems of ML estimation also include choosing the model family which often requires making assumptions about the model to begin with. Little [19] has shown that many of such methods are inconsistent as they are dependent on the data distribution. Hence if the model family is not chosen appropriately, even the best or maximized likelihood estimate will not fit the data well.

2.7 Decision Trees

Zhang *et.al* has studied the cost of missing values in cost sensitive decision trees [21]. He describes three methods; Known Value Strategy (KVS), Null Strategy (NS), Internal Node and Strategy (INS). The KVS was proposed in [22] and utilizes only the known values in the tree building for each test example. For each test example, a new decision tree is built from the training examples with only those attributes whose values are known in the test example. This strategy utilizes all known attributes, but it clearly avoids any missing data directly. It is a lazy tree method where a tree is built during test process. A major drawback of KVS is the relatively high computation cost as new decision trees may have to be built for different test instances that differ in their missing attributes. However, this is generally not a problem if the tree building algorithm is efficient. One can also develop a template for testing test examples with the same subsets of known attributes since the decision trees for the same subsets of known attributes are similar.

The NS assigns a special value, often called "null" in databases, to the missing value. The null value is then treated just as a regular known value in the tree building and test processes. There are multiple problems with this using this strategy. One drawback is that it does not utilize known values as missing values are treated equally as a known value. Furthermore, there can also be more than one situation where values are missing. Hence, replacing all missing values by a single value may not be adequate. In addition, the subtrees can be built under the "null" branch. This potentially suggests that the unknown values are more discriminating than the known values. Simply, it is inadequate to replace all missing values by a single value.

The INS method keeps examples with missing values in internal nodes and does not build branches for them during tree building. Instead, when the tree encounters an attribute whose value is unknown during testing, the class probability of training examples occurring at the internal node is used to classify it. There might be many different situations where the attributes are missing, warranting the classification to be dealt by the internal nodes. This strategy is efficient compared to the KVS as only one tree is built for all test examples.

2.8 One-Class Classifiers

Juszczak and Duin [23] have recently shown that missing data can be addressed by combining one-class classifiers (*occs*) trained on a single dimension feature. The approach is capable of handling any combination of missing features, with the fewest classifiers possible.

In one-class classification, one class of data, called the target class, has to be distinguished from the rest of the feature space. It is assumed that only examples of the target class are available. Objects not originating from the target set, by definition are referred to as outlier objects. A threshold is set on the tails of the estimated probability such that a specified amount of the target data is rejected. In one-class classification, the target class is modeled such that $P(x|\omega_{Tc})$ is the probability that object x belongs to target class.

Each classifier is trained on one attribute at a time belonging to the target class. Hence, the total number of classifiers that need to be trained is $C*F$, where F is the number of features and C is the number of classes. The authors use the one-class classifiers as base classifiers to combine using a variety of fixed combination rules which

include the *mean, product* and *max* rules. Equations 2.3-2.5 show to use the occs algorithm on the missing feature problem on the above mentioned combination rules, where F' is the number of available features.

$$\arg\max_{c}[\sum_{i=1}^{F'} P(x_i \mid \omega_{Tc})] \tag{2.3}$$

$$\arg\max_{c}[\prod_{i=1}^{F'} P(x_i \mid \omega_{Tc})] \tag{2.4}$$

$$\arg\max_{c}[\max_{i} P(x_i \mid \omega_{Tc})] \tag{2.5}$$

Since a single feature x_i is considered at a time during classification, feature interactions are neglected. This can lower the performance of the ensemble. The authors show that the classifiers that are trained on a single feature at a time may not carry enough discriminative information, so the algorithm may often fail to achieve satisfactory classification performance.

2.9 Neural Network Techniques

Gupta and Lam have studied the effect of the neural network weight decay method on data sets with missing values [24]. Similar to the previous techniques, they reconstruct missing values using three different methods: iterative multiple regression, mean and zero replacement. However, their experimental results also show that the higher the percentage of missing values within a dataset, the higher the differential effects from reconstruction methods.

Yoon and Lee provide another algorithm for MLPs to handle missing features. Their Training-EStimation-Training (TEST) algorithm is designed not to make any assumptions about the underlying distribution of the missing inputs [25]. Unlike other

training methods with missing data, it does not assume data distribution models. The authors have also shown that the TEST algorithm may not be appropriate for small training data.

Figure 2.10 provides the pseudocode for the TEST algorithm. This neural network algorithm uses back propagation algorithm to deal with missing values. Firstly, the network learns from instances that have no missing values. The network is trained to duplicate or replicate all of the inputs as outputs using back propagation. Secondly, during testing or validation, when the missing values are detected, the network is used in back propagation mode, so that the internal weights of the network are not adjusted. In this manner the error is propagated all the way back to the inputs. At the input level, an appropriate weight can be derived for the missing values so that it least affects or disturbs the internal structure of the trained network.

1. *Create a neural network with arbitrary weight.*

2. *For each instance that has no missing values x_{nm}, learn the network such that all the input values and output values are approximately the same by back propagation.*

3. *For each instance that has missing values x_m,*
    1. *Input values of the instance*
    2. *If the input values and outputs are not the same, repeat (a). Else the output values are used as input values*
    3. *Replace missing values by the output of the network*

Figure 2.10: Pseudocode for the TEST Algorithm

2.10 Ad-Hoc Model Approach

The original data split into smaller data set, according to the pattern of known features [26]. Each data set is then analyzed or learned as a complete data set. Predictions are then made using a model suitable to the test instance missing those features. Whilst this method is conceptually simple, it suffers from the combinatorial explosion of models to learn large feature spaces with many missing value patterns. At the same time, this approach faces the common problem of having insufficient data to reliably learn from. Furthermore, the question arises, how to combine models and predictions, if more than one model could be suitable for any given case.

CHAPTER 3 – BACKGROUND

This chapter is split into four major sections. Section 3.1 gives a general background on ensemble based systems. Section 3.2 gives a brief outline on the boosting based algorithm, AdaBoost.M1. Section 3.3 discusses the algorithm Learn^{++}, an incremental learning based algorithm motivated by the former algorithm. Section 3.4 describes the Random Subspace Method (RSM), a technique that is core to the algorithms introduced later in this thesis.

3.1 Ensemble System

The integration of ensemble systems to improve classification performance is currently an active research area in machine learning and pattern recognition communities. Ensemble systems, also known under various other names, such as multiple classifier systems (MCS), committee of classifiers or mixture of experts, ensemble systems have shown to produce favorable results compared to those of single expert systems for a broad range of applications, and under a variety of scenarios which include mail sorting, medical test reading and diagnostics, military target recognition, signature verification, to name a few [27]. Ensemble based systems lend themselves to other useful application, including data fusion, where the goal is to extract complementary pieces of information from different data sources, and make a more informed decision about the phenomenon generating the underlying data distributions [28].

3.1.1 Advantages of Ensemble Based Approaches

Many ensemble of classifiers based approaches have been well researched, and hence are now well-established, for improving classifier accuracy and robustness over single

classifier systems [29-32]. In addition to accuracy and precision, ensemble based systems have also been preferred over single classifier systems for several of other reasons. These include reduction of risks, allows taking advantage of the divide-and conquer paradigm, and the ability to process datasets of extreme sizes. Ensemble based systems introduce efficiencies that often cannot be achieved by single classifier systems. For example, the divide and conquer paradigm allows a complex problem to be broken or decomposed into several smaller problems to be handled.

3.1.2 Diversity of Ensemble Based Approaches

The premise of ensemble based approaches is that if several classifiers with sufficient diversity are trained on a given dataset, each will tend to make different errors, and combining these classifiers can reduce the overall error especially if the classifiers make complementary mistakes, the use of multiple classifiers will improve the classification accuracy with respect to that of individual classifiers [33].

Both theoretical and empirical research have demonstrated that an ensemble is able to achieve better performance if the base classifiers in it are both reasonably accurate and tend to err in different parts of the instance, and more recently feature space with regards to misclassified instances. There is much literature that relates diversity to classifier performance [34-37]. Several measures have been defined to quantitatively measure and assess diversity. These include the pairwise, entropy measure, kappa statistic, and Kohavi-Wolpert variance amongst many others [27].

The key in creating a good ensemble is widely recognized as ensuring that the individual classifiers are as diverse as possible. Diversity can be achieved in one of several ways. The most popular methods include using different training datasets to train

individual classifiers to create an ensemble. Bootstrapping and Bagging are often used to draw samples, usually with replacement from the training dataset. These techniques are particularly appealing when the dataset is of limited size.

Figure 3.1 shows the data instances of three classes in a two dimensional space represented as squares, circles and triangles. A random and overlapping subset of the training instance is drawn (indicated by shaded instances). The purpose of drawing subsets is to create diversity such that each classifier now learns a different decision boundary. A strategic combination of these classifiers often produces a more accurate classification than any of the base classifiers in the ensemble.

However, if we had access to a perfect base classifier, we would not have to resort to ensemble based-approaches. In reality, the presence of noise and outliers can and often impair the judgment of classifiers. Hence, by combining the decisions of various diverse classifiers, the ensemble is able to average out inherent noise and able to generate a more accurate decision boundary than a single classifier on its own. If the subset is drawn without replacement, the procedure is known as *jackknife* or *k-fold data split*. Here, the entire dataset is split into k blocks, and each classifier is trained on only a subset of these k blocks, usually, k-1 of them. A different subset of k blocks is selected for each classifier. The procedure for k-fold data splitting, also known as *leave-one-out* is illustrated in Figure 3.2.

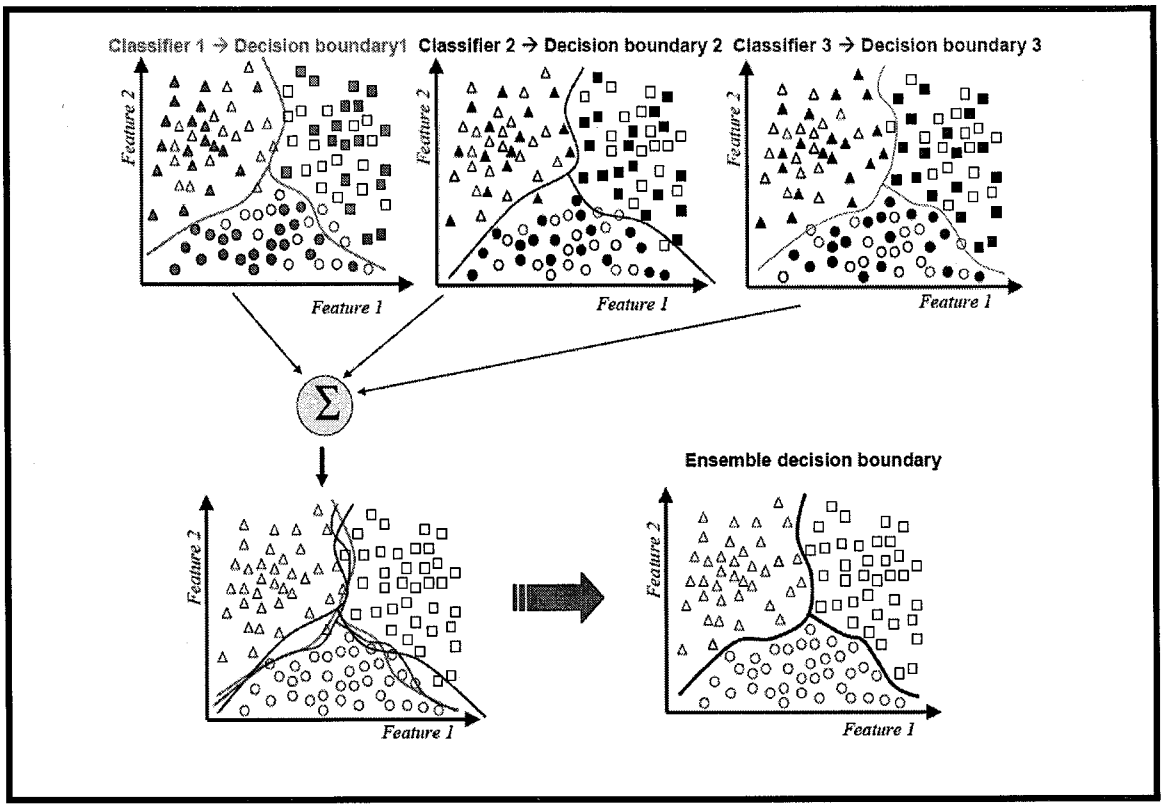

Figure 3.1: Combining classifiers that are trained on different subsets of the training data

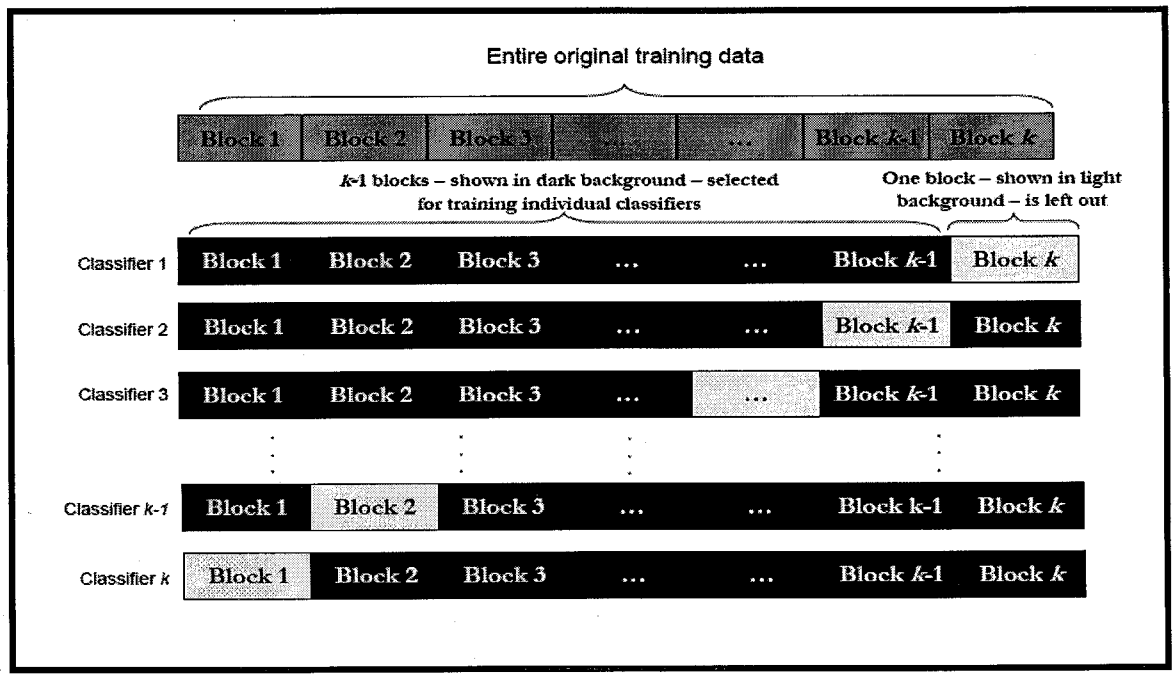

Figure 3.2: *k*-fold data splitting for generating different, but overlapping, training datasets

Diversity can be also be aided by having classifiers of varying weakness within the ensemble. In this case, diversity can also be achieved by using different training parameters for different classifiers, e.g., using different weight initializations, number of layers/nodes, error goals, etc. as in the case for training a series of multilayer perceptron (MLP) neural networks. These free parameters allow one to control or vary the degree of the stability and weakness of the classifiers. Decision trees and neural networks can be viewed as good candidates for this purpose, as their weakness can be controlled by the selection of their free parameters. For example, an MLP with fewer hidden layer nodes and a larger error goal is weaker than the one with more hidden layer nodes and a smaller error goal. A further benefit of ensembles is that they may not have to be limited to a certain architecture or model. While diversity can also be achieved by using entirely different types of classifiers, such as combining MLPs, decision trees, nearest neighbor classifiers and support vector machines, using different models or even different architectures of the same model may only be suited for specific applications that warrant them.

A number of combination techniques have been studied to improve the performance by creating diversity within an ensemble. Amongst these ensemble creation techniques, Bagging, Boosting and the Random Subspace Method (RSM) have enjoyed more attention [38,39].

In brief, the diversity of an ensemble may be achieved by, training weaker classifiers using different random subsets of the training data, using different types of classifiers, varying classifier parameters, or simply using different subsets of features

among individual classifiers. Hence, employing such techniques allows further optimization of ensemble based systems.

3.2 AdaBoost.M1

Boosting is defined as a process of producing a very accurate prediction rule by combining rough and barely accurate rules-of-thumb [27]. It began as a technique for combining trained classifiers with unique sets of strengths and weaknesses. Schapire has shown that for a two class problem, a weak learner that almost achieves high errors can be converted to a strong learner [40]. Schapire and Freund later developed AdaBoost.M1 extending boosting to handle multi-class learning problems [42]. Since then, AdaBoost.M1 has emerged as one of the most successful ensemble algorithms.

AdaBoost.M1 and its multiple variations have enjoyed much success in a wide variety of pattern recognition and machine learning problems. Several variations of the original AdaBoost algorithm have become popular in recent literature. This even includes the Boosting Feature Selection (BFS) algorithm [41], AdaBoost.R [42], AveBoost [43], and Online Boosting [44].

3.2.1 AdaBoost.M1 Explained

AdaBoost.M1 was developed to boost the performance of weak learner classifiers by generating various weak classification hypotheses, and combining them through weighted majority voting of the classes predicted by the individual hypotheses.

The algorithm maintains a weight distribution on all the instances in its training set. A hypothesis is generated by training the classifier using different subsets of the training data. Higher weights are given to those instances that are misclassified and lower

weights are given to those instances that are classified correctly by the current classifier. In this manner, AdaBoost.M1 assigns appropriate weights to each instance such that it is forced to focus its attention on the difficult pasts of the instance space.

At the end of a predetermined number of iterations, AdaBoost.M1 combines all the weak hypotheses generated thus far, through weighted majority voting, and outputs the label that maximizes the sum of the weights of the weak hypotheses predicting that label. Unlike Bagging or Boosting, AdaBoost.M1 uses a rather undemocratic voting scheme, called *weighted majority voting*. AdaBoost.M1 believes that classifiers that have shown good performance during training should be rewarded with higher voting weights than the others. A conceptual overview of AdaBoost.M1 is shown in Figure 3.3 below.

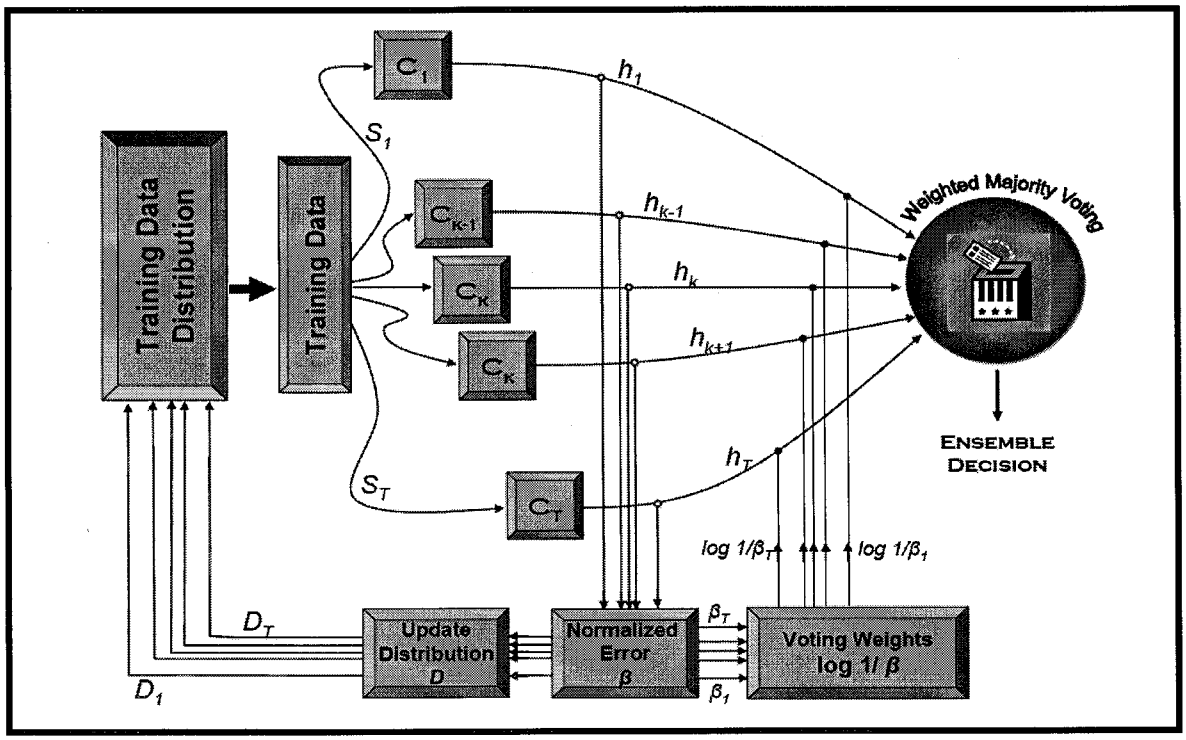

Figure 3.3: Conceptual overview of Algorithm AdaBoost.M1

3.2.2 *Advantages of AdaBoost.M1*

One of the main attractions of AdaBoost.M1 is its ability to decrease the ensemble training error very quickly. Further experiments with AdaBoost show that the testing error continues to decrease with the addition of more classifiers even after the training error reaches zero.

Another desirable property of AdaBoost is its ability to identify *outliers*. This is because AdaBoost focuses its weight on the hardest examples, and very often the examples with the highest weight turn out to be outliers.

AdaBoost.M1 and its variations are often fast, simple and easy to program. It has few parameters to tune. AdaBoost does not require prior knowledge about the weak learner and hence can be easily combined with any method for finding weak hypotheses. This is often adequate when the weak learner is strong enough to achieve reasonably high accuracy. However, this method fails if the weak learner cannot achieve at least 50% accuracy when run on hard distributions.

3.2.3 *Disadvantages of AdaBoost.M1*

The main disadvantage of AdaBoost.M1 is that it is unable to handle weak hypotheses with scaled error ε, greater than 0.5. However, AdaBoost.M2 removes this requirement. The expected error of a hypothesis is $1 - 1/C$, where C is the number of classes or labels. Thus, for $C = 2$, the weak hypotheses need to be better than random guessing. However, the requirement that the error be less than 0.5 is quite strong and may often be hard to meet when $C > 2$ [32].

3.3 Learn^{++}

Learning from new data without forgetting prior knowledge, and without requiring access to the original data is typically referred to as *incremental learning*. The algorithm Learn^{++} was inspired by the former boosting algorithm AdaBoost.M1 and developed for solving incremental learning problems, where the algorithm learns from new data, even when new data introduce instances from previously unseen classes. Many typical algorithms are not designed to accommodate new data and discard their existing set of classifiers. Hence, by losing their previously acquired knowledge at the expense of trying to learn from the new data, many of these algorithms face the plasticity-stability dilemma [45,46]. Previous experiments have shown that Learn^{++} places itself very favorably on the stability-plasticity spectrum [47]. It is able to achieve this because it exploits the synergy of an ensemble of classifiers to incrementally learn additional information from new data [48,49,50].

3.3.1 Learn^{++} Explained

The pseudocode of Learn^{++} is shown in Figure 3.4 and it is described in the following paragraphs below. Inputs to Learn^{++} are (i) the training data $\boldsymbol{S_k}$ of m_k samples drawn from the current database $\boldsymbol{DB_k}$; (ii) a supervised learning algorithm **BaseClassifier;** (iii) and an integer T_k, specifying the number of classifiers to be generated for database $\boldsymbol{DB_k}$. Learn^{++} generates an ensemble of classifiers using different subsets of each training data, $\boldsymbol{S_k}$. A weight distribution $\boldsymbol{D_t}$ is initialized to be uniform, giving each instance an equal likelihood of being selected into the first training subset $\boldsymbol{TR_1}$.

In step 1 of each iteration t, the distribution $\boldsymbol{D_t}$ is obtained by normalizing the weights $\boldsymbol{w_t}$ of the instances updated based on their classification by the previous ensemble. In step 2, a subset of current dataset $\boldsymbol{S_k}$ is then drawn according to $\boldsymbol{D_t}$ to obtain a training data subset, $\boldsymbol{TR_t}$. Learn^{++} calls the **BaseClassifier** in step 3 and trained with $\boldsymbol{TR_t}$. The **BaseClassifier** can be any weak supervised learning algorithm and its goal is to generate a hypothesis h_t, which minimizes the training error. A hypothesis h_t is obtained on the current training subset $\boldsymbol{TR_t}$ and its error ε_t, is calculated in step 4. Learn^{++} requires $\varepsilon_t < ½$ for each hypothesis, h_t. If $\varepsilon_t > ½$, the current hypothesis h_t is deemed too weak, so it is discarded and the algorithm returns to step 2 where is it is replaced with a new h_t, generated from a new training subset $\boldsymbol{TR_t}$. If $\varepsilon_t < ½$, the scaled error of h_t is calculated and all hypotheses generated during the previous t iterations are combined, using weighted majority voting, to construct the *composite hypothesis* H_t on all instances in the current dataset $\boldsymbol{S_k}$ in step 5. The composite error E_t made by H_t is determined in step 6 by adding the distribution weights of all instances misclassified by the current ensemble. The normalized composite error, $\boldsymbol{B_t}$ is computed in step 7 and used in updating the weights $\boldsymbol{w_t}$, which are then used in computing the next distribution $\boldsymbol{D_{t+1}}$. Once T_k hypotheses are generated for each database $\boldsymbol{DB_k}$, the final hypothesis H_{final} can be obtained by weighted majority weighting.

For each new database that becomes available, Learn^{++} first reinitializes the distribution to be uniform, then evaluates the current ensemble on the new training data $\boldsymbol{S_{k+1}}$, by jumping to Step 6 of the inner loop, and updates the distribution based on the performance of the current ensemble. This allows the algorithm to focus on the novel instances from the new database, especially if it introduces new classes as the current

ensemble will not be likely to classify the new classes, causing them to misclassify those instances. This causes the weights of those instances to be increased giving them a higher likelihood of being included into the next training subset.

Inputs: For each dataset drawn from DB_k $k=1,2,\ldots,K$

- Sequence of m_k examples $S_k=\{(\mathbf{x}_i,y_i) \mid i=1,\ldots,m_k\}$,
- Supervised learning algorithm **BaseClassifier**
- Integer T_k, specifying the number of iterations

Do for each database DB_k, $k=1,2,\ldots,K$:

Initialize $w_1(i)=D_1(i)=1/m_k$, $\forall i$, $i=1,2,\cdots,m_k$

If $k>1$,

Go to Step 6, evaluate current ensemble on new data set S_k, and update weight distribution;

End If

Do for $t = 1,2,\ldots,T_k$:

1. Set $D_t = \mathbf{w}_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that D_t is a distribution
2. Draw a training subset TR_t from the distribution D_t.
3. Call **BaseClassifier** to be trained with TR_t.
4. Obtain a hypothesis h_t and calculate its error

   $$\varepsilon_t = \sum_{i:h_t(\mathbf{x_i})\neq y_i} D_t(i) \quad \text{on } S_k.$$

   If $\varepsilon_t > 1/2$, discard h_t, go to step 2.

   Otherwise, compute scaled error as $\beta_t = \varepsilon_t/(1 - \varepsilon_t)$
5. Call weighted majority, obtain composite hypothesis H_t
6. Compute the error of the composite hypothesis

   $$E_t = \sum_{i:H_t(\mathbf{x}_i)\neq y_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)\left[\!\left[H_t(\mathbf{x}_i) \neq y_i \right]\!\right]$$
7. Set $B_t = E_t/(1-E_t)$, and update the weights:

   $$D_{t+1}(i) = D_t(i) \times \begin{cases} B_t, & \text{if } H_t(\mathbf{x}_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$

Call majority voting and output the final hypothesis.

$$H_{final}(x) = \arg\max_{y\in Y} \sum_k \sum_{t:h_t(x)=y} \log\frac{1}{\beta_t}$$

Figure 3.4: Pseudocode of Algorithm Learn^{++}

3.3.2 *Summary of Major Differences between AdaBoost.M1 and Learn^{++}*

Both AdaBoost.M1 and Learn^{++} are very similar in implementation with the exception of two major details. First, AdaBoost.M1 was introduced as a boosting algorithm to handle multi-class distributions and never intended for incremental learning problems. Therefore, it does not introduce and handle new classes like Learn^{++}. Second, AdaBoost.M1 uses the performance of the current single hypothesis to update its weight distribution causing AdaBoost.M1 to focus on other *difficult to learn* instances, potentially including outliers. Learn^{++} on the other hand, uses the ensemble performance through composite hypothesis. This allows a more efficient incremental learning ability, as the algorithm can now learn from the *novel* instances introduced from new classes.

3.3.3 *Recent Advances in Learn^{++}*

While the native combination rule for both Learn^{++} and AdaBoost.M1 has been weighted majority voting, [51] have studied the effects of using various combination rules, on both Learn^{++} and AdaBoost.M1 in an incremental learning setting. The experiments included use of simple majority voting, weighted majority voting, sum rule, median rule, product rule and decision template. [51] conclude that while it has been well established that the choice of combination rules often is application specific, it is also the case for the datasets, even in an incremental learning setting. However, the experiments carried out by [51] do show that even in an incremental setting, simple majority and weighted majority voting schemes often perform better than their counterpart rules.

