# Enhanced Ensemble Learning Technique With A Case Study Of Video Network Traffic Forecasting

Mohamed Alaa El-Dien Mahmoud Hussein Aly

June 2005

**Abstract**

Ensemble learning technique attracted much attention in the past few years. Instead of using a single predictor, this approach utilizes a number of diverse accurate predictors to do the job. Many methods have been proposed to build such accurate diverse ensembles, of which bagging and boosting were the most popular. Another method, called Feature Subset Ensembles (*FSE*), is thoroughly investigated in this work. This technique builds ensembles by assigning each individual predictor in the ensemble a distinct feature subset from the pool of available features. Extensive comparisons are carried out to compare *FSE* with other approaches. In addition, several novel variations to the basic *FSE* are added. The introduced *FSE* variants outperformed the other methods. Experiments were carried out using three different predictor types: least square error (*LSE*), artificial neural networks (*ANN*), and *CART* regression trees.

A case study, the video traffic flow prediction, is investigated. This is a very important application that can improve the efficiency and playback quality of video streaming over the internet. We tried two approaches in order to tackle this important problem. The first is an adaptive algorithm using Sparse Basis Selection technique. The second is the ensemble approach using Artificial Neural Networks (*ANN*). The former technique uses a novel sparse basis selection algorithm, and outperformed most known techniques. The latter is the first attempt to apply the ensemble methodology to address the video traffic prediction problem, and provided very encouraging results.

# Acknowledgements

All thanks and praise go to ALLAH who gave me the strenght that enabled me finish this work.

I am much grateful to my supervisor, Dr. Amir Atiya, who did not spare any effort to help me. He supported me all the way. He was patient and supportive, and provided many invaluable pieces of advice.

Many thanks are due to our department chairman, Dr. Nevin Darwish, for her great help, specially with formal departmental issues.

Finally, this work is dedicated to my Mother, Father, and all my family for their compelete support and prayers.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

This work studies the approach of ensemble learning in a regression context. In this method, the learning model is composed of a group of predictors instead of a single predictor. The individual predictors should both be diverse and accurate to provide enhanced performance. Many methods have been proposed to build such accurate diverse ensembles, of which bagging and boosting are the most popular and successful. These two methods are based on subsampling the training set while giving all the features to each predictors. A new approach, Feature Subset Ensembles (*FSE*), that is based on subsampling the available features, is extensively studied in this work. It is comprehensively compared to bagging, boosting, and sequential selection algorithms using six regression datasets. In addition, several effective novel variants are introduced to the basic *FSE* algorithm.

We further discuss an important problem in current network multimedia applications, namely video network traffic forecasting. It is of crucial importance to enhancing the quality of service of video transmission over the network. Two approaches are used to tackle that problem. The first uses a novel sparse basis selection algorithm to adaptively predict video traffic. The second is using the ensemble approach and treating the problem as a conventional regression task. The two approaches provided encouraging results and outperformed previously used methods.

This work is presented in two parts as follows. Part I deals with predictor ensembles, compares *FSE* to to other techniques, and proposes several variations to it. Chapter 2 gives an overview of ensemble learning approach, discusses the importance of diversity in predictor ensembles, and details bagging and boosting algorithms. This is followed by an overview of feature selection process and discussion of the two forward selection algorithms in chapter 3. Chapter 4 provides

a survey of ensemble methods based on feature selection. Then, chapter 5 details the approach used in comparing *FSE* to other approaches and the different variations introduced to the basic *FSE* method. The complete results together with conclusions are given in chapter 6.

Then, part II dicusses the case study of video traffic prediction. Chapter 7 offers an introduction to the general problem. It discusses its importance and gives an overview of MPEG coding technique. Chapter 8 introduces the general sparse basis selection idea and details the approach used together with the results. Chapter 9 then explains the ensemble approach applied to the video traffic prediction. Finally, the general conclusions are given in chapter 10.

# Part I

# Predictor Ensembles

# Chapter 2

# Ensemble Learning

## 2.1 Introduction

Over the history of machine learning, one single learning model, that we will call ***predictor***, was built to solve a given problem at hand. From a set of candidate predictors, only one is chosen to do the job. However, this single predictor may not be the best one available. Moreover, helping it with other predictors can prove advantageous in improving the prediction accuracy. The technique of using multiple predictors for solving the same problem is known as Ensemble Learning. It has proved its effectiveness over the last few years and still attracts much research.

## 2.2 Predictor Ensembles

Machine learning aims at creating learning models (***predictor*s**) that are trained using a set of examples (training set) and can then act on other unknown examples (test set) based on its training experience. The task can either be a classification or a regression task. In a classification task, each example is associated with a discrete class label, which should be predicted correctly by the predictor. The regression task, on the other hand, associates a real-valued number with each example. There are two branches of learning algorithms: supervised and unsupervised learning. In unsupervised learning, the predictor is given a training set of unlabeled examples. The predictor has to figure out which example belongs to which class and act accordingly. However, In supervised learning, the predictor is given a set of labeled examples and should produce a good representation of this training set.

The normal way of doing things was to use a learning algorithm to produce a single predictor using the training set. This approach has been widely employed in the machine learning community and produced good results. However, it has some shortcomings. First, the problem of choosing the best predictor is not straightforward. Some algorithms produce a single predictor, and some can produce multiple candidate predictors. The best predictor is not necessarily the one having the least training error, as it may overfit the training set. The best one, however, is the one that produces the best accuracy on the unseen data, which is not known at training time