3.3.4 Problems of Learn^{++}

Learn^{++} has been shown to work rather well on a variety of real world problems and applications. However, despite its successes, it has faced certain optimization problems. This includes the relatively large number of classifiers required for learning instances coming from new classes. When a new dataset introduces a previously unseen class, new classifiers are trained to learn from the new class; however the existing classifiers continue to misclassify instances from the new class. Therefore, the decisions of the former classifiers trained on previous classes tended to outvote the decision of the newer classifiers. Hence, the original Learn^{++} tended to suffer from classifier proliferation; a sufficient number of new classifiers often needed to be generated to recognize the new class before the ensemble decision was effective.

[52] have recently presented a modified version of the predecessor algorithm Learn^{++}. The novelty of the new algorithm, Learn^{++}.MT is its use of preliminary confidence factors and class specific performance (CSP) in assigning voting weights, based on cross-referencing the classes that have been seen by each of the classifier during training. The modified approach overcomes the outvoting problem inherent in the original Learn^{++}. The strong success of CSP in Learn^{++}.MT has motivated its use in Learn^{++}.MFv2.

3.4 Random Subspace Method

An ensemble is generally more accurate than any of the base classifiers in the ensemble. Both theoretical and empirical research has shown that an effective ensemble should consist of base classifiers that have high classification accuracy, and have the ability to make different errors in the instance space. One approach for generating an ensemble of

diverse base classifiers is the use of different feature subsets, or *ensemble feature selection* [53]. Classifiers trained on different subsets of the feature subspace will tend to err in different sub-domains of the feature space [54].

Ho introduced the Random Subspace Method (RSM) [55,56] as an efficient approach for ensemble feature selection. The RSM has much in common with both bagging and boosting, but instead of sampling different instances, the RSM samples from the feature space. The RSM can be described as a procedure that randomly selects a smaller number of dimensions from a higher dimensional feature space. Skurichana points out that this method has been found to work well when there is redundant information that is "dispersed" across all the features rather than concentrated in a subset of them [57]. RSM-inspired methods have been found to do well in a variety of applications [58,59,60].

3.4.1 Random Subspace Method Described

We briefly describe the RSM algorithm in this section. Let each training object X_i (i = $1,\ldots, m$) in the training sample set $X = (X_1, X_2, \ldots, X_m)$ be a d-dimensional vector $X_i^d = (x_{i1}, x_{i2}, \ldots, x_{id})$. In the RSM, d' features are randomly selected, where $d' < d$. The modified training object becomes $X_i^{d'} = (x_{i1}, x_{i2}, \ldots, x_{id'})$. In this manner, the RSM constructs T classifiers each trained on d' features, where T is predefined or decided ahead of time. The original algorithm is shown in Figure 3.5.

Training

Repeat for $t = 1, 2, \ldots T$:

a. Select $d' < d$ features randomly from the original feature space d.

b. Construct a classifier using all m instances, $X = (X_1, X_2, \ldots, X_m)$, using only d' features

Testing

For all instances $i = 1, \ldots, m$:

Combine hypotheses of classifiers $h_t(x)$, $t = 1, 2, \ldots T$, by simple majority voting

$$H_i(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} 1$$

Figure 3.5: Pseudocode of Algorithm Random Subspace Method

3.4.2 Advantages of RSM

The RSM has several desirable properties. The large dimensionality of the feature space is often difficult for conventional multivariate search techniques. Hence, Ho proposes a different approach. Instead of finding the most useful or discriminative features, RSM simply builds random subsets of the feature space.

At the same time, the RSM can increase the small training sample size by constructing classifiers from various subsets. This can be particularly useful when the number of training instances is relatively small compared to its dimensionality. While the training sample size remains constant, the RSM creates classifiers in a subspace dimensionality lower than the original feature space. Therefore, the relative training sample size increases as the RSM takes advantage of the possible subsets within the training sample. Hence, RSM relatively increases the size of the training samples.

Data received may often have redundant features. The RSM selects features based on a stochastic approach. Data in a high dimensional space can be exploited more effectively if they contain redundant features [61]. Hence, one may obtain better classifiers trained in random subspaces than in the original feature space.

The RSM originally proposed by Ho uses majority voting. However, the RSM is not limited to simple majority voting and can use more sophisticated combination rules as investigated by Tsymbal [62]. The combined decision of such classifiers may be superior to a single classifier constructed on the original training set in the complete feature set [38]. Ho [55] shows that while most other classifier methods suffer from the curse of dimensionality, the RSM embraces and takes advantage of the high dimensionality.

Another key attraction of the RSM is its raw simplicity. It does not require any special bookkeeping or computation. Despite its simplicity, the combination of multiple random subset selections has been shown to contain sufficient discriminative information [63].

The RSM may also be used in collaboration with other well known techniques. Wang and Tang have extended their work to include a random sampling based LDA method for high dimensional data classification [64].

Ho [55] has also shown that simple random selection of features subsets are an effective techniques for ensemble feature selection. She accredits the effectiveness of the RSM because the lack of accuracy is compensated by the diversity within the ensemble created through random subsets. There are also other potential benefits aside from increased accuracy performance [65]. Random subspaces, which also require fewer

attributes, utilize lesser memory because only the chosen percentage of features needs to be stored.

Bagging, Boosting and the RSM have been theoretically and experimentally investigated and compared to each other. The performance of Boosting and Bagging is known to be dependent on a large training sample size since they both require a better representation of the distribution of the data classes to distinguish among them well. While these three ensemble creation techniques have been found to beneficial for regression, classification trees and perceptrons, Skurichina and Duin have shown that the RSM is able to outperform the former two algorithms since it is able to take advantage of the low cardinality of the data, creating weak and diverse classifiers in random subspaces [38].

CHAPTER 4 – APPROACH

This chapter is split into four sections. Section 4.1 provides general information that is central to both algorithms described in this thesis. Section 4.2 formally introduces and provides a detailed outline of the algorithm Learn^{++}.MF. Section 4.3 provides the motivation for Learn^{++}.MFv2 and provides the rationale for the modifications introduced in Learn^{++}.MFv2. Section 4.4 introduces the algorithm Learn^{++}.MFv2 along with a detailed description of its modifications.

4.1 Preliminaries

The proposed algorithms make two basic assumptions: First, the feature set may contain attributes that are irrelevant, or less informative than others. Second, the redundancy must be distributed randomly over the feature set. Hence, a dataset with consecutive feature values that are strongly correlated with each other, as they are in time series data, do not meet these requirements. These assumptions are primarily due to the random nature of feature selection, and are shared with all RSM based approaches. There are many applications where the data include redundant features, whose identities are unknown, and that are not tied to each other through a time series function, or better yet, that are in fact at least reasonably class-conditionally independent. The proposed approaches are designed for such applications.

Both algorithms use an ensemble of classifiers approach to classify data with missing features. In order to keep track of which classifiers are used in classifying any given instance, we define the *universal set* and *usable set* of classifiers. The universal set of classifiers includes all classifiers that have been generated thus far. The usable set of

classifiers is the instance specific set of actual classifiers that can be used in identifying the given instance. Similarly, we also define the set of *unusable* classifiers, which for any given instance, are those classifiers that require the features missing in the given instance. We show empirically that if U classifiers yield a certain classification performance on data with no missing features, then a similar performance can be achieved simply by generating additional weak classifiers to obtain a total of U *usable* classifiers on data that have missing features. The assumptions for this property to hold are the availability of a dataset with sufficient redundancy, and a set of weak classifiers each of which is trained with slightly different parameters.

4.2 Learn^{++}.MF

Learn^{++}.MF is a modification of the original Learn^{++} algorithm. While Learn^{++}.MF does not deal with incremental learning, it has its roots in ensemble learning. Learn^{++}.MF was designed specifically for the missing feature problem. Learn^{++}.MF combines an ensemble of classifiers approach with random feature selection to classify data with missing features [66]. It has been inspired in part by the Random Subspace Method (RSM), where an ensemble of classifiers are trained using random subsets of the features to improve the diversity of the ensemble to aid in its generalization performance, or in selection of optimal features [8].

Learn^{++}.MF takes advantage of the *instability* of weak classifiers. Using an ensemble of weak classifiers approach has additional benefits. First, the training time is often less for generating multiple weak classifiers compared to training one strong classifier. Strong classifiers spend a majority of their training time in fine tuning the desired decision boundary, whereas weak classifiers completely skip the fine-tuning stage

as they generate a rough approximation of the decision boundary. Furthermore, weak classifiers are also less likely to suffer from *overfitting* problems, since they avoid learning outliers, or quite possibly a noisy decision boundary. A strategic combination of these classifiers then reduces the individual errors.

While using a smaller subset of features for a classification problem is not new, the feasibility of this strategy on the missing features problem has been mostly unexplored, and constitutes the main focus for this initial effort. The basic idea in the proposed approach is to generate a sufficiently large number of classifiers, each trained with a randomly selected subset of the features. When an instance **x** with missing feature(s) needs to be classified, only those classifiers trained with the features that are presently available in the given instance **x** are used to determine the correct classification.

A key difference of Learn^{++}.MF from many of the techniques mentioned earlier is that Learn^{++}.MF tries to make the most of the existing features, instead of trying to estimate or impute the values of the missing ones. Hence, it does not introduce the biases many of the previously mentioned algorithms often introduce.

As mentioned above, the Learn^{++}.MF algorithm uses an ensemble of classifiers, each of which is trained on a random subset of the entire feature space. While it is not essential to the algorithm, we initially assume that the training data has no missing features, and /or there is sufficient training data with all its features intact. The algorithm focuses on the more commonly seen, and potentially more annoying case of field data containing missing features.

The pseudocode and block diagram of the Learn^{++}.MF algorithm are provided in Figure 4.1 and Figure 4.2 respectively. The inputs to the algorithm are (1) the training

data set $\boldsymbol{D}$; (2) the number of features, *nof*, to be used for training individual classifiers; (3) a supervised classification algorithm (**BaseClassifier**), and the number of classifiers to be created T; and (4) the sentinel value *sen* to designate a missing feature. The data set $\boldsymbol{D}$ contains m instances, each with f number of features. At each iteration t=1,...,T, the algorithm creates an additional classifier C_t. The individual features to be used with each classifier is randomly drawn from an iteratively updated distribution $\boldsymbol{P}_t$ that ensures that each classifier is as diverse as possible with respect to the feature combinations selected. Specifically, at iteration t, a subset of features, $\boldsymbol{F}_{selection}(t)$, is drawn according to $\boldsymbol{P}_t$, such that those features with higher weights are more likely to be selected. These features are then used in training current classifier, C_t. $\boldsymbol{P}_1$ is initialized to be uniform, so that each feature has equal initial likelihood of being selected into $\boldsymbol{F}_{selection}(1)$. This distribution is very similar to the approach taken by AdaBoost.M1 and Learn^{++}. Both AdaBoost.M1 and Learn^{++} maintain a distribution to keep track of instances, whereas in this effort the distribution is used to keep track of features.

The block diagram of Learn^{++}.MF is given in Figure 17, and described in detail below. For each iteration, $\boldsymbol{P}_t$ is first normalized to obtain a legitimate distribution (step 1).

$$P_t = P_t \Big/ \sum_{j=1}^{f} P_t(j) \tag{4.1}$$

Next, *nof* (number of features) features are randomly drawn from $\boldsymbol{P}_t$ (step 2) which constitute the set $\boldsymbol{F}_{selection}(t)$. The t^{th} classifier C_t is trained (step 3) using the features in $\boldsymbol{F}_{selection}(t)$ and tested on training data (step 4). The t^{th} classifier C_t must achieve a minimum of 50% correct classification on its training data to ensure that it has a meaningful classification capacity.

TRAINING

Inputs:

- Sentinel value *sen*.
- **BaseClassifier** and the number of classifiers, T.
- Training data set $D=\{(\mathbf{x}_i, y_i) \mid i=1,\ldots,m\}$ with m instances and f features.
- Number of features used to train each classifier, *nof*.

Initialize $P_1(j) = 1/f, \forall j, j = 1, \cdots, f$

Do for $t = 1, 2, \ldots, T$:

1. Set $P_t = P_t / \sum_{j=1}^{f} P_t(j)$ so that P_t is a distribution.
2. Draw *nof* features for $F_{selection}(t)$ from P_t.
3. Call **BaseClassifier** to generate a weak classifier using only those features in $F_{selection}(t)$ for each $\mathbf{x}_i$.
4. Obtain a hypothesis C_t and its performance $Perf_t$ on D. If $Perf_t < 50\ \%$, discard C_t and go to step 2.
5. Set $P_t(F_{selection}(t)) = P_t(F_{selection}(t)) \cdot (1/\lambda),\ 0 < \frac{1}{\lambda} < 1$

end loop

VALIDATION/TESTING

Do for each validation instance $\mathbf{x}_i$

1. $M_{feat}(i) = \arg(\mathbf{x}_i(j) == sen),\ \forall j, j = 1, \cdots, f$.
2. $c(i) = \arg\max_{y} \sum_{t:C_t(\mathbf{x})=y} [\![M_{feat}(i) \notin F_{selection}(t)]\!]$.

end loop

Figure 4.1: Pseudocode of Algorithm Learn^{++}.MF

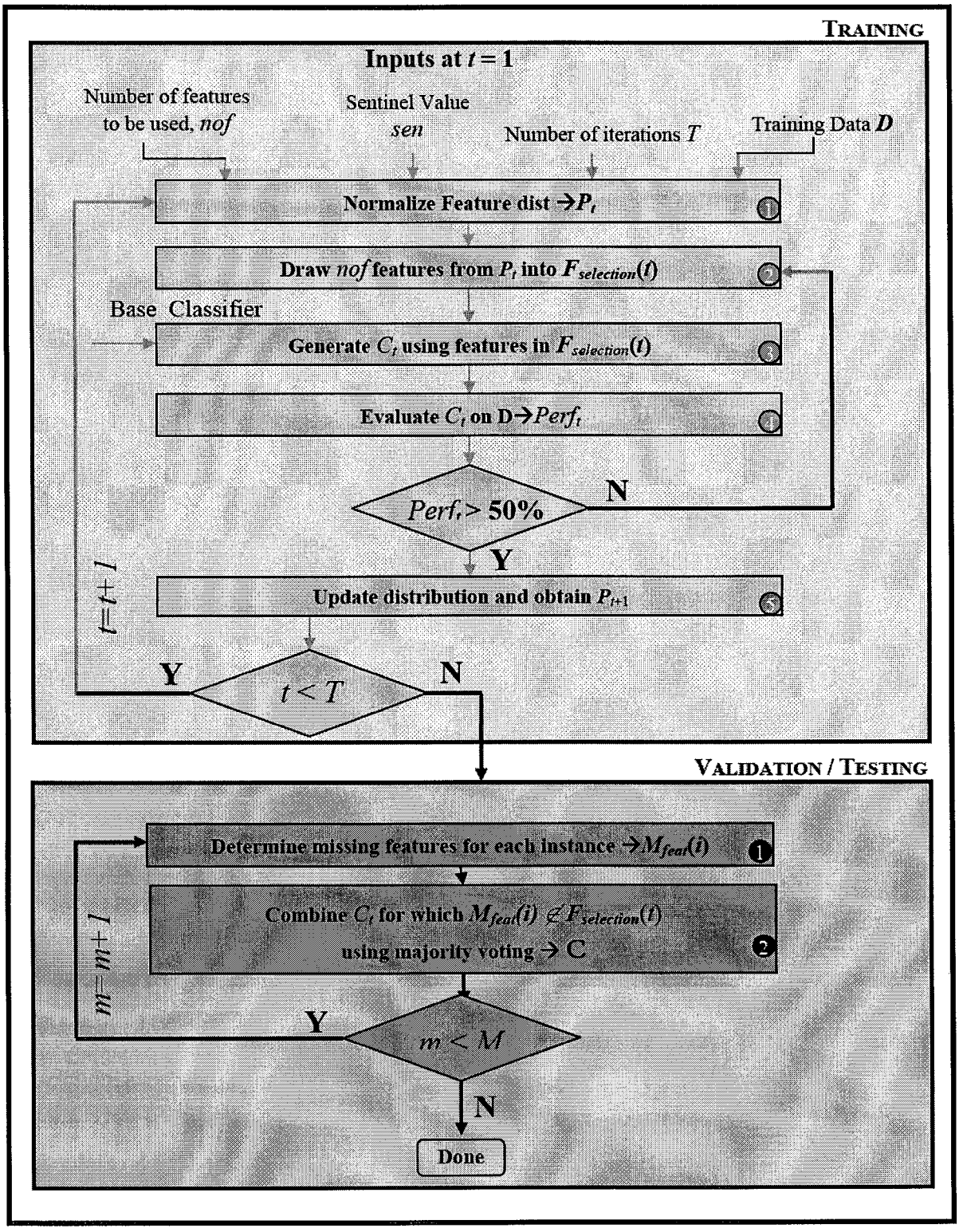

Figure 4.2: Block Diagram of Algorithm Learn^{++}.MF

The distribution $\boldsymbol{P}_t$ is then updated (step 5) according to

$$P_t(F_{selection}(t)) = P_t(F_{selection}(t)) \cdot \frac{1}{\lambda} \tag{4.2}$$

such that the weights of those features that appear in the current $\boldsymbol{F}_{selection}(t)$ are reduced by a factor of λ, where $0 < \frac{1}{\lambda} < 1$. Those features not in the current selection effectively receive higher weights when $\boldsymbol{P}_t$ is normalized again in step 1 of next iteration. This strategy helps ensure that the individual classifiers are trained with as diverse features as possible. While the algorithm does use the notion of random subspacing for its features, the selection of features that Learn^{++}.MF takes is not entirely random. Learn^{++}.MF selects its features according to the feature distribution, $\boldsymbol{P}_t$. The motivation of this feature distribution is meant to ensure that the features selected are as diverse as possible. However, it does not mean the features that have been selected in the previous trial have no chance of being selected by the current trial. Instead, they have a lesser likelihood of being selected. Hence, the subspacing selection method of Learn^{++}.MF can be viewed as an autonomous, pseudorandom procedure.

During the validation phase, the algorithm searches for sentinels, the place holders for missing data. To ensure that actual values are not mistaken for the sentinel, *sen* should be chosen as a value not expected to occur in the data. All features *j, j=1,...,f* with a sentinel value in the given instance are then flagged and placed into the set of missing features $\boldsymbol{M}_{feat}(i)$ for that instance $\mathbf{x}_i$. Finally, all classifiers C_t whose feature selection list $\boldsymbol{F}_{selection}(t)$ did not include those in $\boldsymbol{M}_{feat}(i)$ (that is, *usable classifiers* that did not use any of the features in $\boldsymbol{M}_{feat}(i)$) are combined through majority voting to determine the classification of instance $\mathbf{x}_i$. This constitutes the ensemble classifier $C(i)$ for $\mathbf{x}_i$:

$$c\ (i) = \arg\max_{y} \sum_{t:C_t(\mathbf{x})=y} [|\, M_{feat}(i) \notin F_{selection}(t)\,|] \tag{4.3}$$

where $[|\,.\,|]$ evaluates to 1, if the predicate is true, and zero otherwise. We note that the number classifiers T should be chosen large enough to create adequate number of usable classifiers for the problem at hand, as described in results.

4.3 Motivation for Learn^{++}.MFv2

Learn^{++}.MF works well in a variety of applications with missing features as discussed in detail in the Results section [66,67]. However, there is room for improvement and Learn^{++}.MF can be further optimized to boost and improve its performance.

In the current algorithm Learn^{++}.MF, the classifiers are combined through a simple majority voting method, where each classifier votes on the class it predicts, and the class that receives the most votes is chosen as the final classification. Voting schemes such as simple majority voting do not take into account the local expertise of the base classifiers. This approach can be viewed as being sub-optimal, however, as not all classifiers perform equally well [62]. In other applications that use ensemble approaches, it has been well established that a weighted majority voting scheme can improve the performance of the ensemble if the voting weights are properly chosen.

Brodley and Lane [68] have studied various aspects of creating effective ensembles. Also, they show that increasing the coverage of an ensemble through diversity is not enough to ensure increased prediction accuracy if the integration method does not utilize the coverage. Thus, while it has been established that diversity and coverage are often pertinent conditions for ensemble accuracy, it is important for the ensemble to have a good integration method that will further utilize the diversity of the base classifiers.

Hence, we look for a possible integration method that may take advantage of the diversity and coverage.

Such an integration method can come from confidence estimation and combination rules such as weighted majority voting. The goal of ensemble combination rules is to maximize the useful information provided by all classifier outputs in such a way that the correct decisions are amplified and the incorrect ones are cancelled out. Properties of different combination rules have been well researched [69,70] within the context of improving generalization performance of an ensemble system. In any combination rule, the final decision is the class that receives the largest support from the ensemble. If there is reason to believe that some classifiers are more competent than others, giving higher weights to those classifiers may improve the classification performance. The integration of an appropriate combination rule will help us to boost the classification performance of the former algorithm.

A classifier's confidence can be used to adjust its voting weight. The concept of class specific performance CSP_{tc}, introduced earlier in [52] is defined as the training data performance of the ensemble's t^{th} classifier in correctly identifying a particular class, ω_c, for $c=1,2,...,C$. Hence, by incorporating the CSP_{tc} into the algorithm, the local expertise of the classifier for each class can be incorporated into the algorithm. A classifier's weight that takes class information into consideration may be a better quantity as a weight factor since it utilizes the coverage of the dataset. We provide a detailed description to derive the CSP later in the forthcoming section.

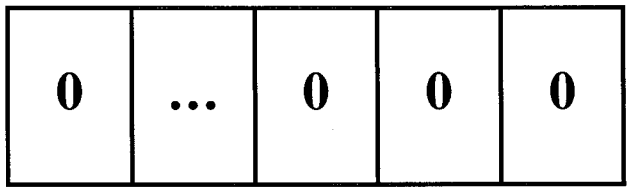

Figure 4.3: Minimum Output Variance of a Classifier

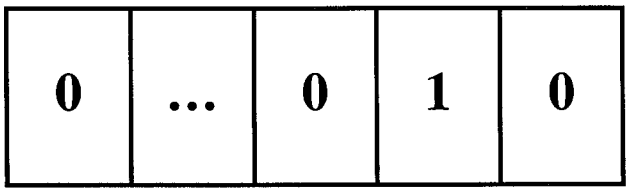

Figure 4.4: Maximum Output Variance of a Classifier

We define the output variance as the variance of the output values of a classifier for which the desired outputs are binary encoded: if the correct class is ω_c, then the c^{th} output node of the classifier is 1 and all others are zero. The minimum possible output variance is zero, which would occur if all output nodes have the same value. Such an outcome can be interpreted as the classifier not being able to make a decision (zero confidence in decision). We show an example above in Figure 4.3. The maximum possible output variance is $1/C$, which would occur if one output is 1 and all the rest are zero, which can be viewed as the classifier being absolute positive of its decision. We show an example above in Figure 4.4. For consistency across databases of different number of classes, the output variances are also normalized to [0 1] range by multiplying them with C.

In this work, the product of the class-specific performance CSP_{tc}, and output variance are used as a measure of classifier confidence. The heuristic and intuitive assumption in using the output variance as a confidence measure is as follows: a classifier with a large output variance has a higher confidence in its own decision. We increase the voting weights of those classifiers that are more confident in their decision with respect to

this measure. While high confidence decision does not necessarily assure accuracy, empirical evidence suggests that a well trained classifier usually has a high output variance when the decision is in fact correct; and a low output variance when its decision is incorrect. It was noticed that – on all databases tried so far – the classifier was almost always correct when the output variances were above a certain threshold, and almost always incorrect when below. This threshold was also empirically determined as the mean plus two times the standard deviation of the output variances of the misclassified instances (of the training data). It should be emphasized that the output variance is not a measure of accuracy, and hence is *not* used to determine whether a classifier is correct in its decision, but merely as a measure to assess competing classifiers, in determining which classifiers' decision should be weighted more favorably.

4.4 Learn^{++}.MFv2

The Learn^{++}.MFv2 algorithm uses an ensemble of classifiers, each of which is trained on a random subset of the entire feature space. Similar to Learn^{++}.MF, Learn^{++}.MFv2 assumes that the training data has no missing features, and /or there is sufficient training data with all its features intact. Like its predecessor, Learn^{++}.MFv2 focuses on the more commonly seen, and potentially more annoying case of field data containing missing features.

The pseudocode and block diagram of the Learn^{++}.MFv2 algorithm are provided in Figure 4.5 and Figure 4.6, respectively. The inputs to the algorithm are (1) the training data set $\boldsymbol{D}$; (2) the number of features, *nof*, to be used for training individual classifiers; (3) a supervised classification algorithm (**BaseClassifier**), and the number of classifiers to be created T; and (4) the sentinel value *sen* to designate a missing feature. The data set

$\boldsymbol{D}$ contains m instances, each with f number of features. The algorithm is set to run T times, generating an additional classifier C_t, h_t (hypothesis) at each iteraction t=1,...,T. The number of classifiers T should be chosen large enough to create adequate number of usable classifiers for the problem at hand.

At each iteration t=1,..., T, an iteratively updated distribution $\boldsymbol{P}_t$ for selecting which *features* of the selected instance are used for training the next classifier C_t. The discrete distribution $\boldsymbol{P}_t$ is essentially created to assign a weight to each feature. At each iteration t, a subset of features, $\boldsymbol{F}_{selection}(t)$, is drawn according to $\boldsymbol{P}_t$ such that those features with higher weights are more likely to be selected into $\boldsymbol{F}_{selection}(t)$, for training current iteration's classifier, h_t. Before the first iteration, $\boldsymbol{P}_1$ is also initialized to be uniform, unless there is reason to choose otherwise, so that each feature has equal likelihood of being selected into $\boldsymbol{F}_{selection}(1)$.

The distribution $\boldsymbol{P}_t$, is normalized in step 1 of the iterative loop, so that their sum equals to 1, and that a legitimate distribution is obtained.

$$P_t = P_t \Big/ \sum_{j=1}^{f} P_t(j) \tag{4.4}$$

Next, *nof* features are randomly drawn from $\boldsymbol{P}_t$ in step 2. The features selected constitute the set $\boldsymbol{F}_{selection}(t)$. The *nof* value should be selected carefully. A high *nof* value has been found to yield better classifiers because each classifier will be trained with more features. However, fewer qualifying classifiers will then be available to classify instances with missing features. Conversely, a low *nof* value will result in higher number qualifying classifiers for each instance; however they may be too weak to achieve a meaningful classification performance. Previous trials [67] has shown that using a dataset with a large dimensionality of redundant features, one can often obtain similar

performances over a wide range of *nof*, at a cost of varying percent of classifiable instances in the test dataset.

Only those features listed in $F_{selection}(t)$ are used for each instance for training h_t. The trained classifier h_t is then tested on the training data $\boldsymbol{D}$ in step 3. The current hypothesis h_t, is required to achieve a minimum 50% correct classification performance on $\boldsymbol{D}$ in step 4 to ensure that it has a meaningful classification capacity. If h_t does not meet this requirement, then a new $F_{selection}(t)$ is drawn and a new classifier is generated.

In step 5, $\mathbf{Mvr}_t$ is created as a vector of the output variances of the classifier for all misclassified instances

$$\mathbf{Mvr}_t = \text{outvar}(h_t(\mathbf{x}_i) \neq y_i), \forall \mathbf{x}_i, i = 1,...,m \tag{4.5}$$

where outvar(.) is simply the variance of the classifier outputs. We refer our reader to Section 3.4 for our earlier definition of output variance. The threshold for the output variance, Tvr_t, is then computed in step 6 as the mean of $\mathbf{Mvr}_t$ plus two times its standard deviation.

TRAINING

Inputs:

- Sentinel value *sen*.
- **BaseClassifier** and the number of classifiers, T.
- Training data set $D=\{(\mathbf{x}_i, y_i) \mid i=1,\ldots,m\}$ with m instances and f features.
- Number of features used to train each classifier, *nof*.

Initialize $P_1(j) = 1/f, \forall j, j = 1,\ldots,f$

Do for $t = 1, 2,\ldots, T$:

1. Set $P_t = P_t \Big/ \sum_{j=1}^{f} P_t(j)$ so that P_t is a distribution.
2. Draw *nof* of features for $F_{selection}(t)$ from $\boldsymbol{P}_t$.
3. Call **BaseClassifier** to generate a weak classifier using only those features in $F_{selection}(t)$ for each $\mathbf{x}_i$.
4. Obtain a hypothesis C_t, and its performance $Perf_t$ on D. If $Perf_t < 50$ %, discard C_t and go to step 2.
5. $\mathbf{Mvr}_t = \text{outvar}(h_t(\mathbf{x}_i) \neq y_i), \forall \mathbf{x}_i, i = 1,\ldots,m$
6. $Tvr_t = \text{Mean}(\mathbf{Mvr}_t) + 2\,\text{std}(\mathbf{Mvr}_t)$

7a. $CI_c = \sum_{i=1}^{m} [\,|\, h_t(\mathbf{x}_i) = \omega_c = y_i \,|\,], \forall c, c = 1,\cdots, C$

7b. $I_c = \sum_{i=1}^{m} [\,|\, h_t(\mathbf{x}_i) = \omega_c \,|\,], \forall c, c = 1,\cdots, C$

7c. $CSP_{tc} = CI_c / I_c$

8. Set $P_t(F_{selection}(t)) = P_t(F_{selection}(t)) \cdot (1/\lambda)$, $0 < \dfrac{1}{\lambda} < 1$

end loop

VALIDATION/TESTING

Do for each validation instance $\mathbf{x}_i$

1. $\boldsymbol{M}_{feat}(i) = \arg(\mathbf{x}_i(j) == sen), \ \forall j, j = 1,\cdots, f$,
2. $Avr_t(i) = C\,\text{outvar}(h_t(i))$
3. **If** $Avr_t(i) > Tvr_t$, $Avr_t(i) = KAvr_t(i)$ **End if**
4. $H_t(i) = \arg\max_{\omega_c} \sum_{t:h_t(\mathbf{x}_i)=\omega_c} Avr_t(i) CSP_{tc} \left[\left| M_{feat}(t) \in F_{selection}(t) \right|\right]$

end loop

Figure 4.5: Pseudocode of Algorithm Learn^{++}.MFv2

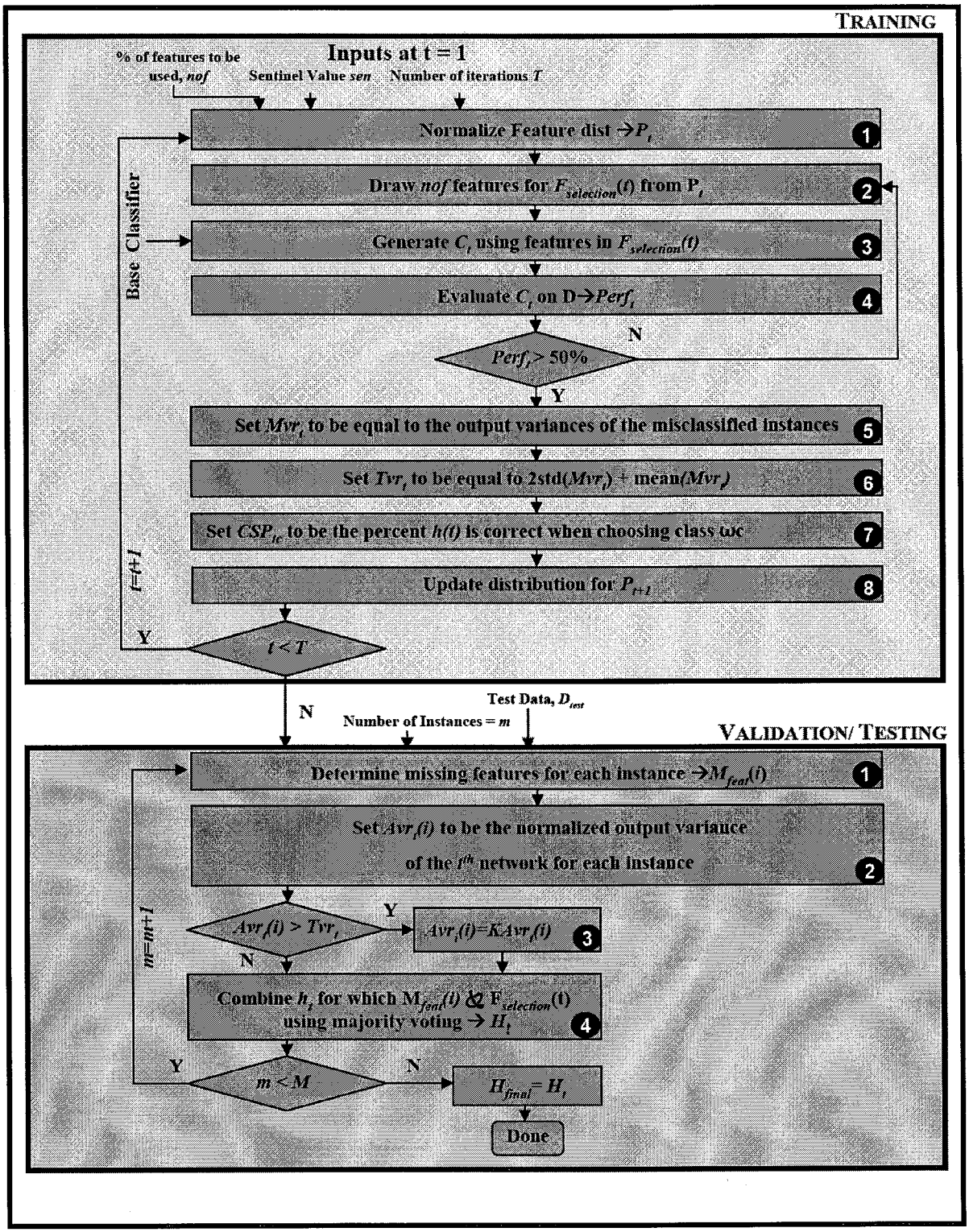

Figure 4.6: Block Diagram of Algorithm Learn^{++}.MFv2

Tvr_t is newly introduced into Learn^{++}.MFv2 and will later serve as the decision boundary for output variance in combining classifiers: if the output variance of h_t, while classifying an instance during testing, is above its Tvr_t threshold, then the voting weight of h_t will be proportionately increased, in anticipation that it is making a correct decision.

The class specific performance is further added as a confidence measure in Learn^{++}.MFv2. In order to compute class specific performance as the second weight adjustment factor in step 7, CI_c is first calculated as the number of instances that the classifier correctly identifies from class ω_c

$$CI_c = \sum_{i=1}^{m} [| h_t(\mathbf{x}_i) = \omega_c = y_i |], \forall c, c = 1, \cdots, C \tag{4.6}$$

followed by I_c, the total number of instances that the classifier identifies as class ω_c, correct or otherwise:

$$I_c = \sum_{i=1}^{m} [| h_t(\mathbf{x}_i) = \omega_c |], \forall c, c = 1, \cdots, C \tag{4.7}$$

where $[| \bullet |]$ evaluates to 1 if the predicate holds true. The class specific performance CSP_{tc} of h_t on class ω_c is then the proportion that the t^{th} classifier is correct when assigning an instance to class ω_c:

$$CSP_{tc} = CI_c / I_c, 0 \le CSP_{tc} \le 1 \tag{4.8}$$

Next, the distribution $\boldsymbol{P}_t$ is updated in step 8 according to

$$P_t(F_{selection}(t)) = P_t(F_{selection}(t)) \cdot \frac{1}{\lambda} \tag{4.9}$$

such that the weights of those features that appear in $\boldsymbol{F}_{selection}(t)$ are reduced by a factor of λ, *where* $0 < \frac{1}{\lambda} < 1$. The value of $\frac{1}{\lambda}$ should be selected carefully so as to ensure that the

weights of the features are not drastically reduced. Those features that were not in the current selection effectively have their weights increased when $\boldsymbol{P}_t$ is normalized again in step 1 of iteration t+1. While the algorithm does use the notion of random subspace selection of its features, the selection of features that Learn^{++}.MFv2 and its predecessor, Learn^{++}.MF take are not entirely random as explained earlier in this chapter. It should be noted again that T should be chosen large enough to create an adequate number of usable classifiers for the problem at hand.

After T classifiers are generated, the algorithm proceeds to the validation phase. During the validation phase, the algorithm searches for sentinels, the placeholders for missing data, in step 1 of the validation phase. To ensure that actual values are not mistaken for the sentinel, *sen* should be chosen as a value not expected to occur in the data. All features j, $j=1,\ldots,f$ with a sentinel value in the given instance are then flagged and placed into the set of missing features $\boldsymbol{M}_{feat}(i)$ for that instance $\mathbf{x}_i$. For final classification, only *qualifying classifiers* whose feature selection list $\boldsymbol{F}_{selection}(t)$ did not include those in $\boldsymbol{M}_{feat}(i)$ (that is, classifiers that did not use any of the features in $\boldsymbol{M}_{feat}(i)$) are combined to determine the classification of test instance $\mathbf{x}_i$.

As t^{th} classifier evaluates $\mathbf{x}_i$, its output variance is computed and normalized to [0 1] range as $Avr_t(i)$ in step 2 of the validation phase,

$$Avr_t(i) = C\,\mathrm{outvar}\left(h_t(i)\right) \tag{4.10}$$

If $Avr_t(i)$ is greater than the threshold variance Tvr_t calculated earlier (which indicates that h_t is probably correct in classifying $\mathbf{x}_i$), then it is multiplied by a constant K, where $1 < K \leq 2$ is to be used in the increasing voting weight of h_t. Hence, a classifier is that is more confident in its own decision will have its weights increased. However, it

should be noted that if it does not meet the above condition, the classifier is not penalized and its own weight is not lowered. For any given instance $\mathbf{x}_i$, each usable classifier's vote is obtained as the product of Avr_t and CSP_{tc}, and these classifiers are then combined through weighted majority voting to obtain the composite hypothesis $H_t(i)$ for the i^{th} instance $\mathbf{x}_i$:

$$H_t(i) = \arg\max_{\omega_c} \sum_{t:h_t(\mathbf{x}_i)=\omega_c} Avr_t(i) CSP_{tc} \left[\left| M_{feat}(i) \notin F_{selection}(t) \right| \right] \tag{4.11}$$

The current composite hypothesis becomes the final hypothesis for the instance $\mathbf{x}_i$.

In summary, Learn^{++}.MFv2 has introduced some modifications to the original Learn^{++}.MF algorithm. This includes the CSP_{tc} (class specific performance of the t^{th} classifier, and Tvr_t (serves as the decision boundary for output variance in combining classifiers). These modifications attempt to boost the performance of the algorithm above that of Learn^{++}.MF by taking advantage of previously well known techniques such as the weighted majority voting to utilize the coverage of the data to attempt to be a more robust algorithm.

CHAPTER 5 – IMPLEMENTATION AND RESULTS

This chapter is split into four sections. Section 5.1 describes the testing procedure used by both Learn^{++}.MF and Learn^{++}.MFv2 for the datasets used in validating the results. Section 5.2 describes each of the datasets and the simulation results of the algorithms Learn^{++}.MF and Learn^{++}.MFv2 on those datasets. Section 5.3 summarizes the observation and trends based on the simulation results. Section 5.4 provides an overall evaluation of both the algorithms.

5.1 Testing Procedure

In all cases, multilayer perceptrons were used as the base classifier, though any supervised classifier can be used. The training parameters of the base classifiers (e.g. error goal, number of hidden layer nodes), were not fine tuned, but rather selected as reasonable values for the given database, giving the ensemble additional diversity through relatively instable base classifiers.

The *global number of features* in the dataset is defined as the product of the number of features per instance f, and the number of instances, m; and a single *test trial* as evaluating the algorithm with 0.0% ~ 30.0% (in steps of 2.5%), of the global number of features missing from the test dataset. The algorithms were evaluated with different number of features, *nof*, used in training individual classifiers. All performance figures are averages of 10 test trials, reported with 95% confidence intervals. Missing features were simulated by randomly replacing actual feature values with sentinels.

An example of the global number of features is shown in Figure 5.1 which has 20 instances and 8 features. A white or clear block is shown as having a clean or present

feature. A black block, on the other hand, represents a feature missing or corrupt. Figure 5.2 and Figure 5.3 show examples of 10% and 20% of the feature space missing or corrupt. This corresponds to 16 and 32 black blocks on the figures respectively.

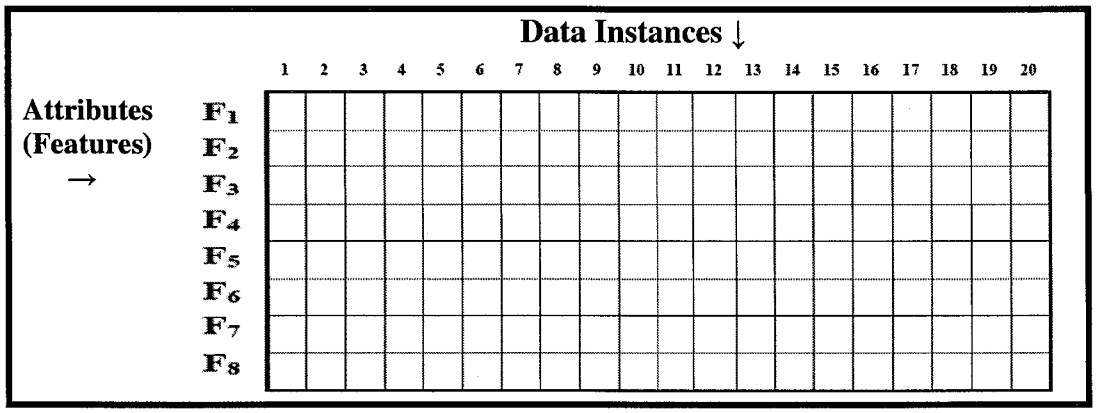

Figure 5.1: Example of a feature space with 20 instances having 8 features

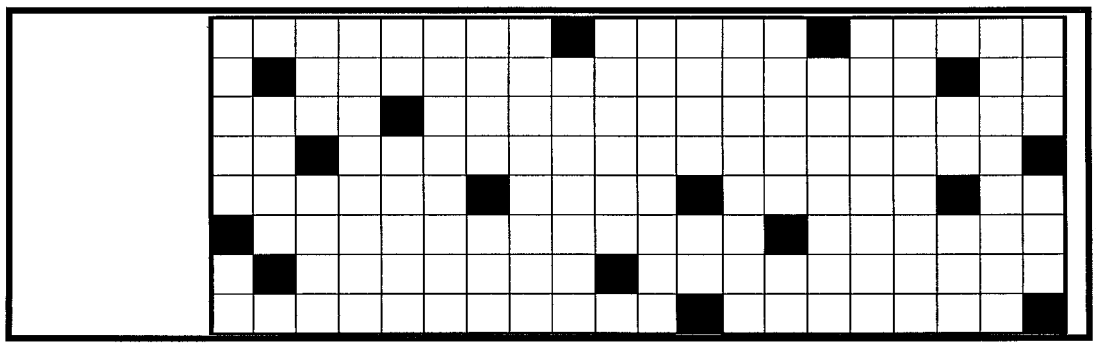

Figure 5.2: Example of 10% feature space artificially missing or corrupt

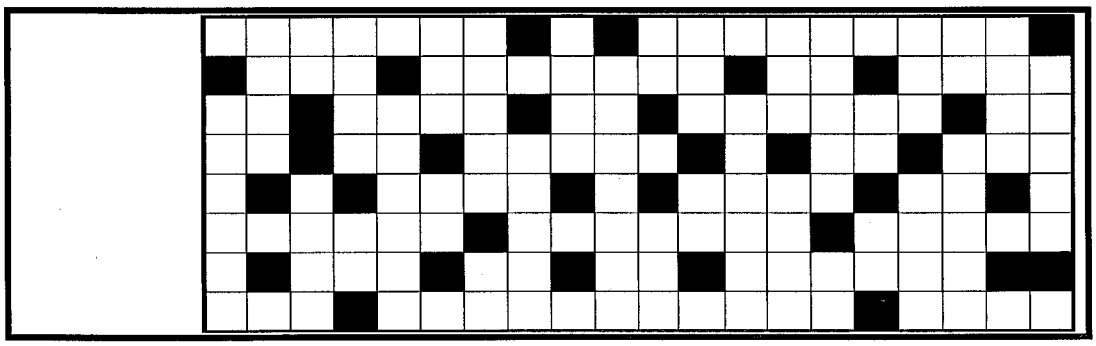

Figure 5.3: Example of 20% feature space artificially missing or corrupt

5.2 Discussion of Simulation Results

Learn^{++}.MF and Learn^{++}.MFv2 were evaluated on two real world applications of hazardous gas identification (VOC-I and VOC-II), and eight benchmark databases (Ionosphere (ION), Wine, Dermatology (DERMA), Wisconsin Breast Cancer (WBC), Water, Pen Digits (PEN), Optical Character Recognition (OCR) and E-coli) from the UCI repository [71]. Table 5.1 provides a summary of the data distributions for all databases used in our experiments. For each database, the number of features f is given in parentheses, followed by class-specific size of training and test datasets. Table 5.2 provides the various values of *nof*, and the total number of classifiers generated for each database. For example, the Ionosphere database (ION) consists of a total of f=34 features, was used with four different *nofs* (8, 10, 12, and 14 out of 34 features). T was set to 1000. The effect of these parameters, and hence suggestions for their selection, are discussed in detail in the following paragraphs.

We see many similar trends between the algorithms' behaviors. Hence, instead of repeating many of the trends for each subsection, the most important or noteworthy issues are discussed in detail in the following subsections.

Table 5.1: Data Distribution for all datasets evaluated

Dataset ↓	*Class→*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*Total*
ION (34)	**Train**	30	30	-	-	-	-	-	-	-	-	-	-	60
	Test	100	110	-	-	-	-	-	-	-	-	-	-	210
WBC (30)	**Train**	100	100	-	-	-	-	-	-	-	-	-	-	200
	Test	100	100	-	-	-	-	-	-	-	-	-	-	200
WINE (13)	**Train**	49	61	38	-	-	-	-	-	-	-	-	-	148
	Test	10	10	10	-	-	-	-	-	-	-	-	-	30
WATER (38)	**Train**	96	44	26	22	-	-	-	-	-	-	-	-	188
	Test	95	44	25	22	-	-	-	-	-	-	-	-	186
VOC-I (6)	**Train**	30	30	50	30	40	-	-	-	-	-	-	-	180
	Test	34	34	62	34	40	-	-	-	-	-	-	-	204
ECOLI (5)	**Train**	60	40	30	20	10	-	-	-	-	-	-	-	160
	Test	40	30	20	15	10	-	-	-	-	-	-	-	115
DERMA (34)	**Train**	70	40	50	30	30	15	-	-	-	-	-	-	235
	Test	30	20	21	18	18	5	-	-	-	-	-	-	112
PEN (16)	**Train**	50	50	50	50	50	50	50	50	50	50	-	-	500
	Test	50	50	50	50	50	50	50	50	50	50	-	-	500
OCR (62)	**Train**	100	100	100	100	100	100	100	100	100	100	-	-	1000
	Test	50	50	50	50	50	50	50	50	50	50	-	-	500
VOC-II (12)	**Train**	5	5	5	5	5	5	5	5	5	5	5	5	60
	Test	2	2	2	2	2	2	2	2	2	2	2	2	24

Table 5.2: Number of features (nof) and number of classifiers (T) used for each dataset

Dataset ↓	nof_1	nof_2	nof_3	nof_4	nof_5	T
ION (34)	8	10	12	14	-	1000
WBC (30)	10	12	14	16	-	1000
WINE (13)	3	4	5	6	7	200
WATER (38)	12	14	16	18	-	1000
VOC-I (6)	2	3	-	-	-	100
ECOLI (5)	2	3	-	-	-	1000
DERMA (34)	8	10	12	14	-	1000
PEN (16)	6	7	8	9	-	250
OCR (62)	16	20	24	-	-	1000
VOC-II (12)	3	4	5	6	-	200

5.2.1 *Volatile Organic Compound I Dataset*

This database consisted of responses of six quartz crystal microbalances (QCM) to five volatile organic compounds, including ethanol (ET), xylene (XL), octane (OC), toluene (TL), and trichloroethylene (TCE). Of the 384 six-dimensional signals, 180 were used for training and 204 for testing. Previous experiments have shown that using optimized classifiers trained by using all six features tested on the test dataset with no missing features performed at 86.2%, setting the benchmark target for this database. Two values of *nof* were considered: 2 and 3, out of six, corresponding to 33.3% and 50% of available features, respectively. T was set as 100 classifiers.

Table 5.3 summarizes the test performances for both algorithms Learn^{++}.MF and Learn^{++}.MFv2. The columns in Table 5.3 includes both the *nofs,* and also include the percent of the total number of the instances that could be processed (correctly or otherwise) with the existing ensemble for both algorithms. For brevity and space considerations, we denote v1 to mean Learn^{++}.MF and v2 to mean Learn^{++}.MFv2.

We begin by discussing the results of Learn^{++}.MF first. Table 5.3 indicates that the algorithm performed quite well, even for a substantial amount of missing features. Note that the first row with 0.0% missing features indicates the algorithm performance when individual classifiers were trained on *nof* features, but with no features missing on the test data. The proximity of this number to the target 86.2% (obtained when all 6 features were used) especially for the case when *nof*=3/6, indicates that this dataset does include redundant features.

Also, since the features for each classifier are selected at random, it is possible that the particular features available for any given instance (after removing the missing

features) do not match any of the feature combinations selected by the classifiers. Such instances cannot be processed, as there would be no classifier trained with the unique combination of the available features. The performances given in the tables are calculated on those instances that can be processed, and the percent of such instances are also shown in tables.

To describe the effect of the algorithm's free parameters, we also provide a set of four figures for each database analyzed. Figure 5.4a summarizes the ensemble performance of Learn^{++}.MF on this dataset. Figure 5.4b shows the average performance of a single usable classifier trained with *nofs* = 2/6 and 3/6. We note that the ensemble significantly outperforms the single classifier for either selection of *nof*, and the ensemble is also able to process a larger portion of the data. A single usable classifier trained on 3 out of 6 features is able to classify 72% of the instances with a 73% performance, whereas the ensemble of classifiers is able to classify all instances even when 10% of the features are corrupt or missing with a performance of 85%. We also observe that a classifier trained on a larger *nof* normally achieves a higher generalization performance, but is only able to classify lesser instances as the percent of missing feature increases. This phenomenon is true for both the ensemble and a single usable classifier and can be attributed to the fact that when larger number of features is used for training, fewer classifiers are available to accommodate instances that have many missing features. In fact, the probability that there may be no classifier available for a specific combination of missing features increases, if larger number of features were used for training, since a larger number of features are required for processing these instances. Of course, in the limiting case, if all features were used for training, there would be no classifier available

even for a single missing feature, which brings us back to the motivation behind using an ensemble trained with random feature subsets. Figure 5.4c, shows percent of instances that can be processed decreases with increasing ratio of missing features for both the ensemble and a single usable classifier. The ensemble for both the *nofs* are shown on the top portion of this figure, whereas the single classifier for both the *nofs* is shown in the bottom half. This is closely related to the percent of usable classifiers for any given combination with or without missing features as seen in Figure 5.4d. Figure 5.4d depicts the decline in percent of usable classifiers for any given classifiable instance for both the *nof* values considered, as a function of the ratio of missing features.

Table 5.3: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the VOC-I Dataset

	(nof = 2/5)				*(nof = 3/5)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	77.45 ± 0.00	100	81.86 ± 0.00	100	85.29 ± 0.00	100	91.67 ± 0.00	100
2.50%	77.70 ± 0.47	100	81.91 ± 0.31	100	84.80 ± 0.23	100	91.47 ± 0.34	100
5.00%	77.89 ± 0.58	100	81.67 ± 0.44	100	84.79 ± 0.76	100	91.13 ± 0.42	100
7.50%	77.39 ± 0.83	100	82.01 ± 0.57	100	84.75 ± 0.87	100	90.88 ± 0.55	100
10.00%	77.18 ± 0.60	100	82.30 ± 0.91	100	84.28 ± 0.69	100	90.44 ± 0.76	100
12.50%	77.43 ± 0.69	100	82.16 ± 0.81	100	83.94 ± 1.08	100	90.29 ± 1.52	100
15.00%	76.88 ± 0.83	100	81.67 ± 0.74	100	83.64 ± 0.59	99	90.10 ± 0.79	100
17.50%	77.96 ± 0.67	100	82.50 ± 0.50	100	82.80 ± 1.67	99	89.80 ± 0.93	100
20.00%	77.08 ± 0.90	100	82.21 ± 0.79	100	81.56 ± 0.66	98	89.85 ± 1.12	100
22.50%	77.70 ± 0.80	99	82.26 ± 0.81	100	82.98 ± 1.48	96	89.95 ± 0.88	100
25.00%	76.80 ± 1.03	99	82.06 ± 1.07	100	82.02 ± 1.37	96	89.12 ± 1.78	100
27.50%	77.10 ± 1.26	98	81.27 ± 0.92	100	81.04 ± 1.20	94	88.82 ± 0.82	100
30.00%	75.36 ± 1.60	98	81.23 ± 1.25	100	80.54 ± 1.72	92	88.48 ± 1.41	100

Table 5.3 also summarizes results obtained by using Learn^{++}.MFv2 and Figure 5.5 illustrates the trends obtained by Learn^{++}.MFv2 in a similar manner described above. Figure 5.5a indicates one clear advantage in using Learn^{++}.MFv2: the ensemble performances of Learn^{++}.MFv2 with *nofs* = 2/6 and 3/6 are 81.86% and 91.67% with no features missing respectively, whereas the ensemble performances of Learn^{++}.MF with

nofs = 2/6 and 3/6 are 77.45% and 85.29% respectively when no features missing. With 30% of the feature space missing or corrupt, Learn^{++}.MFv2 was still able to achieve 81% and 88% for *nofs* =2/6 and 3/6 as opposed to Learn^{++}.MF achieving 75% and 80% respectively. Using Learn^{++}.MFv2 on this dataset has seen a 4-6% increase in ensemble performance. We refer the reader to compare these figures to the performance of the ensembles created by Learn^{++}.MF.

Earlier, we reported that the target performance for this dataset using an optimized classifier trained on all features yielded 86.2% performance. However, the Learn^{++}.MFv2 ensemble performance of 91.67% was able to surpass the target performance using *nof* = 3/6. A possible explanation for this can be attributed to potentially irrelevant features in the original database, where the removal of such features combined with weighted majority voting further improves the performance. Figure 5.5b shows the average performance of a single usable classifier from Learn^{++}.MFv2 ensemble trained with *nofs* = 2/6 and 3/6. There is an approximate 7-10% increase in the performance of a single usable classifier under Learn^{++}.MFv2. This explains the reason for the increase in performance of the ensemble. Figure 5.5c shows that the percent of instances that can be processed decreases with increasing ratio of missing features for both the ensemble and a single usable classifier, an expected outcome. Figure 5.5d reports the average decline in percent of usable classifiers for any given classifiable instance, for both *nof* values considered as the ratio of missing features increases. These trends were also seen by Learn^{++}.MF earlier on the same dataset.

As confirmed by the results on the other datasets (presented below), a larger *nof* provides a better initial performance than a smaller *nof*, but the ensemble is more

resistant to unusable feature combinations with a lower *nof*. This makes perfect sense: larger *nof* gives better performance – on those instances that can be processed – since more features are available to learn the underlying data distributions. However, lower *nof* is more resistant to unusable feature combinations, since fewer features are necessary to identify any given instance. Hence, the choice of *nof* presents an interesting trade-off, which is further analyzed in the following results.

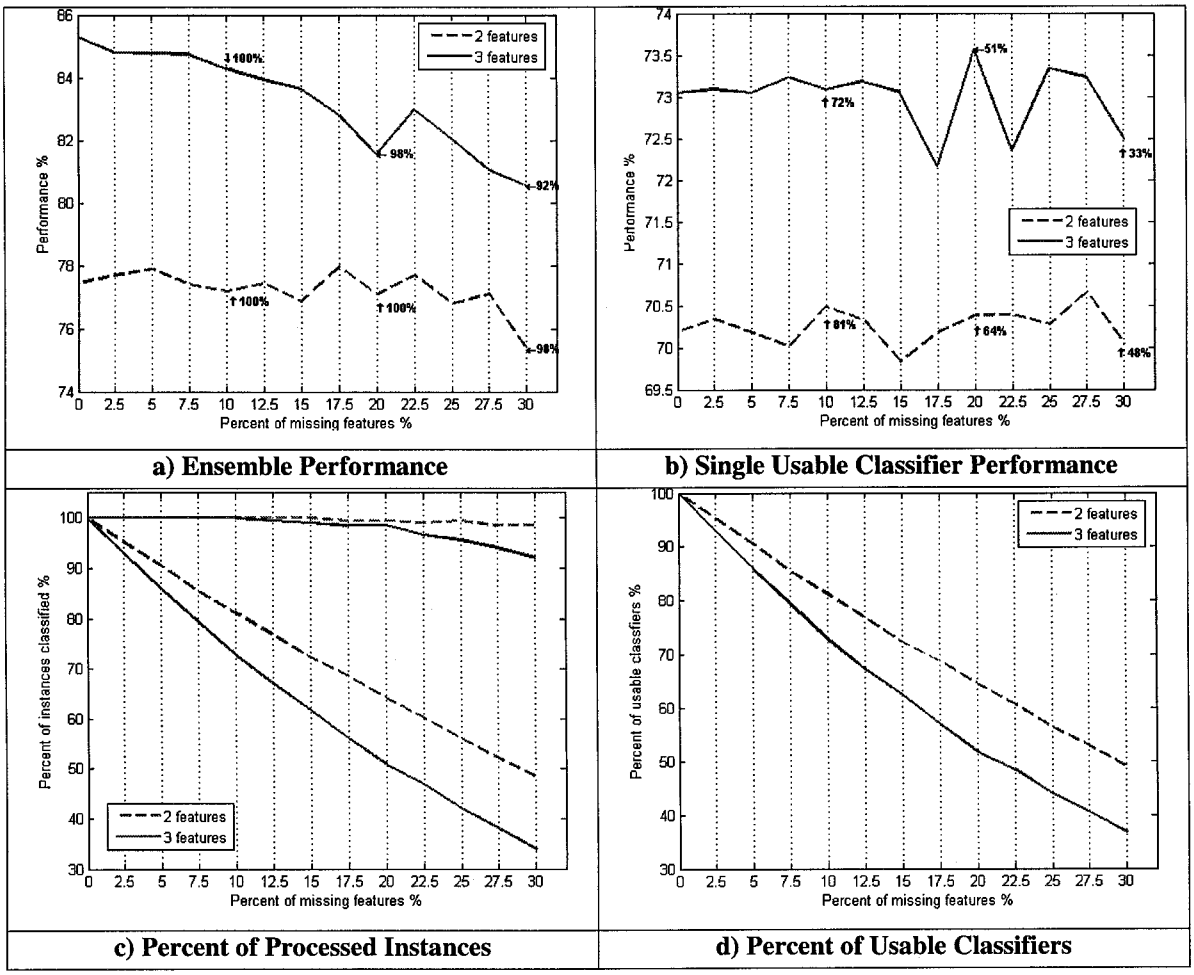

Figure 5.4: Learn^{++}.MF Performance Results on VOC-I Dataset

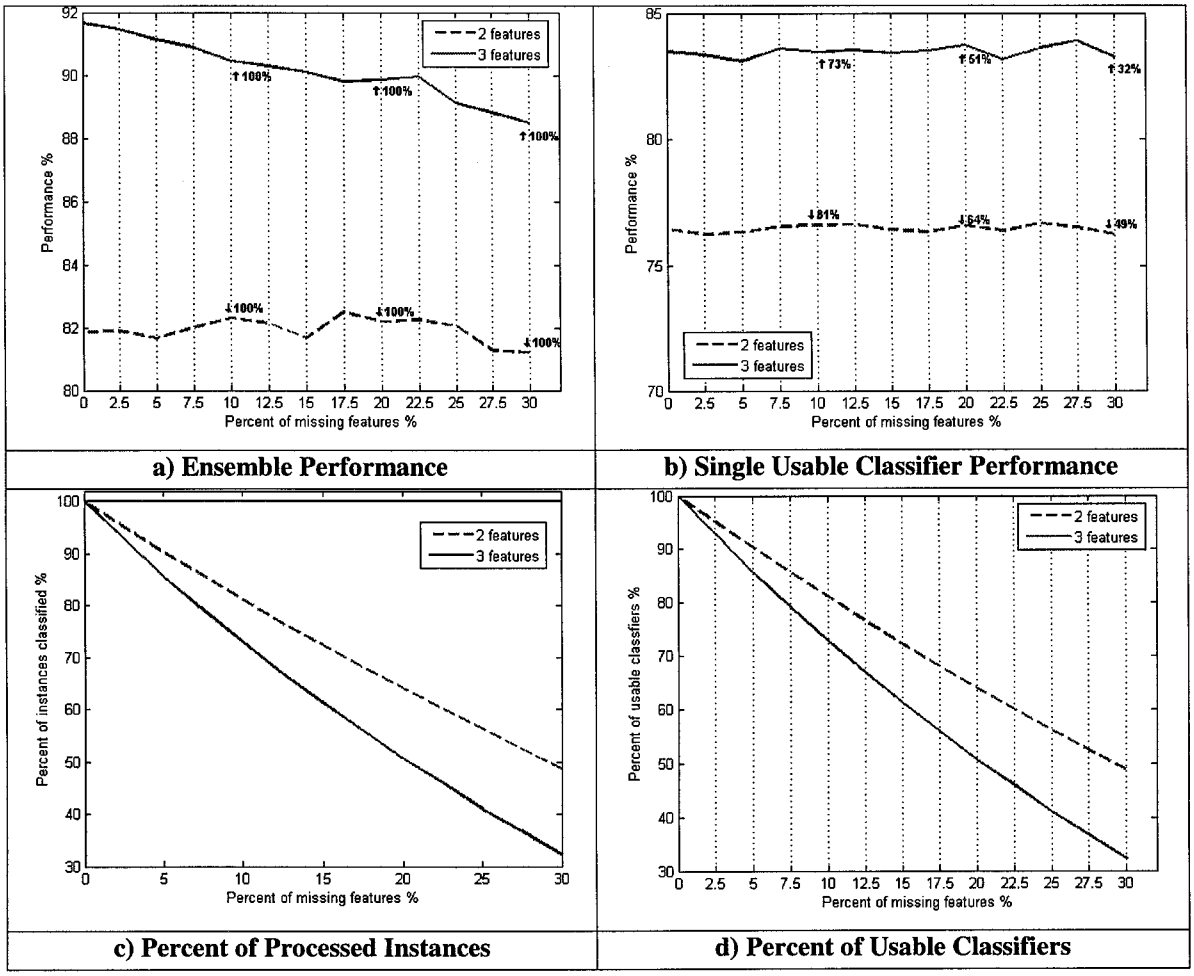

Figure 5.5: Learn^{++}.MFv2 Performance Results on VOC-I Dataset

5.2.2 *Volatile Organic Compound II Dataset*

This smaller, but higher dimensional version of the VOC database consisted of responses of twelve QCMs to twelve VOCs, including acetone (AC), acetonitrile (ACN), toluene (TL), xylene (XL), hexane (HX), octane (OC), methanol (ME), ethanol (ET), methyl ethyl ketone (MEK), tricholoroethylene (TCE), tricholoroethane (TCA), and dichloromethane (DCA). Of the 84 12-dimensional signals, 60 were used for training (five per class) and 24 for testing (two per class). Previous experiments have shown that optimized classifiers trained using all features with no missing features performed at 100%, setting the benchmark target for our experiments. Four values of *nof* were considered: 3, 4, 5 and 6 features, out of 12, corresponding to 25%, 33.3%, 41.7% and 50.0% of the features, respectively. T was set to 200 classifiers.

Table 5.4 summarizes the test performances for Learn^{++}.MF and Learn^{++}.MFv2. Similar to the previous dataset, Table 5.4 provides results for both *nofs,* and also includes the percent of of the instances that could be processed (correctly or otherwise) with the existing ensemble.

Figure 5.6a summarizes the performance of Learn^{++}.MF on the VOC-II dataset. Figure 5.6a clearly shows that both the ensemble performances, and the percent of instances that can be processed by the ensemble decline as the percentage of missing features increases; however, they both remain reasonably high up to a 20% missing features rate. In fact, with *nof* = 4, 5 and 6, the ensembles performances approach or reach the target benchmark.

Similar to the previous case, we observe the following: the initial performance of the algorithm using *nof*=3 out of 12 started at 95.8% when all features were intact. Its

initial performance was lower than the initial performance level of 100% for classifiers that had been trained with a larger *nof*. However, the algorithm using *nof* = 3, experienced a slower rate of decline in performance and, for the most part, managed to maintain a steady performance as the percent of missing features increased. With 30% of the features missing, the ensemble could still process 98% of the dataset and achieve an 88% classification performance. In comparison, the initial performance of the algorithm using 6 out of 12 features is 97.7% with no features missing. However, it can only process 88% of the dataset, and even then, can only achieve an overall performance of 83%.

We observe again that the initial performance with a larger *nof* is higher; however, the performance drop is steeper with increased missing features, compared to ensembles trained on fewer features. The decline in the percent of instances that can be processed is also steeper when larger *nof* is used to train the classifiers.

The initial performance of Learn^{++}.MF using *nof* = 3 out of 12 started at 95.8% when all features were intact. This performance was lower than the initial performance level of 100% for classifiers that had been trained with a larger *nof*. However, the performance of Learn^{++}.MFv2 using *nof*=3 out of 12 started with 100%. It was able to maintain this performance even when ~7.5% of the feature space was randomly missing. For this dataset, Learn^{++}.MFv2 was able to match the performance of its predecessor on all *nofs* even when trained on a smaller *nof*. Our earlier trials from Learn^{++}.MF has shown that using a lower *nof* normally results in a lower ensemble performance. However, such ensembles were also more tolerant to the percent of instances that they could process. In this case, Learn^{++}.MFv2 is able to take advantage of the weighted majority schemes and

obtain a higher generalization performance that was impossible for its predecessor even with using a smaller *nof* as seen in this case when using *nof*=3/12.

Table 5.4: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the VOC-II Dataset

	(nof = 3/12)				*(nof = 4/12)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	95.83 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100
2.50%	96.67 ± 1.26	100	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100
5.00%	96.67 ± 1.26	100	100.00 ± 0.00	100	100.00 ± 0.00	100	99.58 ± 0.94	100
7.50%	96.67 ± 1.26	100	98.75 ± 1.44	100	99.58 ± 0.94	100	100.00 ± 0.00	100
10.00%	95.42 ± 2.20	100	99.58 ± 0.94	100	98.33 ± 1.54	100	99.58 ± 0.94	100
12.50%	95.83 ± 1.99	100	98.33 ± 1.54	100	100.00 ± 0.00	100	100.00 ± 0.00	100
15.00%	96.67 ± 1.26	100	98.75 ± 1.44	100	100.00 ± 0.00	100	98.33 ± 1.54	100
17.50%	97.08 ± 2.01	100	98.75 ± 1.44	100	98.33 ± 1.54	100	97.50 ± 2.51	100
20.00%	97.92 ± 1.57	100	97.92 ± 1.57	100	98.33 ± 1.54	100	99.58 ± 0.94	100
22.50%	98.75 ± 1.44	100	99.58 ± 0.94	100	98.33 ± 1.54	100	97.92 ± 2.11	100
25.00%	96.25 ± 2.96	100	95.83 ± 2.43	100	97.50 ± 1.54	100	97.50 ± 1.54	100
27.50%	95.83 ± 1.99	100	96.67 ± 2.35	100	97.50 ± 1.54	100	99.58 ± 0.94	100
30.00%	95.00 ± 2.74	100	95.83 ± 2.43	100	99.17 ± 1.26	100	97.92 ± 1.57	99
	(nof = 5/12)				*(nof = 6/12)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100
2.50%	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100
5.00%	99.58 ± 0.94	100	100.00 ± 0.00	100	100.00 ± 0.00	100	99.58 ± 0.94	100
7.50%	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100
10.00%	100.00 ± 0.00	100	99.58 ± 0.94	100	100.00 ± 0.00	100	99.58 ± 0.94	100
12.50%	100.00 ± 0.00	100	99.58 ± 0.94	100	99.58 ± 0.94	100	99.58 ± 0.94	99
15.00%	100.00 ± 0.00	100	99.58 ± 0.94	100	99.58 ± 0.94	100	99.57 ± 0.98	99
17.50%	99.58 ± 0.94	100	100.00 ± 0.00	100	99.58 ± 0.94	98	99.15 ± 1.28	99
20.00%	99.58 ± 0.94	99	100.00 ± 0.00	100	99.11 ± 1.34	97	98.73 ± 2.03	98
22.50%	98.73 ± 1.46	100	98.70 ± 1.50	98	98.28 ± 1.59	97	98.69 ± 1.50	95
25.00%	98.30 ± 1.57	98	99.15 ± 1.28	98	98.64 ± 1.57	93	98.20 ± 2.22	94
27.50%	97.90 ± 2.12	98	97.86 ± 1.61	98	97.39 ± 2.17	93	99.11 ± 1.34	90
30.00%	96.97 ± 2.09	95	97.42 ± 2.12	97	97.47 ± 3.06	86	97.57 ± 2.49	86

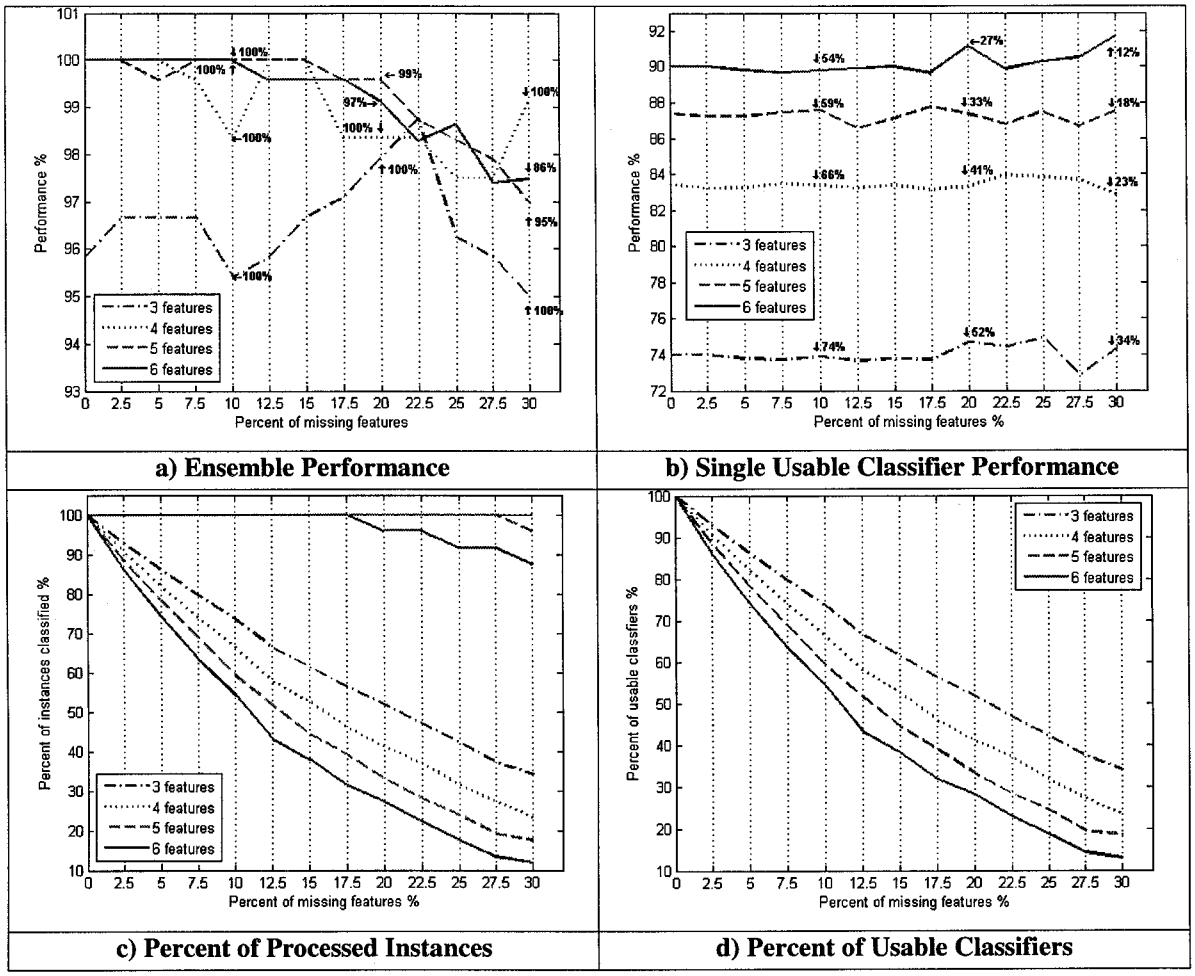

Figure 5.6: Learn^{++}.MF Performance Results on VOC-II Dataset

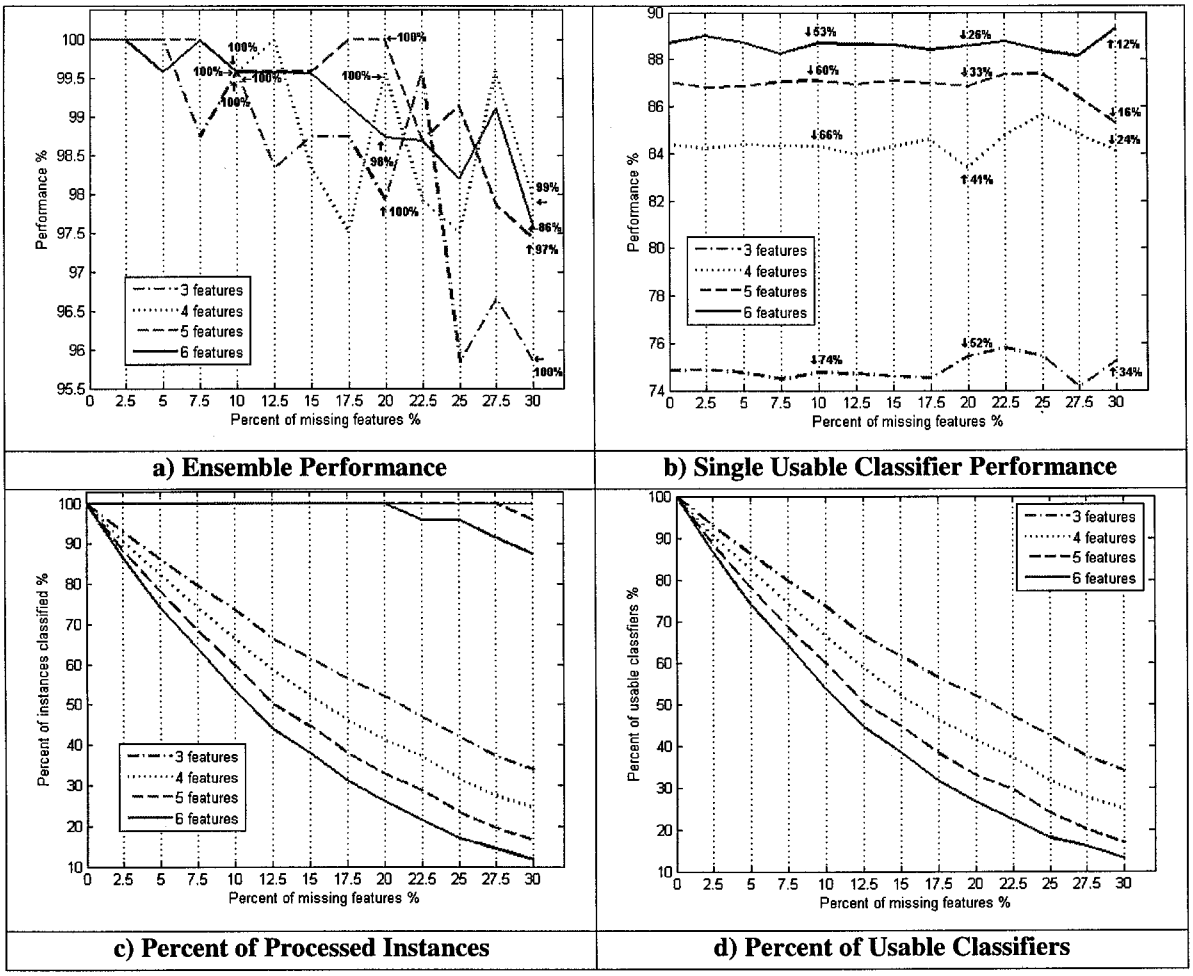

Figure 5.7: Learn^{++}.MFv2 Performance Results on VOC-II Dataset

5.2.3 Ionosphere (ION) Dataset

This benchmark database consisted of 60 training instances and 210 test instances of radar returns through the ionosphere. The targets of the radar system were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; whose signals pass through the ionosphere. Previous experiments have shown that optimized classifiers trained using all features tested on the test dataset with no missing features performed at 95%, setting the benchmark target for this dataset. The algorithms were evaluated on 4 values of *nof* = 8, 10, 12 and 14 out of the 34 available attributes. Table 5.5, Figure 5.8 and Figure 5.9 show the performance results and general trends of both algorithms.

The performance results of Learn^{++}.MF clearly show that there are indeed redundant features in this dataset. The performance for the ensembles trained under the algorithm Learn^{++}.MF were closely weaved between 91-92% for all *nofs*. However, although this did not seem to have a big influence in performance results, the choice of *nof* is critical in the percent of instances that the ensemble can classify and process. It is noted that there were also some mild increases in performances as the ratio of missing features increased. Figure 5.8c shows the percent of instances that can be classified, on average, by a single classifier and an ensemble for all the *nofs* under consideration for this dataset. It is evident from Figure 5.8c that the percent of instances that can be classified declines for any ensemble or list of single usable classifiers. Ensembles or single classifiers trained on a larger *nof* are generally less tolerant to combination of missing features.

Learn^{++}.MFv2 achieved similar ensemble performances for all values of *nofs*. The ensemble performance rates for Learn^{++}.MFv2 were between 94-95% when there were few or no features missing. It was observed that there was generally a 3% increase in the ensemble performance for Learn^{++}.MFv2 in comparison to the Learn^{++}.MF for most of the *nofs*. There was no notable difference in the percent of instances that could be classified by either algorithm for the respective *nofs*.

Table 5.5: Learn^{++}.MF and Learn^{++}.MFv2 Performances of the ION Dataset

	(nof = 8/34)				*(nof = 10/34)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	91.30 ± 0.00	100	94.29 ± 0.00	100	91.30 ± 0.00	100	94.29 ± 0.00	100
2.50%	92.32 ± 0.85	100	93.33 ± 1.11	100	91.01 ± 0.44	100	94.20 ± 0.49	100
5.00%	92.17 ± 1.22	100	92.32 ± 1.20	100	91.45 ± 1.03	100	92.90 ± 1.24	100
7.50%	92.32 ± 1.20	100	93.04 ± 0.82	100	90.14 ± 0.82	100	92.03 ± 1.22	100
10.00%	92.03 ± 1.31	100	91.16 ± 1.03	100	90.87 ± 1.10	100	92.17 ± 1.11	100
12.50%	93.77 ± 0.98	100	92.17 ± 1.31	100	90.68 ± 1.61	100	92.03 ± 0.88	100
15.00%	92.46 ± 1.27	100	92.03 ± 0.88	100	90.96 ± 1.38	100	92.32 ± 1.10	100
17.50%	92.46 ± 1.07	100	91.30 ± 1.46	100	90.87 ± 1.11	100	92.17 ± 1.71	100
20.00%	92.61 ± 1.24	100	92.03 ± 0.88	100	90.33 ± 2.03	100	91.00 ± 1.44	100
22.50%	92.03 ± 1.01	100	93.33 ± 1.22	100	88.87 ± 1.70	99	91.69 ± 1.48	99
25.00%	92.03 ± 1.31	100	90.71 ± 1.30	100	90.30 ± 2.41	98	91.97 ± 1.47	99
27.50%	91.58 ± 1.86	100	92.32 ± 1.47	100	88.75 ± 1.81	97	90.25 ± 1.76	97
30.00%	91.97 ± 1.32	99	91.42 ± 1.25	100	88.91 ± 2.54	94	89.53 ± 1.65	94
	(nof = 12/34)				*(nof = 14/34)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	91.30 ± 0.00	100	94.76 ± 0.00	100	92.75 ± 0.00	100	95.71 ± 0.00	100
2.50%	91.45 ± 0.33	100	94.86 ± 0.69	100	92.32 ± 1.10	100	95.57 ± 0.82	100
5.00%	91.30 ± 0.69	100	95.43 ± 0.72	100	92.61 ± 1.14	100	96.43 ± 0.53	100
7.50%	91.30 ± 0.85	100	95.58 ± 0.55	100	91.74 ± 0.70	100	96.43 ± 1.00	100
10.00%	91.01 ± 0.82	100	95.72 ± 1.31	100	91.45 ± 1.24	100	95.71 ± 1.42	100
12.50%	90.72 ± 0.72	100	95.57 ± 1.47	100	92.03 ± 1.40	99	96.81 ± 1.11	99
15.00%	91.28 ± 0.85	100	94.84 ± 0.98	100	91.59 ± 1.07	99	95.98 ± 1.63	99
17.50%	90.94 ± 1.06	99	96.94 ± 0.84	99	91.16 ± 1.42	97	95.70 ± 1.27	97
20.00%	90.70 ± 2.27	98	95.76 ± 1.33	99	92.58 ± 1.85	93	96.14 ± 1.75	93
22.50%	89.55 ± 1.93	96	96.05 ± 1.25	95	91.83 ± 1.21	89	94.54 ± 2.82	89
25.00%	90.20 ± 2.91	92	93.62 ± 2.27	93	89.82 ± 1.96	80	94.04 ± 2.33	80
27.50%	88.70 ± 2.79	89	94.16 ± 1.78	88	92.25 ± 1.53	71	94.36 ± 2.61	71
30.00%	90.67 ± 2.54	80	94.26 ± 2.66	79	90.25 ± 1.48	60	93.32 ± 3.13	60

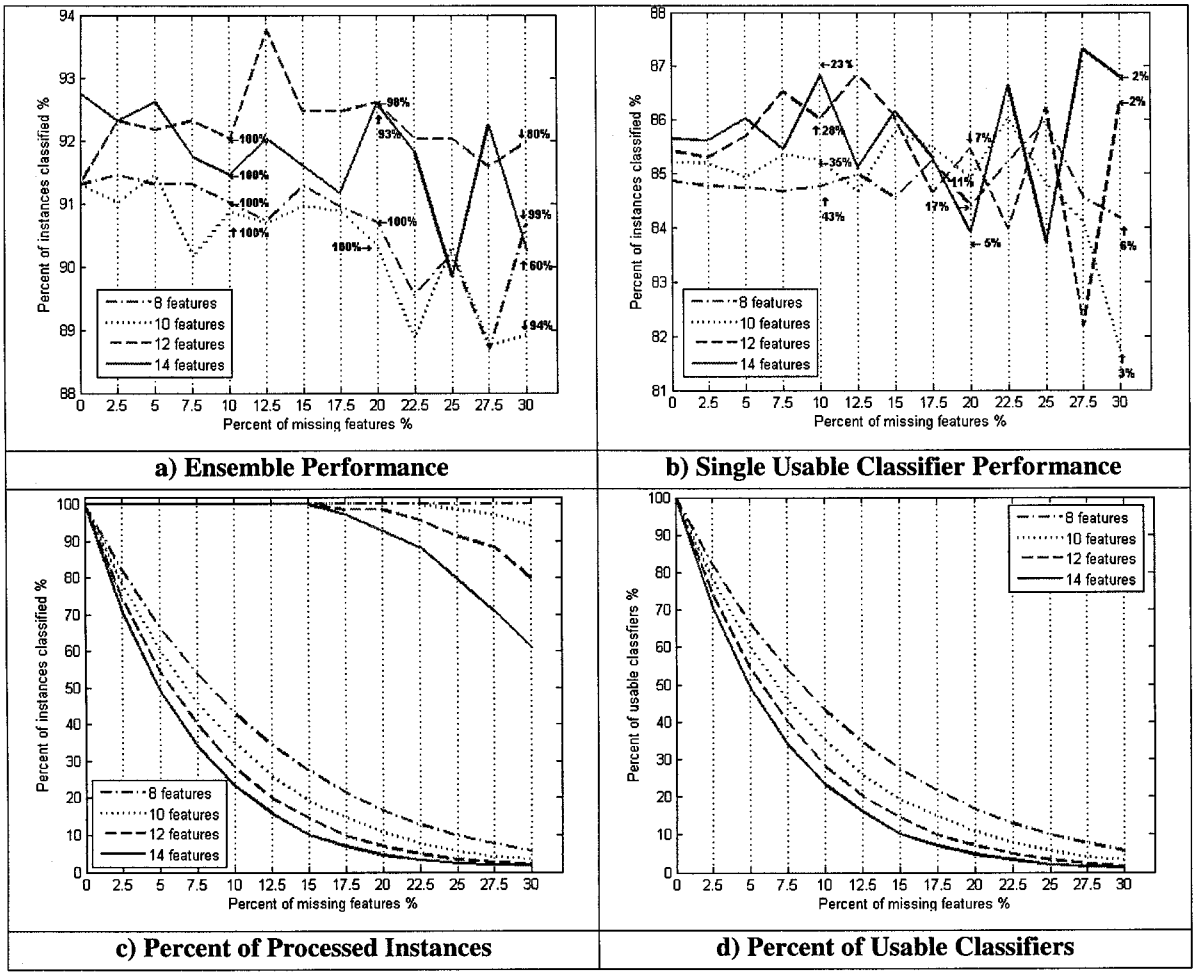

Figure 5.8: Learn^{++}.MF Performance Results on ION Dataset

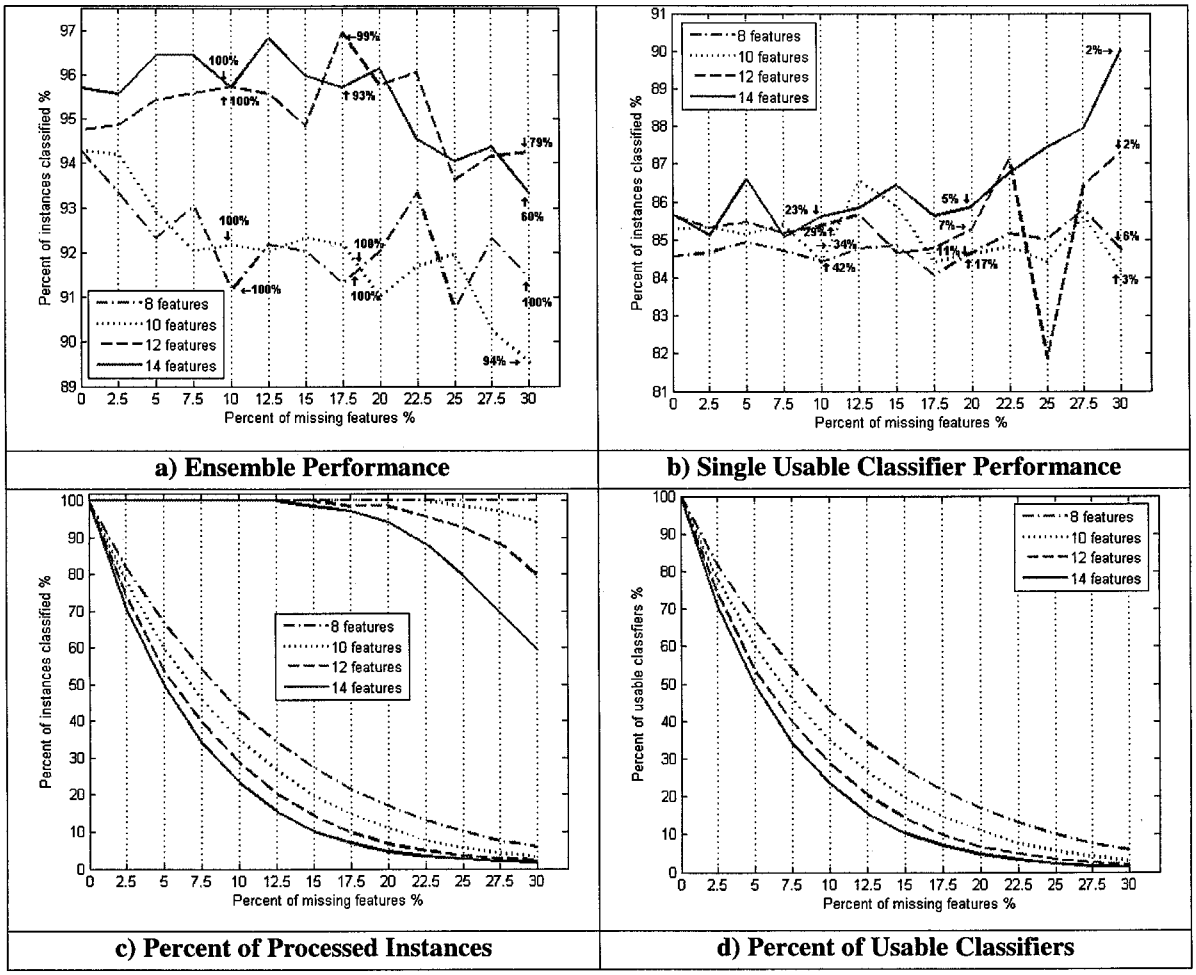

Figure 5.9: Learn^{++}.MFv2 Performance Results on ION Dataset

5.2.4 Wine Dataset

This dataset consists of chemical analyses results of Italian wines grown in the same region, but derived from three different cultivars. The analyses determined the quantities of 13 constituents found in each of the three types of wine. The algorithms were evaluated with five values of *nof* used for training: 3, 4, 5, 6 and 7 features out of 13. *T* was set to 200 classifiers. Table 5.6 shows the performance of both the algorithms Learn^{++}.MF and Learn^{++}.MFv2 on the Wine dataset. Figure 5.10 and Figure 5.11 show the characteristic performance and trends of the respective algorithms on the Wine dataset.

We report our findings on Learn^{++}.MF first. On this somewhat simpler wine database, using 4, 5, 6 or 7 features in training, allowed Learn^{++}.MF to reach 100% classification performance when no features were missing. It did maintain a very high performance for up to 10%-20% missing features (note the confidence intervals), with very little performance drop for larger percentages of missing features. In general, similar trends can be observed on this database as well, such as a higher initial performance with a larger *nof*, with a steeper decline in the performance as well as the percent instances that can be processed. We note that the performance of the Learn^{++}.MF ensemble trained on *nofs* = 3/13 and 4/13 features remained relatively steady, and was able to process the entire dataset (100% of instances) even when 30% of features were missing. The performances for all values of % missing features were within each other's confidence intervals. This reveals the redundancy of features spread across in this dataset. Learn^{++}.MFv2 was able to achieve 100% even with using *nof* = 3/13 features. Hence, one could possibly use a smaller *nof* to train an ensemble using Learn^{++}.MFv2 and be able to

achieve the same initial performance as the ensembles trained under a larger *nofs* using Learn^{++}.MF as seen here. The percent of instances that could be classified by the algorithms for the various *nofs* under consideration did not vary.

Table 5.6: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the Wine Dataset

	(nof = 3/13)				*(nof = 4/13)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	96.67 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100
2.50%	96.67 ± 1.12	100	99.00 ± 1.15	100	99.00 ± 1.15	100	99.67 ± 0.75	100
5.00%	95.33 ± 1.23	100	98.67 ± 1.23	100	99.00 ± 1.15	100	99.67 ± 0.75	100
7.50%	95.67 ± 1.15	100	98.33 ± 1.68	100	99.00 ± 1.15	100	99.33 ± 1.00	100
10.00%	98.00 ± 1.23	100	98.33 ± 1.26	100	99.00 ± 1.15	100	100.00 ± 0.00	100
12.50%	96.33 ± 1.35	100	98.00 ± 1.23	100	98.00 ± 1.67	100	98.33 ± 1.26	100
15.00%	96.67 ± 1.59	100	98.00 ± 1.23	100	98.67 ± 1.23	100	98.33 ± 1.26	100
17.50%	97.00 ± 1.76	100	99.00 ± 1.15	100	97.33 ± 1.88	100	99.33 ± 1.00	100
20.00%	96.00 ± 1.00	100	98.00 ± 1.67	100	96.67 ± 1.12	100	98.67 ± 1.23	100
22.50%	97.33 ± 1.51	100	96.67 ± 1.59	100	98.33 ± 1.26	100	99.33 ± 1.00	100
25.00%	97.00 ± 1.76	100	96.00 ± 1.51	100	98.67 ± 1.67	100	97.33 ± 1.88	100
27.50%	95.67 ± 1.96	100	96.33 ± 1.35	100	97.33 ± 1.51	100	97.32 ± 1.51	100
30.00%	97.67 ± 1.15	100	98.00 ± 1.23	100	97.67 ± 1.61	99	97.67 ± 1.96	100
	(nof = 5/13)				*(nof = 6/13)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100	100.00 ± 0.00	100
2.50%	100.00 ± 0.00	100	99.67 ± 0.75	100	100.00 ± 0.00	100	99.67 ± 0.75	100
5.00%	99.00 ± 1.15	100	99.67 ± 0.75	100	99.67 ± 0.75	100	99.67 ± 0.75	100
7.50%	99.00 ± 1.15	100	99.67 ± 0.75	100	99.67 ± 0.75	100	99.67 ± 0.75	100
10.00%	98.67 ± 1.23	100	99.67 ± 0.75	100	99.33 ± 1.00	100	100.00 ± 0.00	100
12.50%	98.33 ± 1.26	100	98.67 ± 1.23	100	99.67 ± 0.75	99	98.67 ± 1.23	100
15.00%	97.00 ± 1.76	100	98.33 ± 1.26	100	98.67 ± 1.67	100	98.67 ± 1.23	99
17.50%	97.33 ± 1.51	100	98.67 ± 1.23	100	99.00 ± 1.15	100	98.67 ± 0.75	99
20.00%	98.67 ± 1.23	100	99.33 ± 1.00	100	98.31 ± 1.27	99	99.33 ± 1.00	97
22.50%	97.97 ± 1.25	99	98.66 ± 1.24	100	97.93 ± 2.06	97	98.31 ± 1.27	98
25.00%	97.32 ± 1.51	99	97.31 ± 1.01	99	97.57 ± 2.67	96	98.60 ± 1.77	95
27.50%	95.88 ± 2.30	98	97.99 ± 2.31	99	96.39 ± 2.07	92	97.47 ± 2.14	93
30.00%	96.12 ± 2.27	95	98.32 ± 1.26	99	97.34 ± 1.81	88	97.35 ± 1.75	89
	(nof = 7/13)							
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process				
0.00%	100.00 ± 0.00	100	100.00 ± 0.00	100				
2.50%	99.67 ± 0.75	100	100.00 ± 0.00	100				
5.00%	99.67 ± 0.75	100	100.00 ± 0.00	100				
7.50%	99.33 ± 1.51	100	99.33 ± 1.00	100				
10.00%	99.33 ± 1.00	100	98.67 ± 1.67	100				
12.50%	98.67 ± 1.67	99	99.67 ± 0.75	99				
15.00%	98.30 ± 1.28	98	99.67 ± 1.02	98				
17.50%	98.96 ± 1.19	95	97.63 ± 1.65	98				
20.00%	98.94 ± 1.22	95	98.60 ± 1.30	96				
22.50%	98.11 ± 1.43	91	97.87 ± 2.13	93				
25.00%	96.59 ± 2.01	88	98.09 ± 1.94	88				
27.50%	96.48 ± 2.70	82	98.45 ± 1.43	83				
30.00%	97.41 ± 2.53	76	97.43 ± 2.55	79				

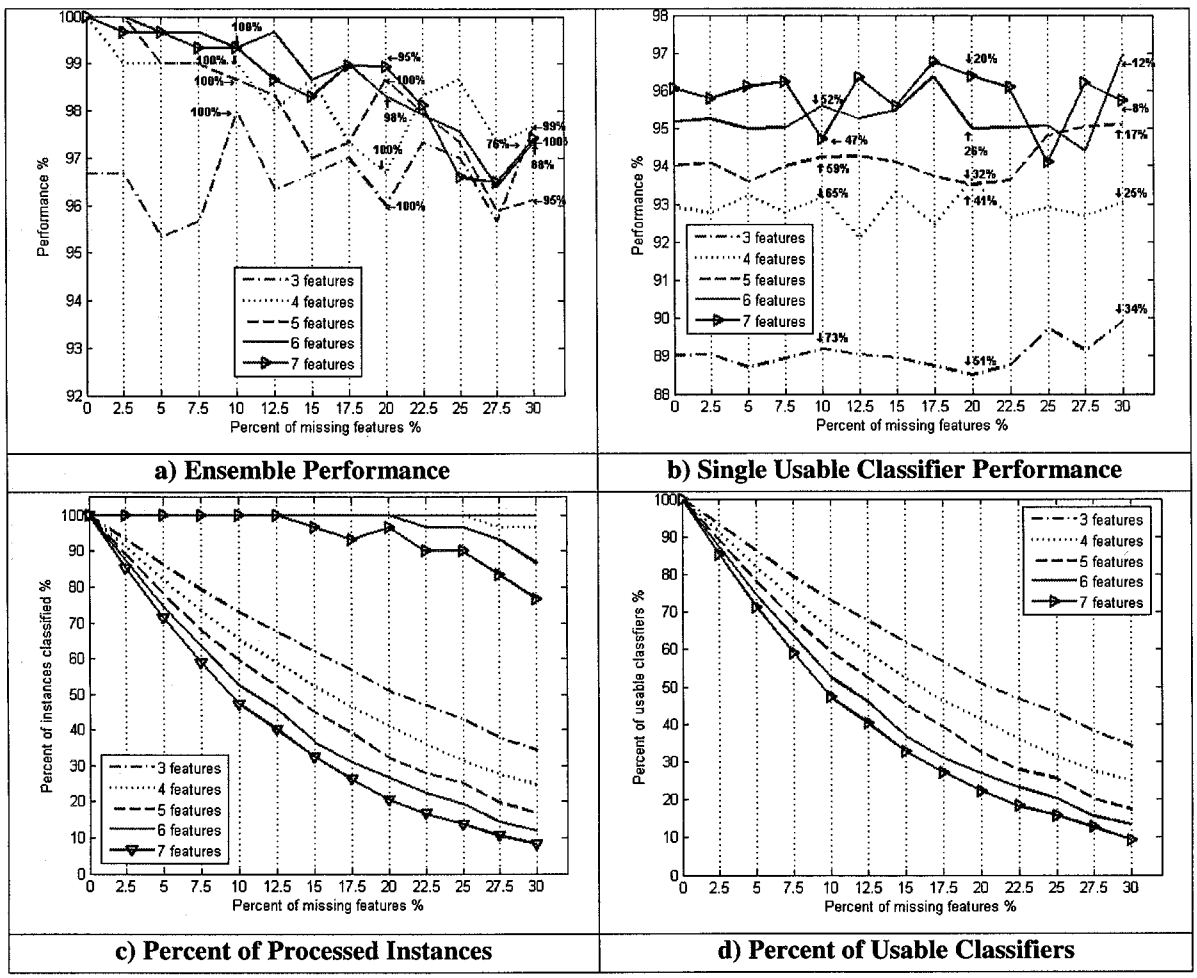

Figure 5.10: Learn^{++}.MF Performance Results on Wine Dataset

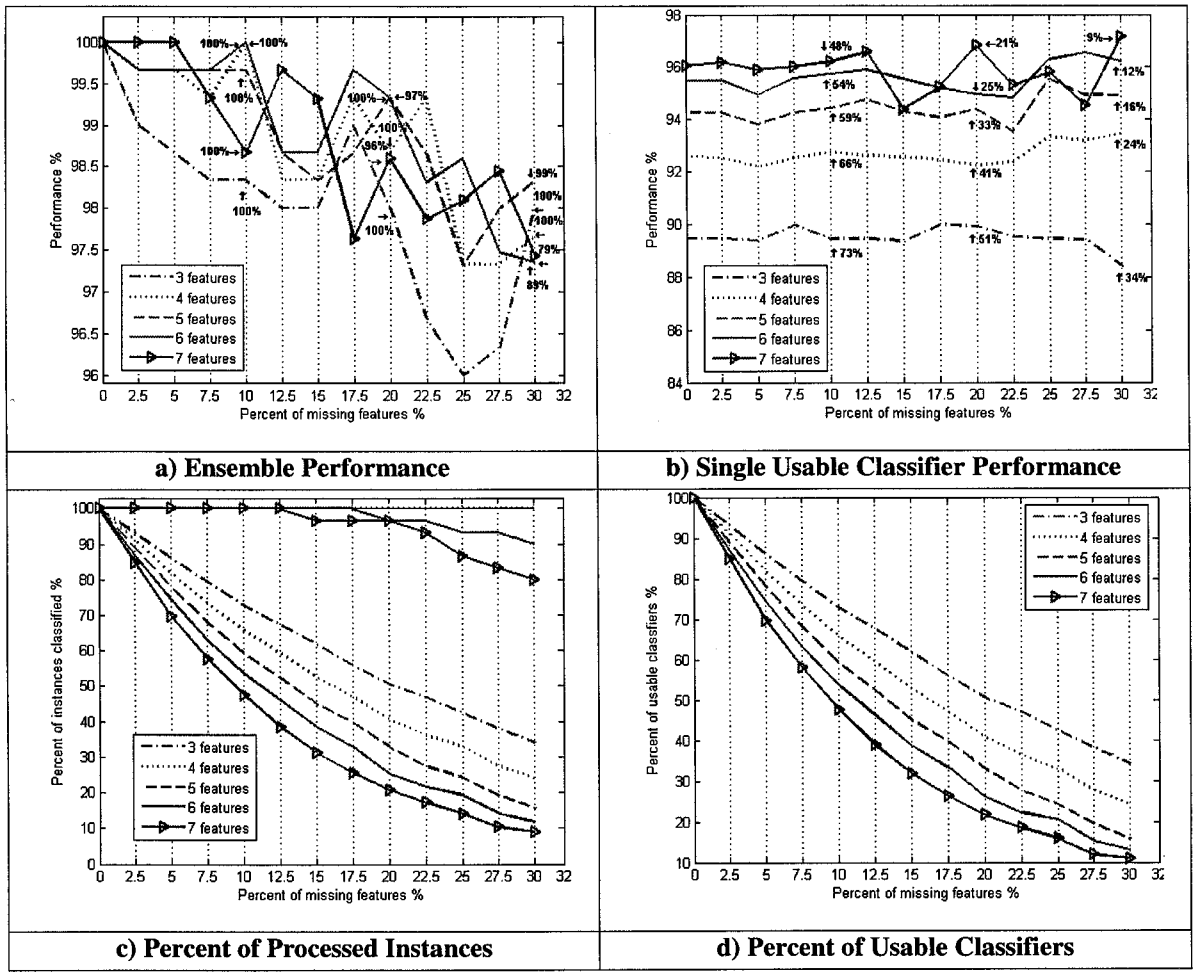

Figure 5.11. Learn^{++}.MFv2 Performance Results on Wine Dataset

5.2.5 Dermatology (DERMA) Dataset

This benchmark database deals with differential diagnosis of erythemato-squamous diseases. The diseases in this group are psoriasis, seboreic dermatitis, lichen planus, pityriasis rosea, chronic dermatitis, and pityriasis rubra pilaris. Usually a biopsy is necessary for the diagnosis, and unfortunately these diseases share many histopathological features. Previous experiments have shown that using optimized classifiers trained using all features tested on the test dataset with no missing features can achieve 100%, setting the benchmark target for this dataset. The algorithms were evaluated by training classifiers using four different values of *nofs*, 8, 10, 12 and 14 out of 34 available attributes. 1000 classifiers were generated for this data.

The performance results of both Learn^{++}.MF and Learn^{++}.MFv2 are shown in Table 5.7 below. The behavioral trends of the algorithms Learn^{++}.MF and Learn^{++}.MFv2 can be seen in Figure 5.12 and Figure 5.13, respectively. The performance results and figures below do not reflect any major differences on this dataset for both of the algorithms. Hence, we generalize the discussion below to entail both algorithms for this dataset. Unless mentioned otherwise, any observation reported is true for both algorithms.

The performance trends of the algorithm on the DERMA dataset were comparable to the trends reported on the other datasets. Again, the algorithm was able to reach 98% correct classification using 8-14 features for each ensemble, indicating that many of the features in this database may in fact be redundant. The performances were very close to each other, regardless of the number of features used for each algorithm, though the most stable response came from the ensemble using the fewest number of features. While the performances for both algorithms declined with the introduction of more missing

features, the ensembles trained on the lower *nofs* (i.e. 8/34) had a higher performance than their counterparts. Similar to previous cases, while the performances were similar for different number of features, the percent of instances that could be processed by the respective ensembles trained under the various *nofs* were not. Using fewer features for training allowed the algorithms to process a larger percentage of the instances: 99-100% of the data could be processed even when 30% of the features missing if the classifiers were trained with 8 features; however, that figure quickly dropped to 82% when they were trained with 12 features for both algorithms.

This dataset was also used by [23] mentioned earlier in the introduction, which also employed an ensemble of classifiers approach, but training single-class classifiers with a single feature at a time. The ensemble performances reported in [23] were in the 50-70% range even for 10% missing features or less. Comparing these results to those presented in [23], we observe that using a random subset of features significantly outperforms (95-98% vs. 40-70% as reported in [23]), a similar approach that uses a single feature at a time. However, such a substantial performance improvement comes at an increased, but justifiable computational cost (1000 classifiers for Learn^{++}.MF and Learn^{++}.MFv2 vs. 210 classifiers reported in [23]).

Table 5.7: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the DERMA Dataset

	(nof = 8/34)				*(nof = 10/34)*			
% Missing Features	**% v1 Mean Performance**	**% v1 Process**	**% v2 Mean Performance**	**% v2 Process**	**% v1 Mean Performance**	**% v1 Process**	**% v2 Mean Performance**	**% v2 Process**
0.00%	97.32 ± 0.00	100	98.21 ± 0.00	100	98.21 ± 0.00	100	98.21 ± 0.00	100
2.50%	97.95 ± 0.53	100	97.77 ± 0.34	100	98.12 ± 0.56	100	98.12 ± 0.47	100
5.00%	97.50 ± 0.40	100	97.59 ± 0.43	100	98.21 ± 0.67	100	98.48 ± 0.31	100
7.50%	97.77 ± 0.69	100	97.05 ± 0.68	100	97.41 ± 0.56	100	98.04 ± 0.72	100
10.00%	97.05 ± 0.85	100	97.68 ± 0.75	100	97.86 ± 0.69	100	98.04 ± 0.72	100
12.50%	96.96 ± 0.96	100	97.50 ± 0.66	100	97.32 ± 0.80	100	97.23 ± 0.82	100
15.00%	96.43 ± 0.80	100	96.70 ± 0.80	100	96.78 ± 0.81	100	98.12 ± 0.70	100
17.50%	97.32 ± 0.52	100	95.80 ± 0.85	100	97.05 ± 1.04	100	97.77 ± 0.81	100
20.00%	96.25 ± 1.16	100	95.71 ± 0.84	100	95.62 ± 1.19	100	97.41 ± 1.11	100
22.50%	96.07 ± 1.32	100	96.16 ± 1.13	100	96.14 ± 1.13	100	96.05 ± 0.69	99
25.00%	95.18 ± 1.21	100	94.18 ± 1.25	100	95.00 ± 1.38	98	95.20 ± 1.44	98
27.50%	95.26 ± 1.09	100	95.00 ± 0.81	100	93.86 ± 1.27	97	94.18 ± 0.98	97
30.00%	94.60 ± 1.17	99	94.54 ± 1.05	100	92.29 ± 1.67	96	94.22 ± 1.57	94
	(nof = 12/34)				*(nof = 14/34)*			
% Missing Features	**% v1 Mean Performance**	**% v1 Process**	**% v2 Mean Performance**	**% v2 Process**	**% v1 Mean Performance**	**% v1 Process**	**% v2 Mean Performance**	**% v2 Process**
0.00%	98.21 ± 0.00	100	99.11 ± 0.00	100	98.21 ± 0.00	100	98.21 ± 0.00	100
2.50%	98.39 ± 0.50	100	99.02 ± 0.36	100	98.21 ± 0.43	100	98.57 ± 0.33	100
5.00%	98.57 ± 0.45	100	98.39 ± 0.66	100	98.48 ± 0.53	100	98.21 ± 0.43	100
7.50%	98.48 ± 0.53	100	98.21 ± 0.52	100	98.30 ± 0.63	100	98.48 ± 0.74	100
10.00%	97.95± 0.68	100	97.86 ± 0.75	100	97.59 ± 0.90	100	97.94 ± 0.43	100
12.50%	98.21 ± 0.67	100	98.04 ± 0.27	100	98.21 ± 0.80	100	97.76 ± 0.81	100
15.00%	98.12 ± 1.10	100	97.32 ± 0.67	100	97.83 ± 0.93	99	97.02 ± 0.87	99
17.50%	97.75 ± 1.26	99	97.22 ± 1.22	100	97.43 ± 1.14	97	96.25 ± 1.15	98
20.00%	96.53 ± 0.80	98	96.56 ± 1.24	98	96.23 ± 1.04	93	95.89 ± 1.20	93
22.50%	95.82 ± 0.73	96	96.22 ± 1.43	97	94.96 ± 1.47	87	95.01 ± 1.48	88
25.00%	93.41 ± 1.39	93	94.77 ± 1.48	94	93.37 ± 1.48	81	94.39 ± 2.27	80
27.50%	93.08 ± 1.78	88	93.57 ± 1.74	88	94.15 ± 1.34	71	92.83 ± 1.60	73
30.00%	92.50 ± 2.61	82	93.64 ± 2.05	82	91.03 ± 2.19	61	92.82 ± 2.17	63

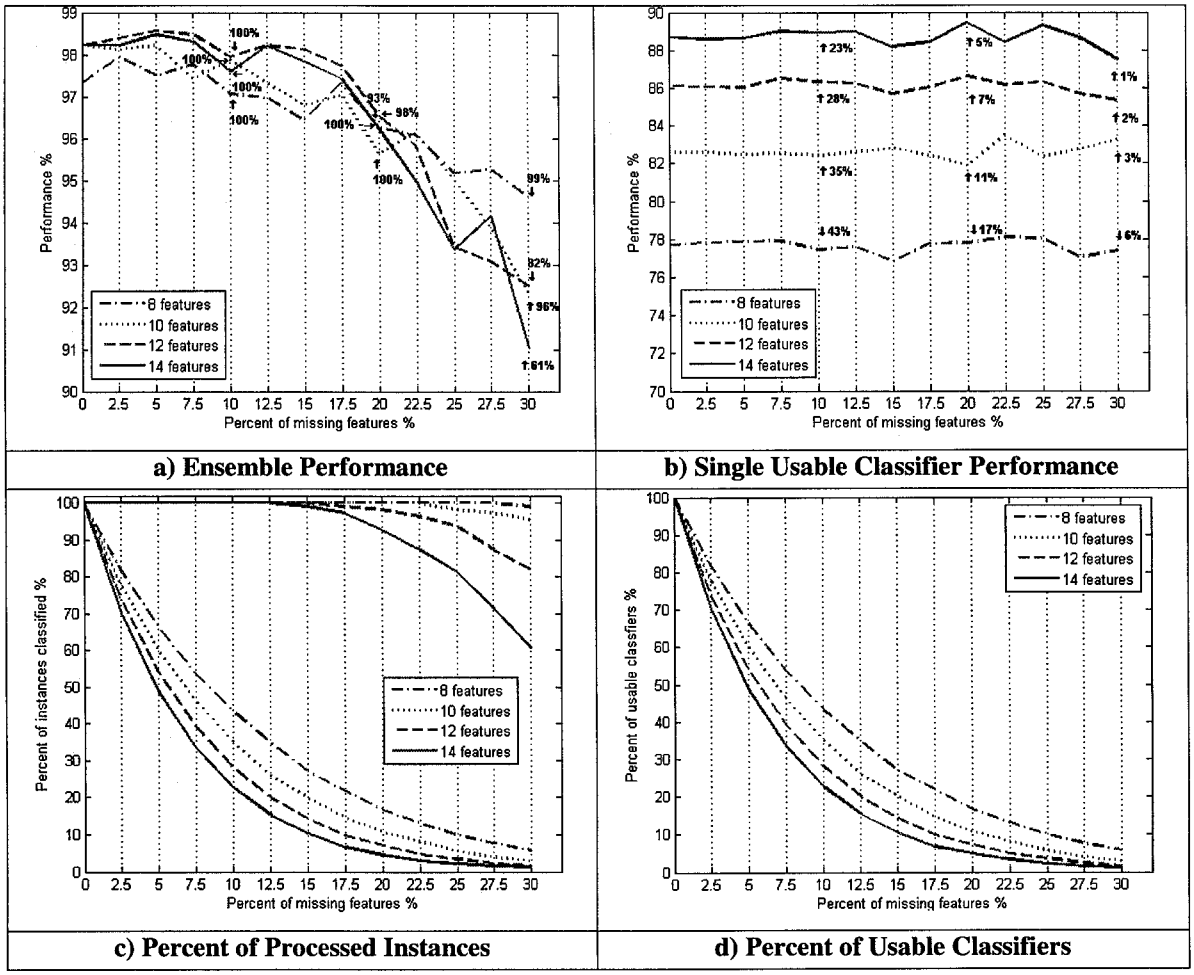

Figure 5.12: Learn^{++}.MF Performance Results on DERMA Dataset

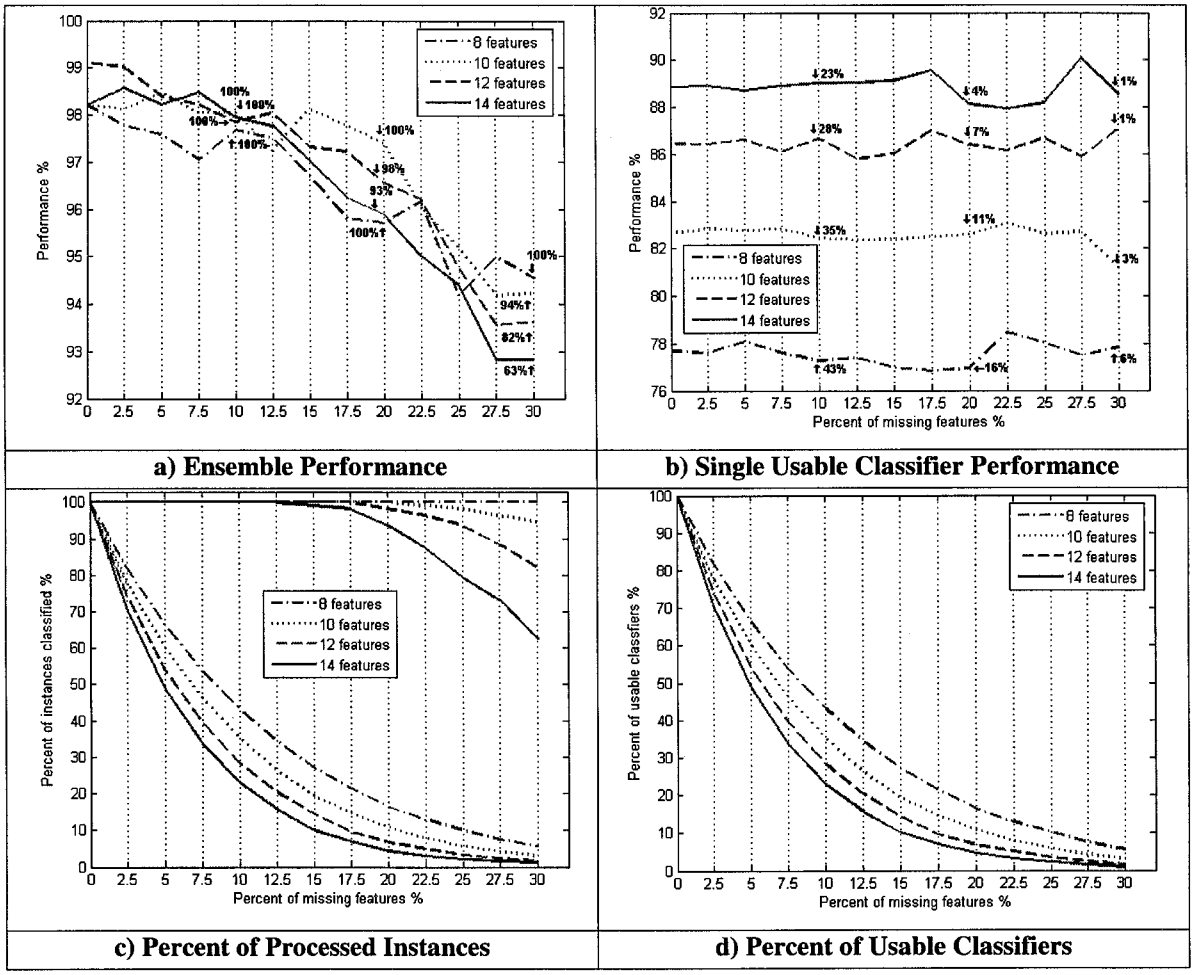

Figure 5.13: Learn^{++}.MFv2 Performance Results on DERMA Dataset

5.2.6 *Wisconsin Breast Cancer (WBC) Dataset*

This database, originally created at The University of Wisconsin, Madison, was also obtained from the UCI machine learning repository [71]. The database consists of 30 features from two classes of breast tumors: *benign* and *malignant*. Previous experiments have shown that using optimized classifiers trained using all features tested on the test dataset with no missing features can achieve 100%, setting the benchmark target for this dataset. Learn^{++}.MF and Learn^{++}.MFv2 were evaluated by training classifiers using four different *nofs*: 10, 12, 14, and 16 out of 30 available attributes. 1000 classifiers were generated for this dataset. The performance results of both Learn^{++}.MF and Learn^{++}.MFv2 on the WBC dataset are shown in Table 5.8. The general behavioral trends of the algorithms Learn^{++}.MF and Learn^{++}.MFv2 can be seen in Figure 5.14 and Figure 5.15, respectively.

The performance results and figures, do not reflect any major differences on this dataset between the two algorithms. Similar to the approach taken above for the DERMA dataset, we generalize the discussion below to entail Learn^{++}.MF and Learn^{++}.MFv2, unless mentioned otherwise.

While both algorithms are able to achieve 95-96% correct classification when the ratio of missing features is low, the ensembles trained on a larger *nof* experienced a faster rate of decline in terms of performance, as observed on previous datasets. The ensemble performance of both algorithms for *nof* = 10/30 features was much higher (i.e. ~93-94%) when 30% of the feature space was missing as opposed to lower ensemble generalization performance (i.e. ~89%) of both algorithms for *nof* = 14/30 features. The generalization performance of the ensemble trained on a lower *nof* had a higher classification even when

the ratio of missing features increased. However, the extent of the difference varies between all datasets. We can relate the higher performance of the ensembles trained on a lower *nof* to the number of usable classifiers. As the percent of missing features increase in the dataset, the number of usable classifiers decreases for any *nof*. However, the decline in the number of usable classifier is significantly steeper for ensembles trained on a higher *nof* as seen in Figure 5.14d and Figure 5.15d, respectively. Hence, for any given classifiable instance, an ensemble trained with a lower *nof* will be more likely to have more usable classifiers to be able to classify it. This explains why ensembles trained on a lower *nof* generally have a slower rate of decline in performance.

Table 5.8: Learn^{++}.MF and Learn^{++}.MFv2 Performances of the WBC Dataset

	(nof = 10/30)				*(nof = 12/30)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	95.50 +/- 0.00	100	97.00 ± 0.00	100	96.00 +/- 0.00	100	97.00 ± 0.00	100
2.50%	95.40 +/- 0.15	100	96.90 ± 0.28	100	95.75 +/- 0.19	100	96.60 ± 0.23	100
5.00%	95.55 +/- 0.36	100	96.80 ± 0.30	100	95.45 +/- 0.26	100	96.55 ± 0.26	100
7.50%	95.50 +/- 0.24	100	97.00 ± 0.24	100	95.40 +/- 0.37	100	96.05 ± 0.49	100
10.00%	95.50 +/- 0.45	100	96.85 ± 0.51	100	95.65 +/- 0.29	100	96.15 ± 0.53	100
12.50%	94.90 +/- 0.56	100	96.30 ± 0.38	100	95.05 +/- 0.43	100	96.35 ± 0.41	100
15.00%	94.90 +/- 0.65	100	96.60 ± 0.41	100	94.64 +/- 0.58	100	96.04 ± 0.39	100
17.50%	94.74 +/- 0.42	100	96.34 ± 0.50	100	95.04 +/- 0.59	99	95.39 ± 0.33	99
20.00%	94.87 +/- 0.62	99	96.53 ± 0.36	100	94.59 +/- 0.71	97	94.92 ± 0.65	97
22.50%	94.84 +/- 0.77	99	96.17 ± 0.98	99	94.26 +/- 0.90	94	94.28 ± 0.75	94
25.00%	93.90 +/- 0.71	98	96.32 ± 0.57	98	94.32 +/- 0.99	91	94.44 ± 1.01	91
27.50%	93.66 +/- 0.89	95	95.87 ± 1.01	95	91.52 +/- 0.67	84	93.51 ± 1.11	84
30.00%	93.43 +/- 0.82	92	94.88 ± 0.81	92	91.89 +/- 1.10	75	92.18 ± 1.57	75
	(nof = 14/30)				*(nof = 16/30)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	95.50 ± 0.00	100	97.50 ± 0.00	100	95.00 ± 0.00	100	97.50 ± 0.00	100
2.50%	95.30 ± 0.30	100	97.00 ± 0.41	100	94.75 ± 0.25	100	97.45 ± 0.26	100
5.00%	95.20 ± 0.35	100	97.00 ± 0.41	100	94.60 ± 0.37	100	97.45 ± 0.26	100
7.50%	95.00 ± 0.45	100	96.95 ± 0.54	100	94.58 ± 0.14	100	97.19 ± 0.42	100
10.00%	95.00 ± 0.48	100	96.85 ± 0.41	100	94.11 ± 0.63	99	96.89 ± 0.70	99
12.50%	95.21 ± 0.89	99	96.68 ± 0.43	99	93.58 ± 0.75	97	96.20 ± 0.62	98
15.00%	94.60 ± 0.74	98	95.83 ± 0.76	98	93.16 ± 0.69	93	95.22 ± 1.13	94
17.50%	93.58 ± 0.72	95	95.10 ± 0.79	95	91.70 ± 1.50	88	94.95 ± 1.25	87
20.00%	93.00 ± 1.43	91	94.34 ± 0.79	91	91.35 ± 1.51	78	93.59 ± 1.44	79
22.50%	91.76 ± 1.05	85	93.38 ± 1.18	85	90.12 ± 0.88	68	93.44 ± 1.29	68
25.00%	89.25 ± 1.28	76	92.66 ± 1.80	76	90.01 ± 1.21	56	91.01 ± 1.48	57
27.50%	88.87 ± 1.62	67	91.10 ± 1.85	67	86.51 ± 1.33	46	90.63 ± 2.51	46
30.00%	90.57 ± 1.63	54	91.46 ± 1.22	54	89.08 ± 2.44	35	89.50 ± 2.88	37

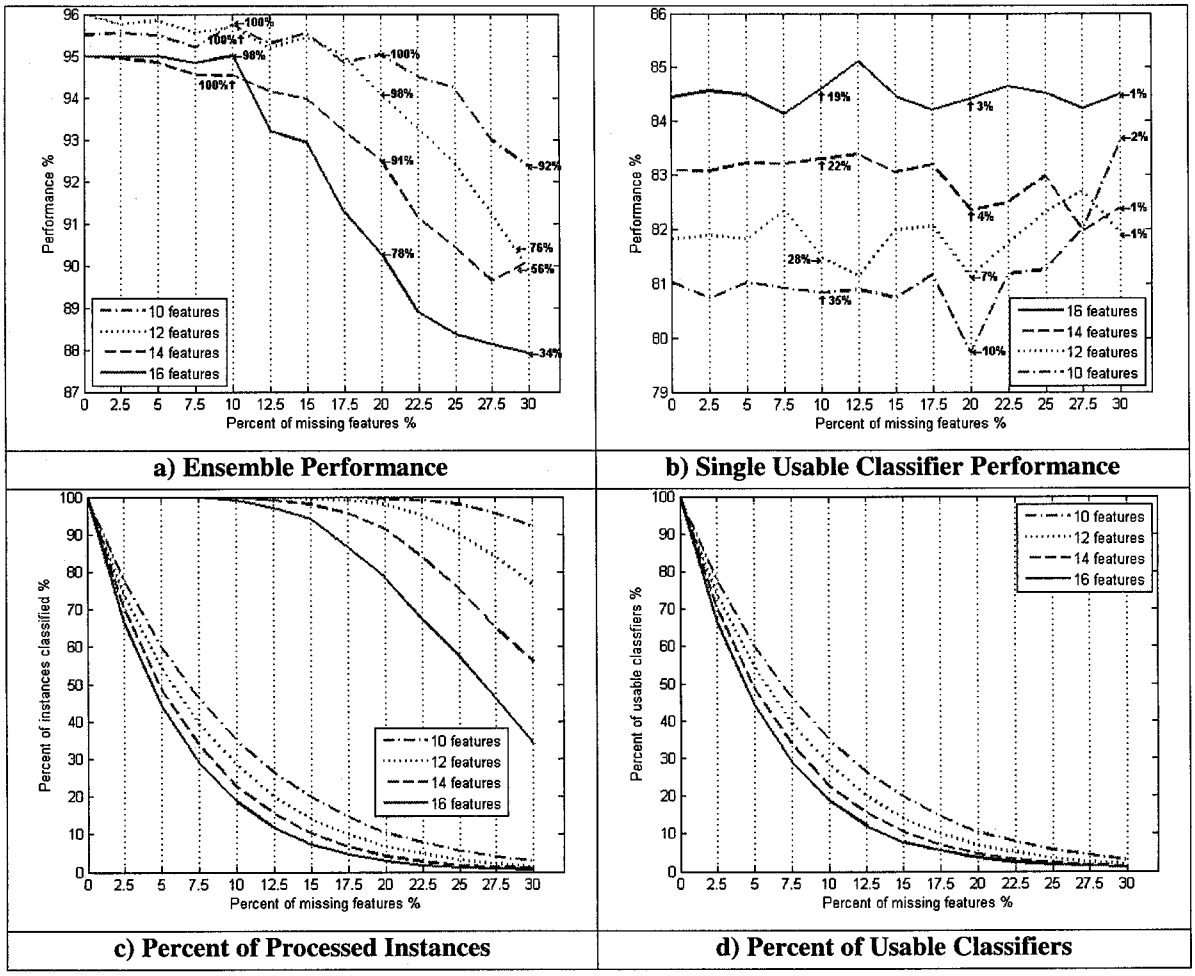

Figure 5.14: Learn^{++}.MF Performance Results on WBC Dataset

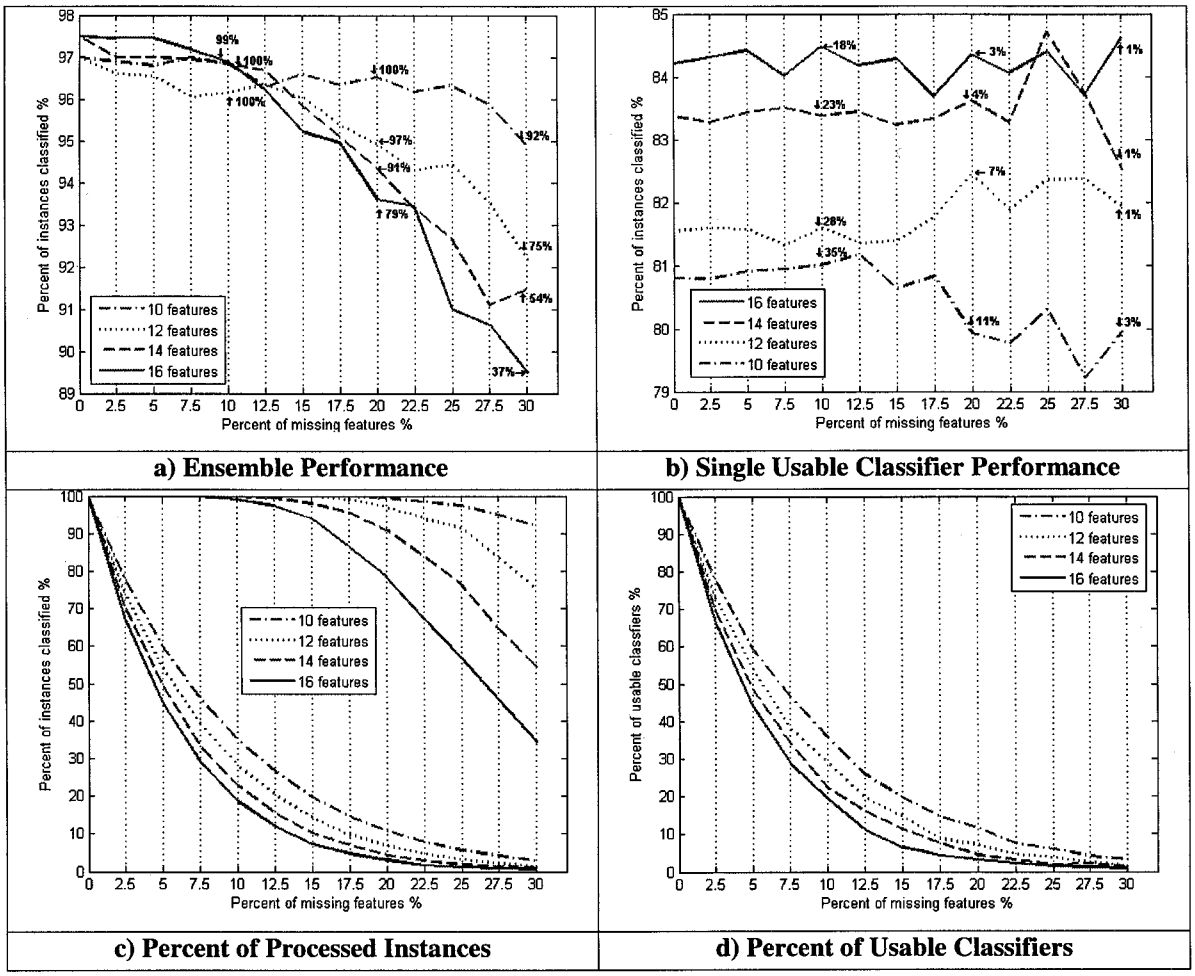

Figure 5.15: Learn^{++}.MFv2 Performance Results on WBC Dataset

5.2.7 *Water Dataset*

This benchmark database contains the daily measures from sensors in an urban waste water treatment plant. The objective is to classify the operational state of the plant in order to predict faults through the state variables of the plant at each of the stages of the treatment process. T was set to 1000 classifiers. Previous trials using an optimized set of classifiers trained on all features have shown near 80% performance results. The algorithms were evaluated on 4 values of *nof* = 12, 14, 16 and 18 out of the 38 available attributes.

The performance results of both Learn^{++}.MF and Learn^{++}.MFv2 on the Water dataset are shown in Table 5.9. The general trends of the algorithms Learn^{++}.MF and Learn^{++}.MFv2 can be seen in Figure 5.16 and Figure 5.17, respectively. We begin by describing the performance and trends of the algorithm Learn^{++}.MF. The performances were very close to each other regardless of the number of features used. Learn^{++}.MF was able to reach at least 78% correct classification using 12-20 features for each classifier. Note that the dynamic range in the performance axis for the ensemble and a single usable classifier shown in Figure 5.16a and Figure 5.16b, respectively, are fairly narrow. While the performances were similar for different number of features, the percent of instances that could be processed by the respective ensembles were not. Similar to the previous case, using fewer features for training allowed the algorithm to process a larger percentage of the instances. The percent of usable classifiers for the 5 values of *nofs* used on this dataset are in Figure 5.16d. *Occam's Razor* states that *entities should not be multiplied without necessity*. Hence, in the case of using a dataset with redundant features, training an ensemble with a lower *nof* may have additional benefits, as seen

here. Once again we observe the percent of usable classifiers with an increase of missing features introduced to the dataset favors the ensemble trained on fewer features.

Figure 5.17 describes the simulation results of the algorithm Learn^{++}.MFv2 in a similar manner described previously. From Table 5.9 and Figure 5.17a, indicate an advantage in using Learn^{++}.MFv2 on this dataset, when compared to its predecessor. The ensemble performances of Learn^{++}.MFv2 achieve a 2-4% increase over its predecessor with no features missing respectively. However, it is able to maintain this advantage for some ratio of missing features as shown in Table 5.9. The Learn^{++}.MFv2 algorithm was able to achieve the target performance of ~80% for most of the *nof* values under consideration. The ensemble performance for the Learn^{++}.MFv2 using a larger *nof* was able to close the gap towards the target performance.

Table 5.9: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the Water Dataset

	(nof = 12/38)				*(nof = 14/38)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	77.96 ± 0.00	100	79.42 ± 0.00	100	77.42 ± 0.00	100	79.96 ± 0.00	100
2.50%	77.85 ± 0.35	100	79.20 ± 0.41	100	77.42 ± 0.40	100	80.01 ± 0.28	100
5.00%	77.31 ± 0.24	100	79.20 ± 0.55	100	77.47 ± 0.49	100	79.26 ± 0.63	100
7.50%	77.63 ± 0.37	100	78.99 ± 0.40	100	77.74 ± 0.58	100	79.58 ± 0.70	100
10.00%	77.47 ± 0.73	100	79.26 ±0.41	100	76.56 ± 0.79	100	78.99 ± 0.86	100
12.50%	76.45 ± 0.70	100	78.77 ± 1.18	100	77.00 ± 0.94	100	79.61 ± 0.72	100
15.00%	76.60 ± 1.15	100	78.27 ± 0.60	100	76.90 ± 0.89	100	79.62 ± 1.15	100
17.50%	77.09 ± 0.90	100	78.43 ± 1.25	100	76.69 ± 0.98	98	78.90 ± 1.20	98
20.00%	76.57 ± 0.60	99	78.21 ± 1.02	99	77.73 ± 1.00	96	78.05 ± 1.00	97
22.50%	76.18 ± 1.04	98	77.72 ± 1.25	98	76.02 ± 2.26	91	77.75 ± 1.09	91
25.00%	74.79 ± 0.98	95	78.44 ± 1.90	95	73.89 ± 1.36	85	76.77 ± 2.36	85
27.50%	75.45 ± 1.79	89	77.47 ± 1.41	92	74.64 ± 2.11	77	77.21 ± 1.58	76
30.00%	74.93 ± 1.70	85	75.91 ± 1.37	85	73.39 ± 2.05	66	74.42 ± 2.19	67
	(nof = 16/38)				*(nof = 18/38)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	77.42 ± 0.00	100	80.54 ± 0.00	100	77.96 ± 0.00	100	80.75 ± 0.00	100
2.50%	78.12 ± 0.51	100	80.81 ± 0.20	100	78.23 ± 0.37	100	81.45 ± 0.73	100
5.00%	77.69 ± 0.87	100	81.62 ± 0.60	100	78.23 ± 0.61	100	80.81 ± 0.76	100
7.50%	77.80 ± 0.70	100	80.38 ± 0.57	100	77.77 ± 0.71	100	81.04 ± 0.82	100
10.00%	77.09 ± 0.92	100	81.28 ± 0.74	100	77.75 ± 1.26	99	82.19 ± 0.64	100
12.50%	76.43 ± 0.71	99	81.47 ± 1.04	99	77.18 ± 0.35	97	81.16 ± 1.08	98
15.00%	77.42 ± 1.05	97	80.98 ± 0.89	97	77.26 ± 1.21	93	80.59 ± 1.52	94
17.50%	77.34 ± 1.26	95	80.81 ± 1.37	94	76.41 ± 2.27	87	79.63 ± 1.54	86
20.00%	76.36 ± 1.27	88	79.86 ± 1.56	88	76.61 ± 1.39	78	79.97 ± 1.79	77
22.50%	76.68 ± 1.97	80	78.36 ± 1.68	79	74.92 ± 1.65	64	77.78 ± 2.06	67
25.00%	77.19 ± 0.88	69	77.66 ± 3.01	69	74.71 ± 3.36	51	80.60 ± 2.50	52
27.50%	74.50 ± 3.69	57	78.89 ± 3.09	56	76.03 ± 3.32	39	77.51 ± 3.58	41
30.00%	74.34 ± 3.81	45	79.49 ± 2.04	46	73.28 ± 4.55	28	78.18 ± 2.47	29

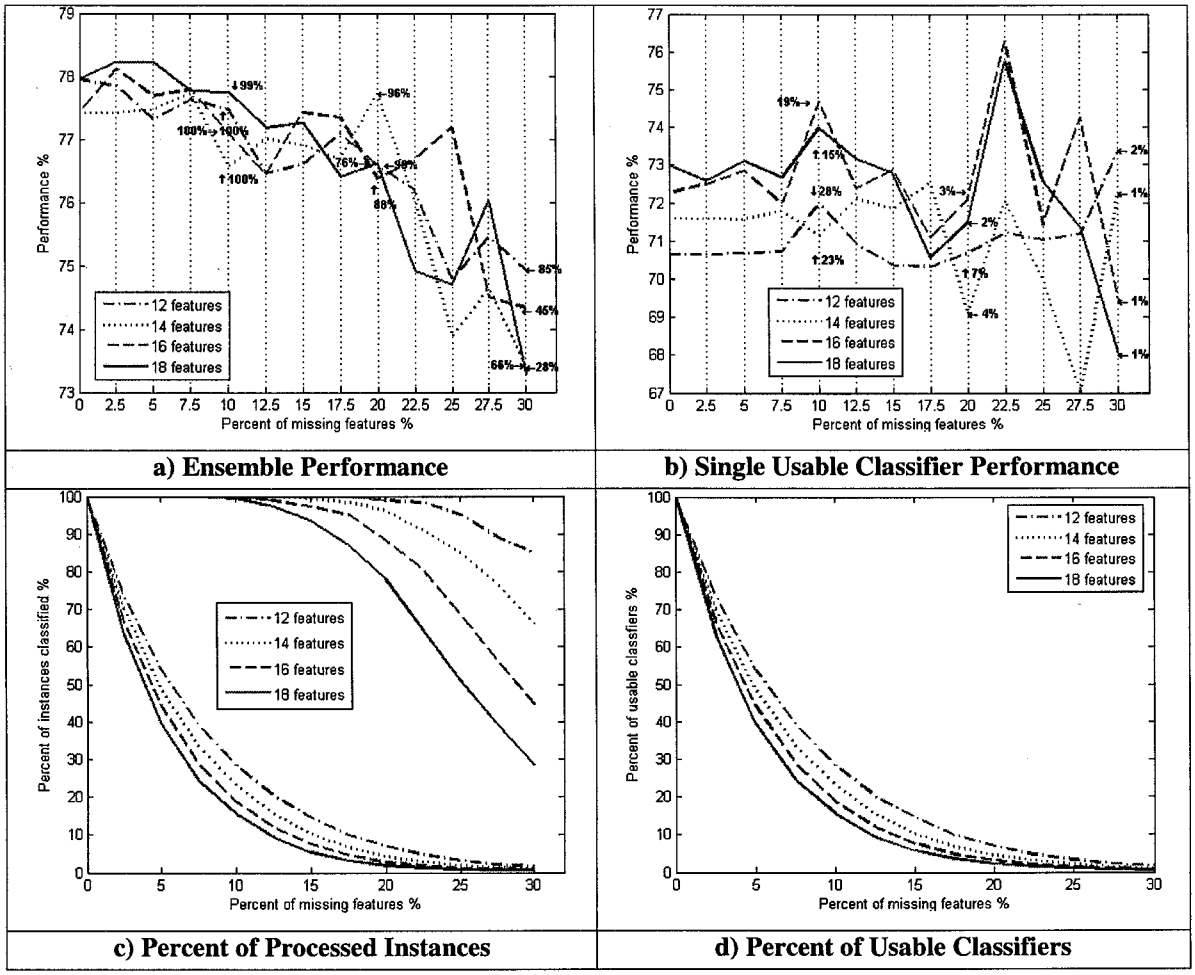

Figure 5.16: Learn^{++}.MF Performance Results on Water Dataset

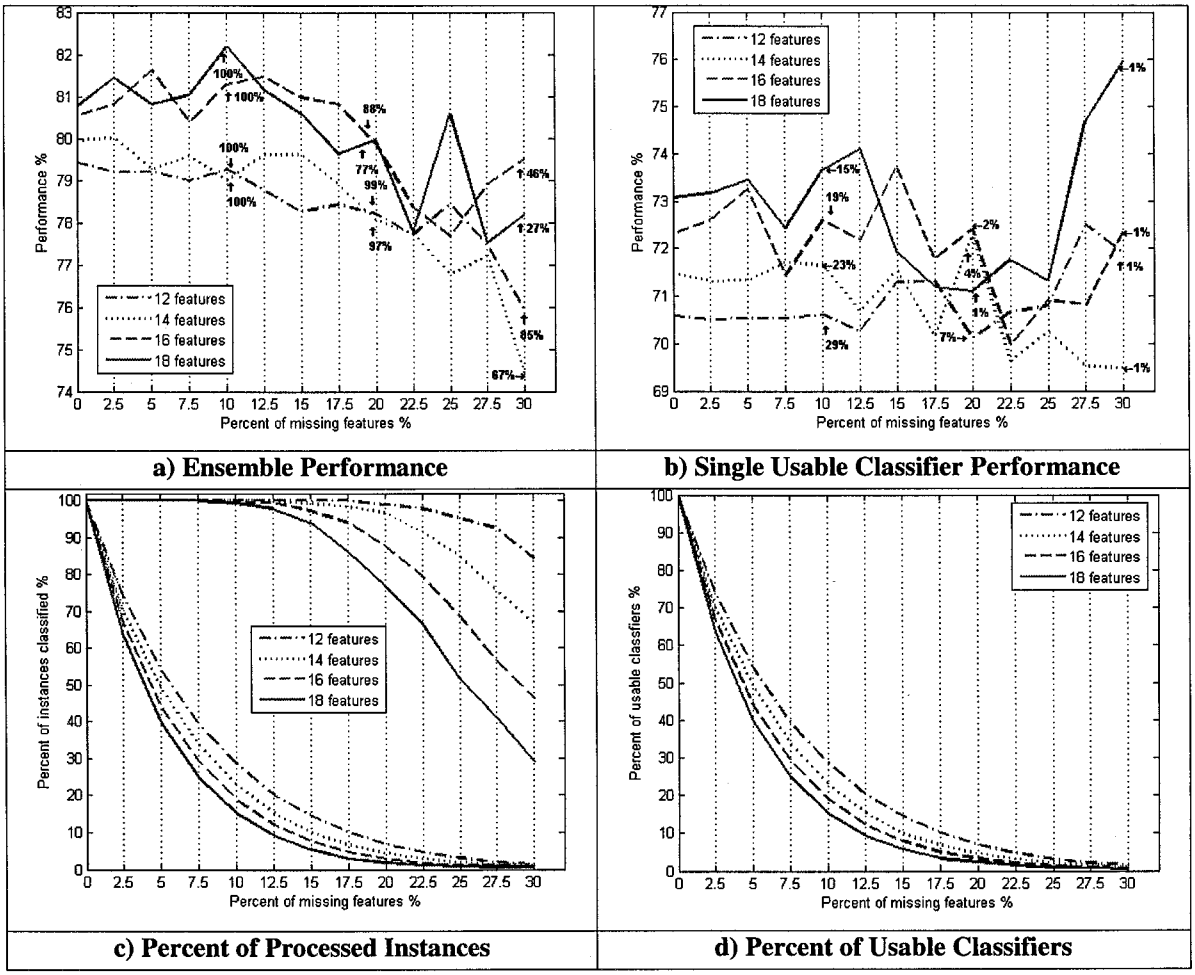

Figure 5.17: Learn^{++}.MFv2 Performance Results on Water Dataset

5.2.8 *Pen Digits Dataset*

This benchmark database originated from Bogazici University, Istanbul, Turkey was obtained from the UCI machine learning repository. It [71] consists of 10 digits, 0-9 having 16 attributes. The data was originally collected using a tablet. The tablet sends x and y tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 milliseconds. Four values of *nof* were considered; 6, 7, 8 and 9, out of 16, corresponding to 37.5%, 43.75%, 50% and 56.25% of the features, respectively. T was set to 250.

The performance results of both algorithms are shown in Table 5.10. The general behavioral trends of Learn^{++}.MF and Learn^{++}.MFv2 can be seen in Figure 5.18 and Figure 5.19, respectively. The performance results and figures, do not reflect any major differences on this dataset for between the algorithms. Again, we generalize the discussion below to entail Learn^{++}.MF and Learn^{++}.MFv2 for this dataset.

Both algorithms are able to achieve performance results in the 90% range for *nofs* = 7,8 and 9/16. With 30% of the feature missing or corrupt, Learn^{++}.MF suffers an 8-12% drop in performance as opposed to the performance it achieved when few or no features were missing. The same is generally true for Learn^{++}.MFv2.

The percent of instances that can be processed should not vary much between the algorithms, which is consistent with our other analyses. This is because both the algorithms rely on the same Random Subspace Method for the selection of their features. A single usable classifier trained on *nof* = 6/16 is able to process 12% of the instances, whereas the Learn^{++}.MF and Learn^{++}.MFv2 ensemble trained on the same *nof* are able to process 95% of the instances even when 30% of the features are missing.

Table 5.10: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the PEN Dataset

	(nof = 6/16)				*(nof = 7/16)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	89.20 ± 0.00	100	89.60 ± 0.00	100	90.40 ± 0.00	100	92.60 ± 0.00	100
2.50%	88.98 ± 0.26	100	89.07 ± 0.21	100	89.82 ± 0.38	100	92.23 ± 0.20	100
5.00%	89.04 ± 0.22	100	88.49 ± 0.39	100	90.02 ± 0.33	100	91.83 ± 0.34	100
7.50%	88.80 ± 0.47	100	88.15 ± 0.39	100	89.60 ± 0.53	100	91.23 ± 0.81	100
10.00%	88.28 ± 0.32	100	87.49 ± 0.57	100	89.21 ± 0.53	100	91.03 ± 0.49	100
12.50%	88.34 ± 0.40	100	87.10 ± 0.46	100	88.45 ± 0.35	100	90.26 ± 0.39	100
15.00%	87.67 ± 0.71	100	86.70 ± 0.58	100	87.97 ± 0.53	100	89.52 ± 0.45	100
17.50%	86.98 ± 0.34	100	86.21 ± 0.50	100	87.27 ± 0.90	99	88.70 ± 0.83	99
20.00%	85.90 ± 1.05	99	85.78 ± 0.49	99	86.30 ± 0.88	98	87.93 ± 0.89	98
22.50%	85.23 ± 0.74	99	84.42 ± 0.95	99	85.53 ± 0.77	97	87.10 ± 1.01	96
25.00%	84.81 ± 0.69	98	83.87 ± 0.87	98	84.29 ± 1.28	94	86.45 ± 0.51	95
27.50%	82.70 ± 0.56	97	83.31 ± 1.09	97	82.97 ± 1.07	91	84.95 ± 0.77	91
30.00%	81.67 ± 1.17	95	82.33 ± 0.84	95	81.39 ± 1.24	88	84.22 ± 1.26	88
	(nof = 8/16)				*(nof = 9/16)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	92.00 ± 0.00	100	93.00 ± 0.00	100	92.80 ± 0.00	100	94.00 ± 0.00	100
2.50%	91.70 ± 0.16	100	92.66 ± 0.23	100	92.64 ± 0.36	100	93.31 ± 0.30	100
5.00%	91.32 ± 0.21	100	91.98 ± 0.36	100	92.26 ± 0.23	100	92.99 ± 0.51	100
7.50%	90.96 ± 0.58	100	91.66 ± 0.44	100	91.65 ± 0.42	100	92.98 ± 0.46	100
10.00%	90.07 ± 0.50	100	91.53 ± 0.45	100	90.96 ± 0.67	99	91.40 ± 0.59	99
12.50%	89.79 ± 0.81	99	90.93 ± 0.91	99	90.79 ± 0.66	98	90.80 ± 0.59	98
15.00%	88.12 ± 0.53	99	89.58 ± 0.69	98	89.33 ± 0.54	97	89.27 ± 0.94	96
17.50%	87.87 ± 0.70	97	88.71 ± 0.77	97	88.99 ± 0.61	93	89.06 ± 0.81	93
20.00%	85.84 ± 0.66	96	87.47 ± 0.73	95	87.30 ± 0.72	90	87.86 ± 0.83	89
22.50%	84.59 ± 1.19	92	86.89 ± 0.77	92	86.12 ± 0.79	84	86.51 ± 1.17	85
25.00%	83.89 ± 1.00	88	86.17 ± 1.18	87	85.02 ± 0.98	77	84.35 ± 0.90	77
27.50%	83.16 ± 1.57	82	84.67 ± 1.17	83	84.36 ± 0.78	69	85.09 ± 1.22	69
30.00%	80.73 ± 1.11	76	83.92 ± 1.31	76	82.75 ± 1.15	61	83.04 ± 1.55	61

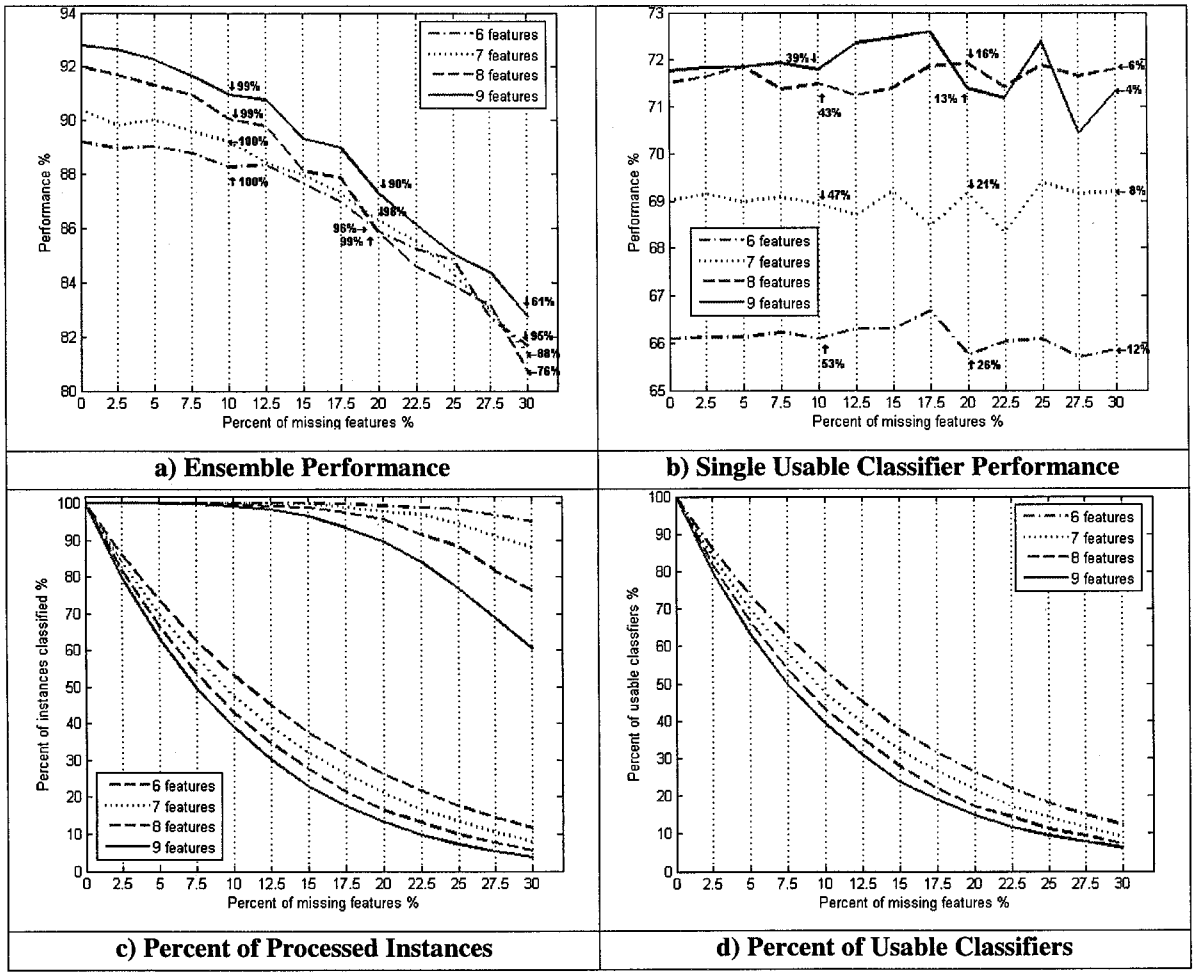

Figure 5.18: Learn^{++}.MF Performance Results on PEN Dataset

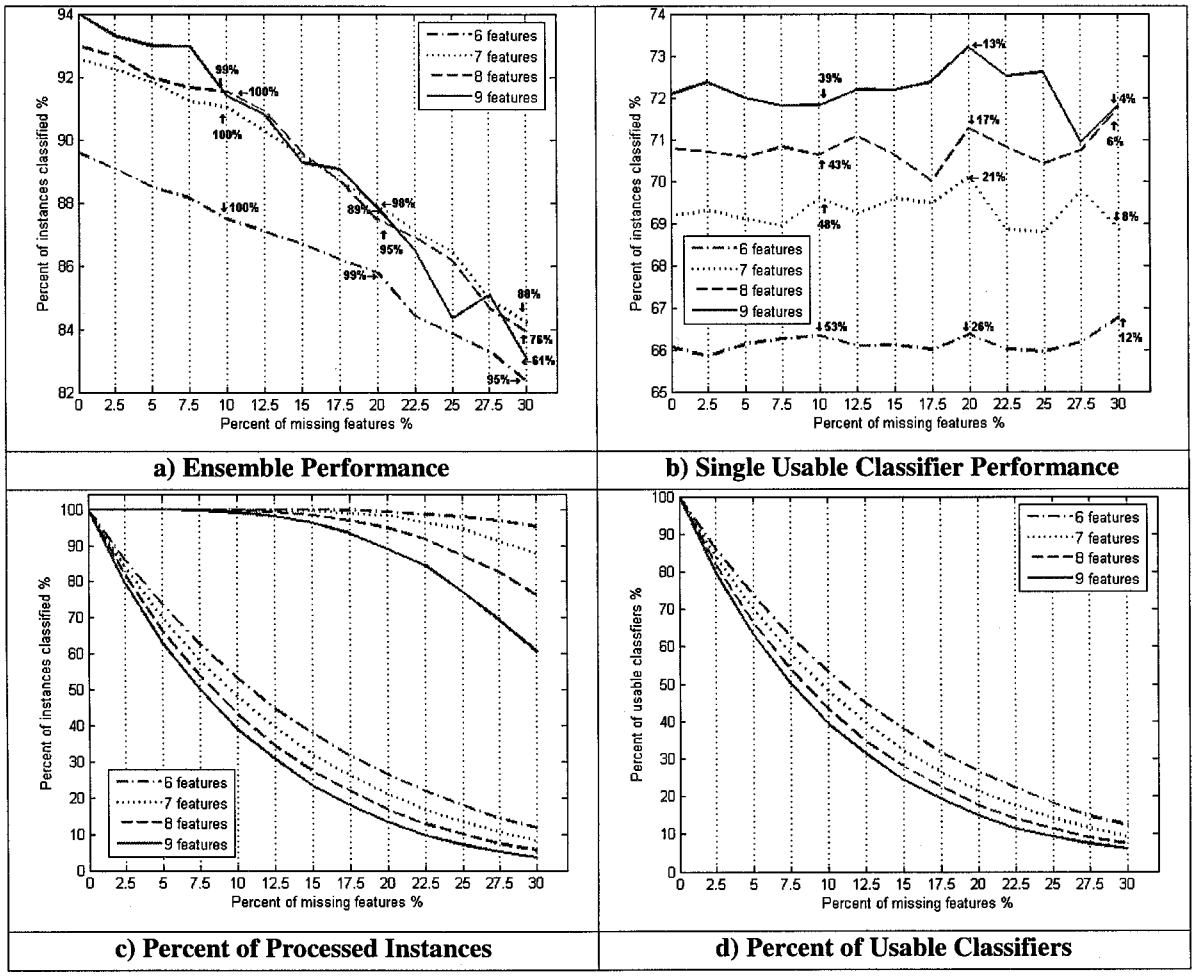

Figure 5.19: Learn^{++}.MFv2 Performance Results on PEN Dataset

5.2.9 *Optical Character Recognition Dataset*

Similar to the previous database, this dataset consisted of the digits 0 through 9. However, they were digitized on an 8x8 grid creating 62 attributes for 10 classes. *T* was set to 1000. Three values of *nof* were considered: 16, 20, and 24 features out of 62, corresponding to 25.81%, 32.26% and 38.71% of the features respectively. Table 5.11 shows the performance of the Learn^{++}.MF and Learn^{++}.MFv2 on this dataset.

The results on this dataset clearly show the benefit of using an ensemble to classify a dataset with missing features. While a single usable classifier trained using *nof* =16 out of 62 features can only classify 19% of the dataset, an ensemble of classifiers trained on the same *nof* can classify 100% of the dataset with even up to 10% of the data missing. As expected, the percent of usable classifiers that can be used to classify the given dataset drops as the percent of missing features increases. While the performances were similar for different number of features, the percent of instances that could be processed by the respective ensembles were not. With 30% of the features missing, an ensemble trained using 24 out of 62 features achieved 88% classification, processing only 9% of the dataset. However, an almost similar performance, 86% was achieved, processing 63% of the dataset using an ensemble trained on 16 out of 62 features. One may see the benefit in using an ensemble trained on a smaller *nof* as in this case. For the three values of *nof,* there was little or no performance drop, processing nearly all the instances in the dataset even with up to 10% of the features missing.

Learn^{++}.MFv2 did not achieve significant increases in performances when the ratio of missing features was relatively low. In fact, the performances were relatively similar even up to 20% of the feature space. However, we noticed some interesting

observations when comparing the performance drop between the algorithms for all values of *nof* as the percent of missing features was increased. With 30% of the feature space was missing, the Learn^{++}.MF achieved a performance of 86% compared to the Learn^{++}.MFv2, which was still able to retain performances at 91% when the *nof* under consideration was 16/62. We see similar trends for all *nofs* when we compare the two algorithms simultaneously.

Table 5.11: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the OCR Dataset

	(nof = 16/62)				*(nof = 20/62)*			
% Missing Features	**% v1 Mean Performance**	**% v1 Process**	**% v2 Mean Performance**	**% v2 Process**	**% v1 Mean Performance**	**% v1 Process**	**% v2 Mean Performance**	**% v2 Process**
0.00%	96.80 ± 0.00	100	96.80 ± 0.00	100	96.80 ± 0.00	100	97.10 ± 0.00	100
2.50%	96.75 ± 0.11	100	96.52 ± 0.13	100	96.98 ± 0.15	100	96.95 ± 0.12	100
5.00%	96.46 ± 0.15	100	96.32 ± 0.17	100	96.84 ± 0.14	100	96.85 ± 0.17	100
7.50%	96.29 ± 0.22	100	96.15 ± 0.26	100	96.84 ± 0.24	100	96.77 ± 0.13	100
10.00%	96.17 ± 0.18	100	96.12 ± 0.17	100	96.22 ± 0.20	100	96.42 ± 0.27	100
12.50%	95.85 ± 0.22	100	95.76 ± 0.30	100	95.49 ± 0.28	99	96.04 ± 0.26	99
15.00%	95.37 ± 0.30	100	95.47 ± 0.29	100	94.87 ± 0.42	97	95.10 ± 0.30	97
17.50%	94.49 ± 0.38	99	94.72 ± 0.35	99	93.38 ± 0.67	93	93.64 ± 0.39	92
20.00%	93.22 ± 0.47	97	94.49 ± 0.37	98	91.83 ± 0.52	83	93.38 ± 0.67	84
22.50%	91.82 ± 0.60	93	93.44 ± 0.49	93	90.69 ± 0.69	70	91.67 ± 0.46	70
25.00%	89.37 ± 0.76	87	92.34 ± 0.32	86	89.45 ± 0.54	55	91.83 ± 0.52	54
27.50%	88.36 ± 0.89	75	93.22 ± 0.47	75	88.29 ± 0.70	40	90.77 ± 0.82	36
30.00%	86.47 ± 0.93	63	91.82 ± 0.61	63	87.95 ± 1.72	26	90.70 ± 0.69	19

	(nof = 24/62)			
% Missing Features	**% v1 Mean Performance**	**% v1 Process**	**% v2 Mean Performance**	**% v2 Process**
0.00%	97.50 ± 0.00	100	97.40 ± 0.00	100
2.50%	97.41 ± 0.06	100	97.38 ± 0.16	100
5.00%	97.19 ± 0.16	100	97.24 ± 0.13	100
7.50%	96.97 ± 0.18	100	96.25 ± 0.25	100
10.00%	96.42 ± 0.33	99	96.28 ± 0.24	99
12.50%	95.12 ± 0.42	95	96.37 ± 0.39	96
15.00%	93.81 ± 0.34	88	96.42 ± 0.33	88
17.50%	92.78 ± 0.72	75	94.88 ± 0.42	75
20.00%	91.58 ± 0.55	58	95.12 ± 0.42	59
22.50%	89.90 ± 0.89	42	93.81 ± 0.34	42
25.00%	89.14 ± 0.96	28	92.78 ± 0.72	27
27.50%	88.51 ± 1.76	17	91.58 ± 0.55	16
30.00%	86.09 ± 1.47	9	89.90 ± 0.89	8

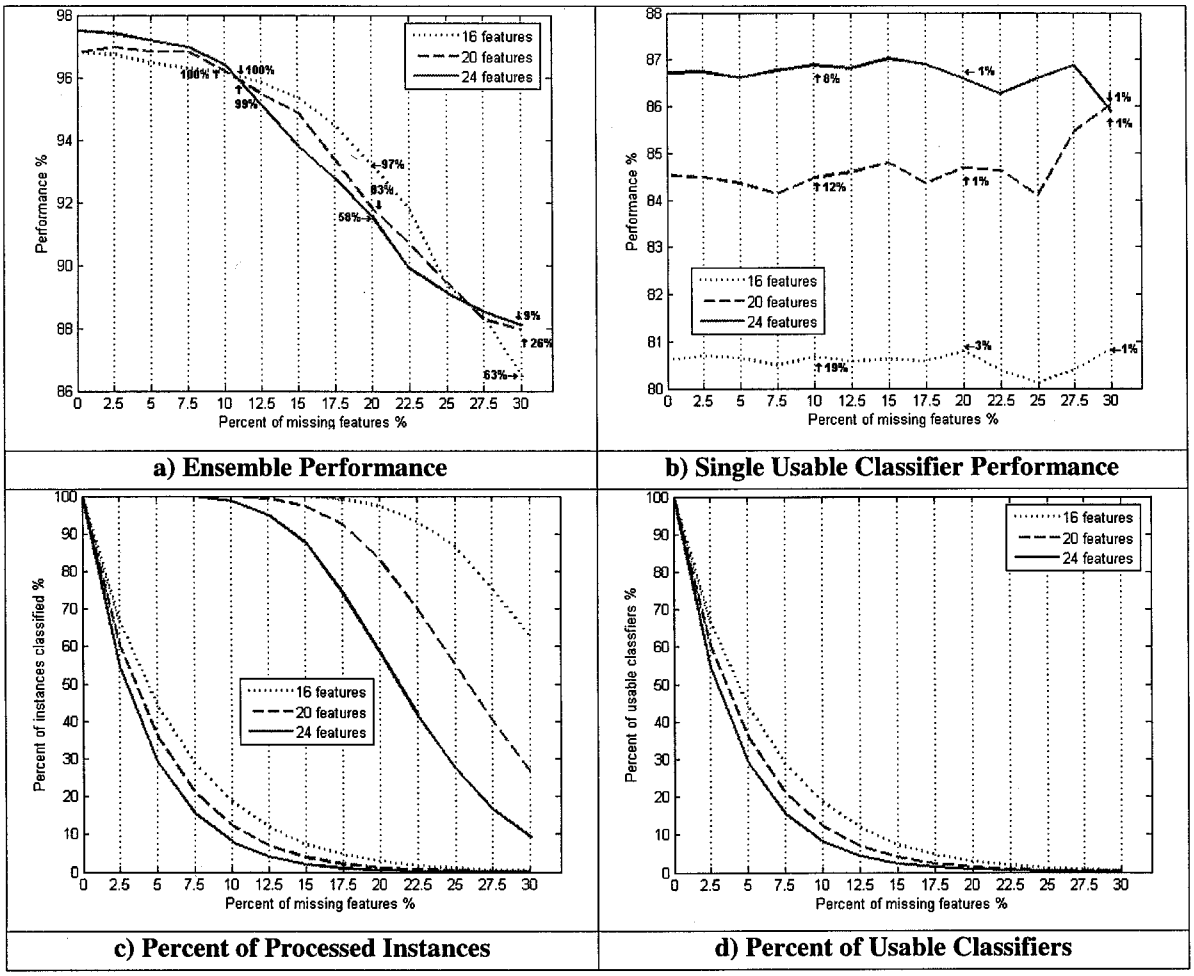

Figure 5.20: Learn^{++}.MF Performance Results on OCR Dataset

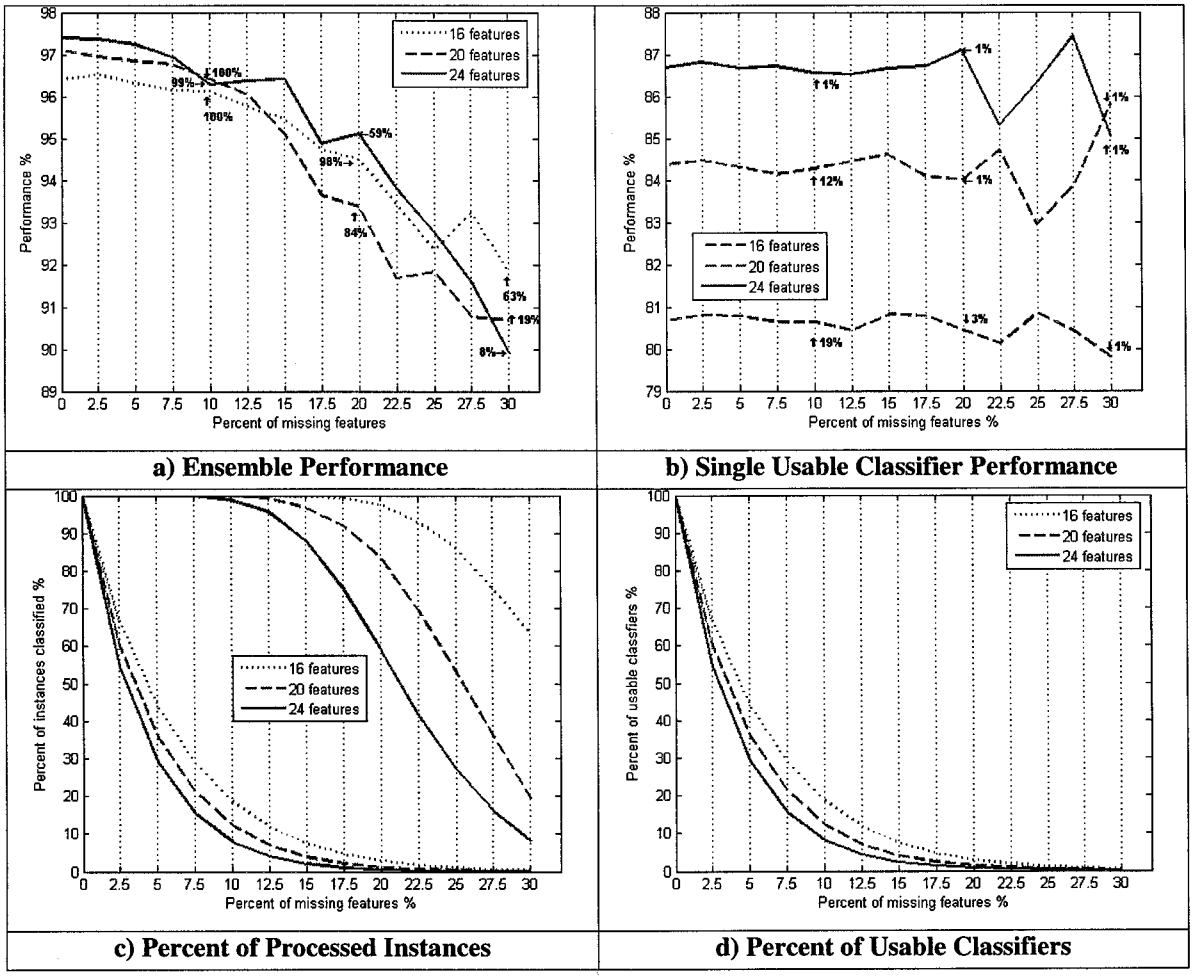

Figure 5.21: Learn^{++}.MFv2 Performance Results on OCR Dataset

5.2.10 E-coli Dataset

This database was created by the Institute of Molecular and Cellular Biology, Osaka, Japan, to predict the localization of protein sites, and it is available from the UCI machine learning repository [71]. This dataset had 7 features, however two features were removed as they were mostly constant values and did not provide discriminatory information to the network. Previous trials using optimized classifiers trained on all the remaining features have allowed us to achieve performances in the 89-90% range, setting the benchmark for this dataset. The algorithms were evaluated by training classifiers using two different values of *nofs*, 2 and 3 out of the 5 available attributes. 100 classifiers were generated for this dataset.

Table 5.12 summarizes the test performance for both algorithms Learn^{++}.MF and Learn^{++}.MFv2. Figure 5.22 and Figure 5.23 provide more insight into the behavior of the algorithms on this dataset. Learn^{++}.MF achieves 73% and 85% classification on *nofs* = 2/5 and 3/5 respectively when no features were missing in the dataset. The proximity of this number to the target 89-90% (obtained when all 5 features where used) range especially for the case when nof=3/5, indicates that this dataset does include redundant features. These figures dropped to 71% and 81% for the respective *nofs* when 30% of the features were missing. A single usable classifier trained on *nofs* = 2/5 and 3/5 maintained 64% and 76% respectively even when 30% of the features were missing.

Learn^{++}.MFv2 was able to achieve 76% and 86% classification when no features were missing. The performance of a single usable classifier trained on *nofs* = 2/5 and 3/5 did not vary from its predecessor. Also, the percent of instances classified by the ensemble did not vary from the former algorithm.

Table 5.12: Learn^{++}.MF and Learn^{++}.MFv2 Performances on the Ecoli Dataset

	(nof = 2/5)				*(nof = 3/5)*			
% Missing Features	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process	% v1 Mean Performance	% v1 Process	% v2 Mean Performance	% v2 Process
0.00%	73.04 ± 0.00	100	75.61 ± 0.00	100	85.22 ± 0.00	100	86.09 ± 0.00	100
2.50%	72.52 ± 0.94	100	75.35 ± 0.88	100	84.87 ± 0.43	100	85.83 ± 0.59	100
5.00%	73.04 ± 1.21	100	75.44 ± 1.05	100	85.01 ± 1.03	100	85.04 ± 0.76	100
7.50%	73.30 ± 1.02	100	74.66 ± 0.68	100	83.72 ± 0.97	99	84.17 ± 1.34	98
10.00%	72.78 ± 1.47	100	74.66 ± 1.15	100	84.71 ± 1.47	97	84.70 ± 0.67	97
12.50%	73.13 ± 1.48	100	74.57 ± 1.75	100	83.86 ± 1.18	97	83.91 ± 1.11	97
15.00%	72.70 ± 1.71	100	74.05 ± 1.60	100	83.99 ± 2.04	96	83.48 ± 1.78	96
17.50%	72.78 ± 1.52	100	74.13 ± 1.21	100	81.85 ± 2.33	93	82.61 ± 1.49	93
20.00%	73.04 ± 1.34	100	74.22 ± 2.01	100	82.36 ± 1.77	89	81.91 ± 1.68	89
22.50%	73.04 ± 1.21	100	73.87 ± 2.31	100	83.04 ± 2.08	88	82.35 ± 1.90	87
25.00%	71.65 ± 1.35	100	73.00 ± 1.88	100	82.48 ± 1.06	81	80.87 ± 1.17	82
27.50%	72.09 ± 2.06	100	73.87 ± 1.81	100	81.64 ± 1.78	78	80.09 ± 2.91	76
30.00%	71.22 ± 1.51	100	73.79 ± 1.67	100	80.65 ± 3.00	72	79.74 ± 2.13	73

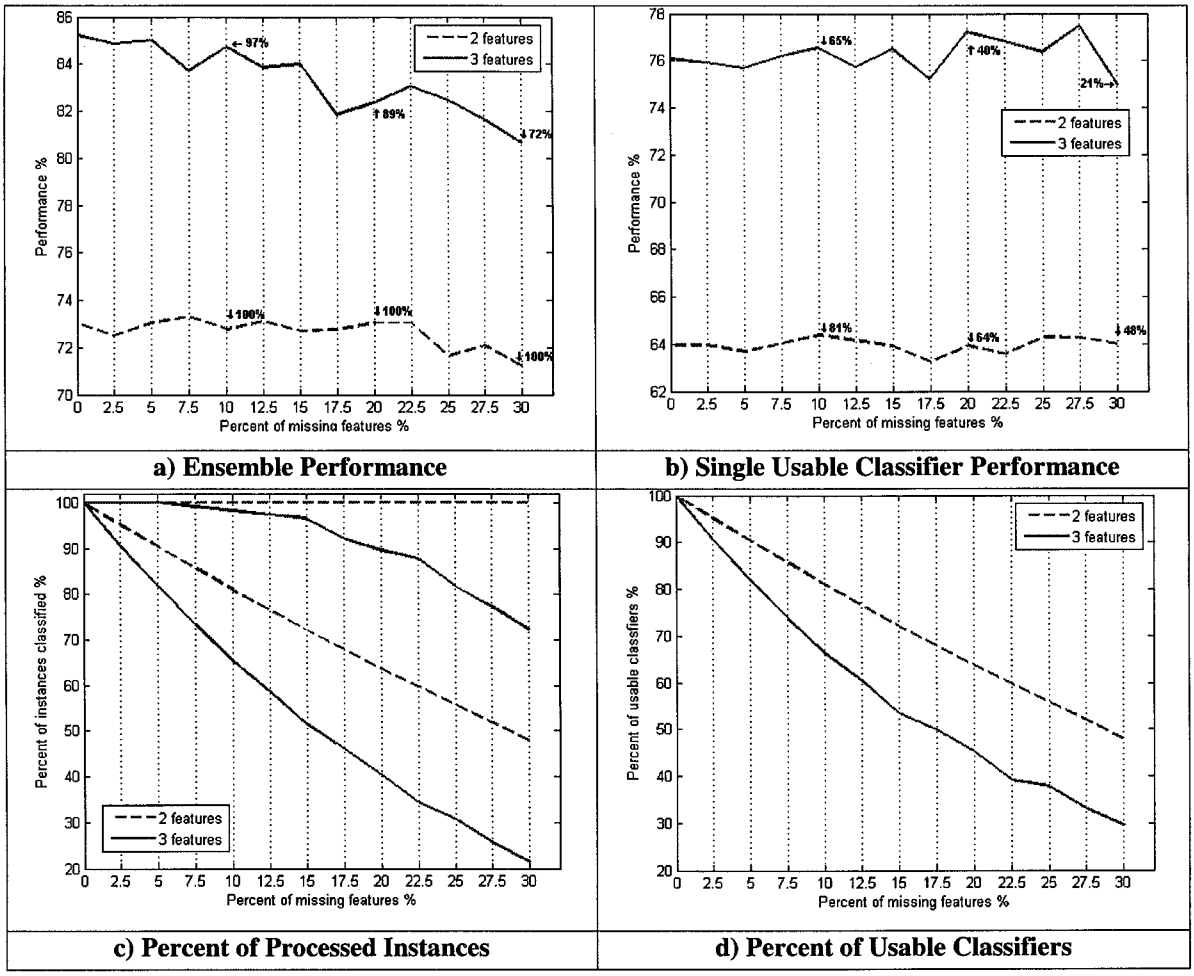

Figure 5.22: Learn^{++}.MF Performance Results on Ecoli Dataset

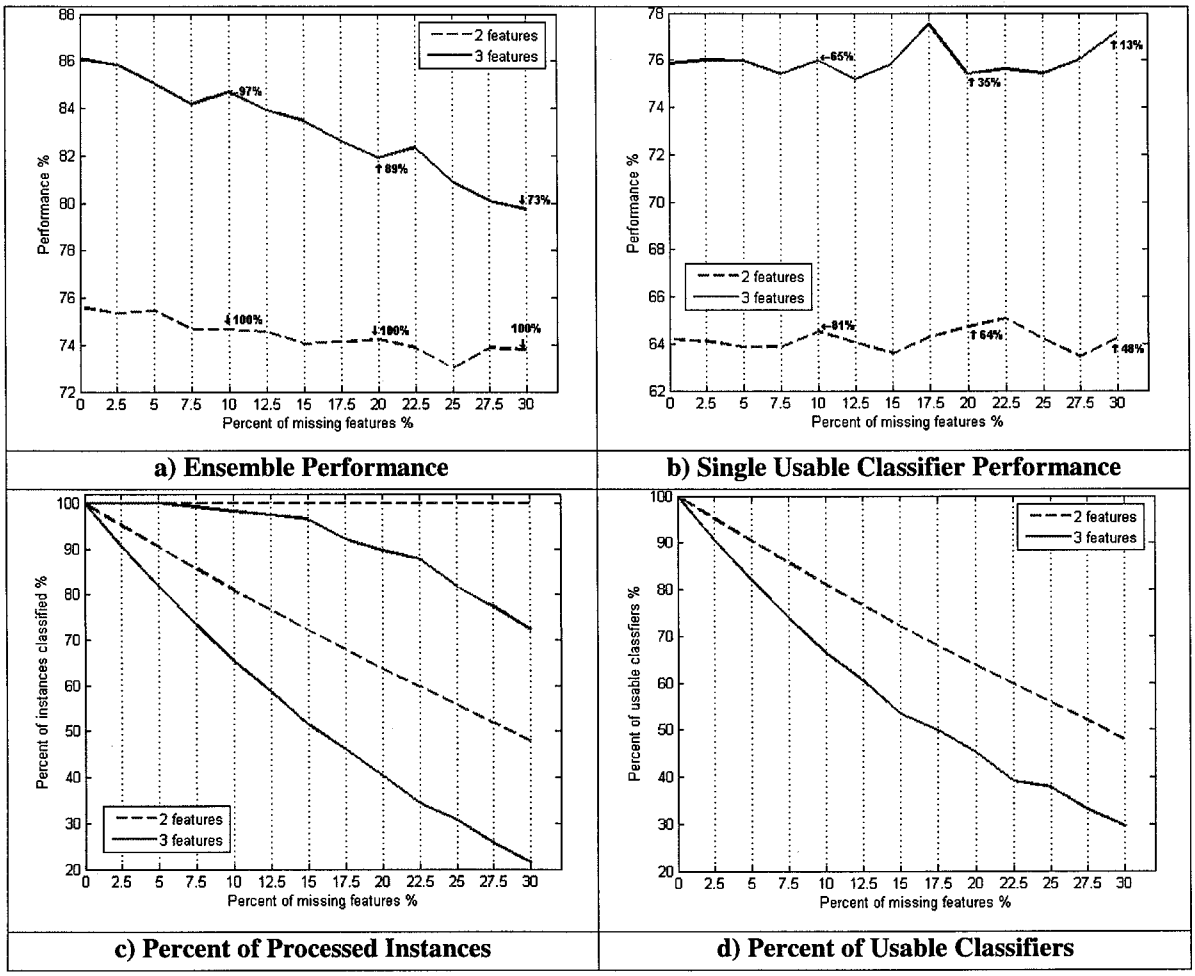

Figure 5.23: Learn^{++}.MFv2 Performance Results on Ecoli Dataset

5.3 Summary of Learn^{++}.MF and Learn^{++}.MFv2

5.3.1 Summary of Learn^{++}.MF

- The algorithm performs remarkably well on classifying data with missing features, with little or no performance drop, with approximately 10-15% of missing features when compared to classifying data with all features intact.
- The choice of the parameter *nof* presents a trade-off: higher classification performance is normally obtained using a larger *nof*, particularly when fewer features are missing. However, the performance drops more rapidly as the percent of missing features increase, compared to using a smaller *nof*.
- Smaller *nof* also results in fewer instances left unprocessed and larger number of usable classifiers, for any given ratio of missing features.
- The algorithm is not able to process all instances for certain values of *nof*.

5.3.2 Summary of Learn^{++}.MFv2

Most of the above arguments for Learn^{++}.MF hold true for Learn^{++}.MFv2 as well. We now point out some of the trends of Learn^{++}.MFv2.

- An ensemble trained on a lower *nof* was able to match some of the performances of ensembles trained with a larger *nof* under Learn^{++}.MF.
- With the exception of a few datasets, any initial performance boost provided by the modifications in Learn^{++}.MFv2 was generally lost when the ratio of missing features increased.

5.4 Evaluation of Learn^{++}.MF and Learn^{++}.MFv2

Two algorithms, Learn^{++}.MF and Learn^{++}.MFv2, have been proposed as possible solutions to the missing feature problem. The algorithms create an ensemble of classifiers, each trained with a random subset of the features, so that any instance with missing features can still be classified using classifiers that did not use those missing features in their training. We have randomly created missing or corrupt data with 0 to 30% of features missing. Thus far, the algorithms have performed remarkably well with negligible or virtually no performance drop for up to 10%-15% of the features missing for most of the datasets that we have. We do observe performance drop as more features become missing. This is especially true when up to 30% of the feature pool is corrupt or missing.

The number of classifiers that need to be generated and cardinality of the feature subsets are important parameters of both algorithms. In general, a relatively large number of classifiers should be generated to ensure that sufficient number of classifiers is available for as many possible combinations of features as possible. The computational burden is not as excessive as it might appear, however, as a typical run on the 34 feature datasets (ION and DERM) needed only a couple of hours to train 1000 classifiers on a P4 machine equipped with 2.0 GHz processor and 512 MB RAM. This is, in part, due to the fact that individual classifiers are relatively weak, obtained by using a small network architecture and high error goal. Hence, each can be trained with relatively little computational burden. In fact, one can argue that – under the assumption of a redundant feature space – random subset selection is quite efficient: An exhaustive run on training with every possible combination of, say, 8 features out of 34 would have normally

required 18,156,204 classifiers, though the algorithms performed remarkably well with only 1000.

Of course, as described earlier, there are algorithms that use far fewer classifiers than Learn^{++}.MF and Learn^{++}.MFv2. For example, combining one-class classifiers trained on single feature at a time can handle any combination of missing features using the fewest possible classifiers (number of features time number of classifiers); however, as shown in the DERMA dataset, the generalization performance suffers significantly due to insufficient distinguishing ability of single features. Conversely, Learn^{++}.MF and Learn^{++}.MFv2 do not claim capability of handling all possible combinations of missing features, but it can typically process – and usually correctly – a substantial large portion of the data, provided that a reasonably sufficient number of classifiers are trained, and, the main assumptions of the algorithm are met.

The second free parameter is the number of features (*nofs*) used to train individual classifiers. In order to obtain a general rule of thumb on proper selection of this parameter, we have analyzed the impact of this parameter on the overall performance as well as on the proportion of instances that can be processed. As described in the earlier section, using a larger number of features for training typically provides better performance when the percent of missing features is less than 10-15%. However, as the percent of missing features increases, the overall performance and the proportion of instances that can be processed by the ensemble drops rapidly. Using fewer features for training, on the other hand, typically provides a more stable performance with a more gradual drop both in performance and ratio of instances that cannot be processed. These results suggest that a near-optimum number of features can be selected if approximate

percent of missing features is known ahead of time. While the specific value depends on the application, the number of features we have used was typically in the 15% to 50% range of the total feature pool.

In recognition of the *no-free-lunch* theorem [72], we acknowledge that any algorithm is effective only to the degree its characteristics match those of the data. In the case of well-established techniques such as Bayesian estimation and expectation maximization, this translates into restrictions on dimensionality, prior knowledge of underlying distributions, and/or availability of sufficiently dense training data. Alternatively, instead of trying to estimate the missing values (and potentially err on this estimation) and introduce bias within the data, Learn^{++}.MF and Learn^{++}.MFv2 try to make the most of the available data.

The restrictions for both algorithms are on the feature sets: first, the algorithms assume that the dataset includes an unknown number of redundant features, and that fewer than the number of available features are in fact adequate to describe the data. The redundancy of the features is not within the feature space and is dispersed across the features. Of course, the number and identities of the redundant features are unknown to us, since they would not have been part of the data otherwise. In other words, the features must not be part of a time-series data. Therefore, signals such as raw electrocardiogram (ECG), cannot be used with this algorithm, however, specific features extracted from the ECG (such as maximum amplitude, area under the QRS complex, rise time and fall time of the QRS complex, etc.) can be used.

It is those applications that meet these two criteria for which the algorithms Learn^{++}.MF and Learn^{++}.MFv2 are expected to be most effective. Fortunately, such

applications are abundant in real world: many practical applications that use a set of different sensors (e.g., temperature, speed, humidity, load, temporal differences between events, etc.) to monitor a physical condition typically meet these conditions. Therefore, the algorithm would be particularly useful, when one or more of the sensors malfunction, or when some of the data become corrupted.

CHAPTER 6 – CONCLUSIONS

This chapter has been divided into three sections. Section 6.1 provides a synopsis of the previous content mentioned in the earlier chapters. Section 6.2 outlines of the accomplishments of the research involved in this thesis, and lastly Section 6.3 provides recommendations and guidelines for future work.

6.1 Synopsis of Thesis

Chapter 1 presented the problem of the missing values in datasets. Chapter 2 described the taxonomy of the general techniques used to solve or counter the missing feature problem. The advantages and disadvantages for these methods were also discussed. Chapter 3 provided an introduction to ensemble based systems detailing some of the reasons for the success of such approaches. This was followed by a brief introduction to two ensemble techniques, AdaBoost.M1 and Learn^{++}. The Random Subspace Method, a core technique in the two algorithms presented in this thesis was also discussed. Chapter 4 formally introduced the Learn^{++}.MF and Learn^{++}.MFv2, two ensemble approach algorithms based on the Random Subspace Method to classify data with missing features. Chapter 5 discussed the results of both the algorithms on multiple real world and benchmark datasets.

6.2 Summary of Accomplishments

This goal of this thesis was to evaluate the feasibility of two similar ensemble-based techniques designed specifically for the missing feature problem. The principal contributions along with the original objectives (mentioned in chapter 1) are revisited below:

1. *To implement an ensemble approach based on the Random Subspace Method for the feasibility of the Missing Feature Problem.* The Learn^{++}.MF selects a subset of the feature space in an autonomous pseudorandom manner and trains a classifier on that subset of features. It then reselects another subset of the feature space and trains another classifier. It does this for a predetermined number of iterations. During validation or testing, it uses only classifiers from its original ensemble trained on features not missing for the particular instance for testing.
2. *To investigate the effects of using such a method for the Missing Feature Problem on multiple real world applications and benchmark datasets.* The algorithm Learn^{++}.MF was evaluated under various conditions by simulating missing features by randomly corrupting the data in each of this datasets. In each of these cases, T, the number of classifiers was kept constant for the specific datasets under investigation. The algorithm was also evaluated using various subsets of the original feature space, mostly in the 15-50% range.

3. *To investigate possible modifications to the former algorithm Learn^{++}.MF to attempt to boost its performance using established techniques such as weighted combination rules.* Learn^{++}.MFv2 was designed specifically with the intention of possible performance boosting. It attempts to take advantage of the local expertise of each of the classifiers with respect to their class specific performance during training. It also introduces a decision boundary to aid its decision making process. The algorithm Learn^{++}.MFv2 was evaluated on the same datasets under the same conditions imposed on the predecessor algorithm.

6.3 Recommendations and Directions for Future Work

There are a number of areas where future work may be performed on the algorithms Learn^{++}.MF and Learn^{++}.MFv2. Inherently, the latter can be viewed as a superset of the former algorithm, so some of the existing shortcomings faced by the original algorithm were faced by the modified version as well.

This includes the notable decrease in the ensemble performance as the ratios of missing features are increased into the dataset. Although the algorithms do manage to maintain a reasonable performance when close to when no features are missing initially for a certain percentage of missing features, the ensemble performance drops as a larger ratio of missing features are introduced into the datasets. This is generally the case for all the datasets that the algorithms were evaluated on. Both of the algorithms are heavily dependent on the number of classifiers to classify any given instance. The success of weak classifiers resides in their ability to introduce sufficient diversity to compensate for their accuracy. Since the number of usable classifiers for any given instance is reduced

each time more features become corrupt in the dataset, their diversity is reduced and along with it their accuracy which explains for the decrease in performance as the ratio of missing features are increased. Hence, it is recommended that the stronger classifiers be trained. However, this should be done with care so as not to introduce *overfitting*.

Future work may include a theoretical analysis of the algorithms to better establish a link among the number of features used, percent missing features, percent data that can be processed, and the number of classifier required to achieve a meaningful performance. Such an analysis may provide further insight to further optimize the algorithm.

REFERENCES

1. V. Tresp, R. Neuneier, S. Ahmad, "Efficient methods for dealing with missing data in supervised learning," G. Tesauro, *et al.* (eds), *Adv. in Neural Inf. Proc. Sys. 7.* MIT Press, 1995.

2. A. Morris, M. Cooke, P, Green, "Some solutions to the missing feature problem in data classification, with application to noise robust ASR," *Proc. Int. Conf. Acoustics, Speech, and Signal Proc.,* vol. 2, pp: 737 - 740, 1993.

3. A.P. Dempster N.M. Laird and D.R. Rubin, "Maximum-likelihood from incomplete data via the EM algorithm (with discussion)," *Jour. of the R. Statist. Soc.*, Series B, pp. 1-38, 1997.

4. M. Jordan, R.Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Comp.*, vol.6, no. 2, pp. 181-214, 1994.

5. G.J. McLachlan and T. Krishnan. The *EM* Algorithm and Extensions. Wiley, New York, 1992.

6. X.L. Meng and S. Pedlow, "EM: a bibliographic review with missing articles," *Proc. Statist. Comput. Sect. Am. Statist. Ass.,* pp. 24-27, 1992.

7. R. Duda, P. Hart, D.Stork, Pattern Classification, 2nd edition, Wiley-Interscience, New York, NY, 2001.

8. M. Skurichina and R Duin, "Combining Feature Subsets in Feature Selection," *LNCS* 3541, pp. 165-175, 2005.

9. R. E. Tarjan, "Depth first search and linear graph algorithms," *SIAM Jour. on Comp.*, 1(2), pp.146-160, 1972.

10. D. Motter and I. Markov, "A Compressed Breadth-First Search for Satisfiability," *Proc. ALENEX 2002*, 2002.

11. W. Morgan, W. Greiff and J. Henderson. Direct Maximization of Average Precision by Hill-Climbing, with a Comparison to a Maximum Entropy Approach. Tech. Report. MITRE Corporation.

12. Morin, R.L., Raeside, D.E.: A reappraisal of distance-weighted k-nearest neighbor classification for pattern recognition with missing data. *IEEE Trans. Syst. Man Cybern*, 11, pp. 241-243, 1981.

13. O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R.B. Altman, "Missing value estimation methods for DNA microarrays," *Bioinformatics*,vol.17, no. 6, pp. 520-525, 2001.

14. S. Oba, M. Sato, I. Takemasa, M. Monden, K. Matsubara, S. Ishii, "A Bayesian missing value estimation method for gene expression profile data," *Bioinformatics*, vol.19, no. 16, pp. 2088-2096, 2003.

15. K.L. Wagstaff, V.G. Laidler, "Making the most of missing values: object clustering with partial data in astronomy," 14th *Astronomical Data Analysis and Systems Conf.*, P. L. Shopbell, M. C. Britton, and R. Ebert, Eds., Vol. XXX, P 2.1.25, 2005.

16. J.M. Brick, G. Kalton, and J.K Kim, "Variance estimation with Hot Deck imputation using a model," Survey Methodology, vol.30, no. 1, June 2004.

17. D. Curran, G. Molenberghs, P.M. Fayers and D. Machin, "Incomplete quality of life data in randomized trials: missing forms," Statistics in Medicine, vol. 17, no. 5, pp. 697-709, 1998.

18. R.J.A. Little and D.B. Rubin. Statistical analysis with missing data. 2edn. ISBN 0-471-18386-5. Wiley Interscience, 2002.

19. R.J.A. Little, "Consistent regression methods for discriminant analysis with incomplete data," *J. Amer. Statist. Assoc.*, vol. 73, pp. 319-322, 1978.

20. P.D. Allison. Missing Data. Sage University Paper Series on Quantitative Applications in the Social Sciences, series 7, no. 136, 2001.

21. S. Zhang, Z. Qin, C.X. Ling and S. Sheng, "Missing is Useful: Missing Values in Cost-Sensitive Decision Trees," *IEEE Trans. on Knowledge and Data Engineering*, vol. 17, no. 12, 2005.

22. C.X. Ling, Q. Yang, J. Wang, and S. Zhang, "Decision Trees with Minimal Costs," *Proc. 21st Int'l Conf. Machine Learning* (ICML04), 2004.

23. P.Juszczak and R.P.W. Duin, "Combining One-Class Classifiers to Classify Missing Data," *LNCS* 3077, pp.92-101, 2004.

24. A.Gupta and M. Lam, "The weight decay backpropagation for generalizations with missing values," *Annals of Operations Research*, vol. 78, no. 1, pp. 165-187, 1998.

25. H. Schoner. Working with Real Datasets. *PhD Thesis*. Berlin Univ. of Technology, Berlin, 2004.

26. H. Schoner, "Working with Real-World Datasets," *PhD Thesis*, Berlin University of Technology, 2004.

27. L.I. Kuncheva, Combining Pattern Classifiers, Methods and Algorithms, New York, NY: Wiley Interscience, 2005.

28. D. Parikh, M.T. Kim, J. Oagaro, S. Mandayam, and R. Polikar, "Ensemble of Classifiers Approach for NDT Data Fusion," *IEEE International Ultrasonics, Ferroelectrics,and Frequency Control Joint 50th Anniversary Conf.,* pp. 1062 – 1065, 2004.

29. A.J.C. Sharkey, "On Combining Artificial Neural Nets," *Connection Science*, Special Issue on Combining Artificial Neural Networks: Ensemble Approaches 8 (3,4) pp. 299-314, 1996.

30. J. Kittler, M. Hatef, R.P. Duin, J. Matas, "On combining classifiers," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20, no.3., pp. 226-239, 1998.

31. L.I. Kuncheva, "A theoretical study on six classifier fusion strategies", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.24, no.2, pp. 281-286, 2002.

32. Y. Freund and R.E. Schapire, "Decision-theoretic generalization of on-line learning and an application to boosting," *J. of Comp. and System Sci.*, vol. 55, no. 1, 119-139, 1997.

33. L. Didaci and G. Giacinto, "Dynamic Classifier Selection by Adaptive k-Nearest-Neighborbood Rule," *MCS 2004, LNCS 3077*, pp. 174-183, 2004.

34. L.I. Kuncheva, "Diversity in multiple classifier systems," *Information Fusion 6*, 3-4, 2005.

35. A. Tsymbal, M. Pechenizkiy, and P. Cunningham, "Ensemble clustering in medical diagnostics," *Proc. of the 17th IEEE Symposium*, pp. 576 – 581, 2004.

36. R.E. Banfield, L.O. Hall, K.W. Bowyer, and W.P. Kegelmeyer, "A New Ensemble Diversity Measure Applied to Thinning Ensembles," *Int. Workshop on MCS*, pp. 306-316, 2003.

37. D. Greene, A. Tsymbal, N. Bolshakova, and P. Cunningham. Ensemble Clustering Medical Diagnostics. *Technical report*, Trinity College, Dublin, 2004.

38. M. Skuruchina and P.W. Duin, "Bagging, Boosting and the Random Subspace Method for Linear Classifiers," *Pattern Analysis and Applications*, no. 5, pp. 121-135, 2002.

39. N. Rooney, D. Patterson, A. Tsymbal and S. Anand, Random Subspacing for regression ensembles. *Technical report*. Trinity College, Dublin, 2004.

40. R.Schapire, "The strength of weak learner," Machine Learning, vol. 5, pp. 197-227, 1990.

41. K. Tieu and P. Viola, "Boosting image retrieval," *IEEE Conf. on Computer Vision and Pattern Recognition 2000*, pp. 148-235, 2000.

42. Y. Freund and R. E. Schapire, "Decision-theoretic generalization of on-line learning and an application to boosting," *J. of Comp. and System Sci.*, vol. 55, no. 1, pp. 119-139, 1997.

43. N.C. Oza, "Boosting with Average Weight Vectors," *MCS: 4th Intl. Workshop*, pp. 15-24, 2003.

44. N.C. Oza and S.J. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," *Knowledge Discovery and Data Mining*, pp. 359-364, 2001.

45. S. Grossberg, "Nonlinear neural networks: principles, mechanisms and architectures," *Neural Networks*, vol.1, no. 1, pp. 17-61, 1988.

46. A. Kulakov and D. Davcev, "Tracking of Unusual Events in Wireless Sensor Networks Based on Artificial Neural-Networks Algorithms," *Int. Conf. on Information Technology*, vol. 2, pp. 534-539, 2005.

47. A. Gangardiwala and R. Polikar, "Dynamically weighted majority voting for incremental learning and comparison of three boosting based approaches," *Proc. of Int. Joint Conference on Neural Networks (IJCNN 2005)*, vol. 5, pp. 1331-1336, Montreal, QB, 2005.

48. R. Polikar, L. Udpa, S. Udpa, V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Transactions on System, Man and Cybernetics (C), Special Issue on Knowledge Management*, vol. 31, no. 4, pp. 497-508, 2001.

49. R. Polikar, S.Krause, L. Burd, "Dynamic weight update in weighted majority voting for Learn^{++}," *Proc. of Int. Joint Conference on Neural Networks (IJCNN 2003)*, vol. 4, pp. 2770-2775, Portland, OR, 20-24 July 2003.

50. M. Lewitt and R. Polikar, "An ensemble approach for data fusion with Learn^{++}," *4th Int. Workshop on Multiple Classifier Systems (MCS 2003)*, Springer LINS vol. 2709 , pp. 176-185, Surrey, England, June 11-13 2003.

51. H. Syed Mohammed and R. Polikar, "Comparison of Ensemble Techniques under Different Combination Rules for Incremental Learning of New Concept Classes," *Technical Report*, Rowan University, 2005.

52. M. Mulbhaier, A. Topalis, and R. Polikar, "Learn^{++}.MT: A New Approach to Incremental Learning," *MCS 2004, LNCS 3077*, pp. 52-61, 2004.

53. D. Opitz, "Feature Selection for Ensembles," *In Proc. of 16th National Conf. on Artificial Intelligence*, AAAI Press, pp. 379-384, 1999.

54. A. Tsymbal, P Cunningham, M. Pechenizkiy and S. Puuronen, "Search strategies for ensemble feature selection in medical diagnostics," *Proc. of the 16th IEEE Symposium*, pp. 124 – 129, 2003.

55. T. K. Ho. "The random subspace method for constructing decision forests," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 20(8) pp. 832--844, 1998.

56. T. K. Ho. "Random decision forests," *In 3rd Intl. Conf. on Document Analysis and Recognition*, pp. 278-282, 1995.

57. M. Skurichina. "Stabilizing weak classifiers," *PhD Thesis*, Delft University of Technology, Delft, The Netherlands, 2001.

58. A. Bertoni, R. Folgieri, and G. Valentini, "Bio-molecular cancer prediction with random subspace ensembles of Support Vector Machines," *NETTAB 2004, Workshop on Models and Metaphors from Biology to Bioinformatics Tools*, 2004.

59. N.V. Chawla and K.W. Bowyer, "Random subspaces and subsampling for 2-D face recognition," *CVPR 2005*, vol. 2, pp. 582 – 589, 2005.

60. X. Wang and X. Tang, "Using Random Subspace to Combine Multiple Features for Face Recognition," *6th IEEE International Conf. on Automatic Face and Gesture Recognition*, p. 284, 2004.

61. E. Pekalska, M. Skurichina and R.P.W. Duin, "Combining Fisher Linear Discriminants for Dissimilarity Representations," *LNCS 1857*, pp. 117-126, 2000.

62. A. Tsymbal, S. Puuronen, and D.W. Patterson. Ensemble Feature Selection with the Simple Bayesian Classification. *Technical report*. Trinity College, Dublin, 2004.

63. J. Zhang and Y. Liu. SVM decision boundary based discriminative subspace induction. *Technical Report*. The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2002.

64. X. Wang and X. Tang, "Random Subspace Based LDA for Face Recognition," *Proc. of CVPR*, 2004.

65. R.E. Banfield, L.O. Hall, K.W. Bowyer, D. Bhadoria, W.P. Kegelmeyer and S. Eschrich, "A Comparison of Ensemble Creation Techniques," *MCS 2004, LNCS 3077*, pp. 223-232, 2004.

66. S. Krause and R. Polikar, "An Ensemble Approach to the Missing Feature Problem," *Int. Joint Conf. on Neural Net.*, vol. 1, pp. 553-558, Portland, OR, 2003.

67. H. Syed Mohammed, N. Steponosky and R. Polikar, "An Ensemble Technique to Handle Missing Data from Sensors," *IEEE Sensors Applications Symposium*, TX, 2006.

68. C. Brodley and T. Lane, "Creating and Exploiting Coverage and Diversity," *Proc. AAAI-96 Workshop on Integrating Multiple Learned Models*, pp. 8-14, 1996.

69. J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas, "On combining classifiers," *IEEE Trans. on PAMI Int.*, vol. 20, no. 3, pp.226-239, 1998.

70. Y. Wu, E.Y. Chang, and W.C. Lai, "Optimal Multimodal Fusion for Multimedia Data Analysis," *ACM Int. Conf. on Mutlimedia*, pp. 564-571, 2004.

71. C.L. Blake, C. Merz, UCI Repository of machine learning databases: http://www.ics.uci.edu/~mlearn/MLRepository.html.

72. D.W. Wolpert and W.G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Trans. on Evol. Comp.*,vol. 1, no. 1, 1997.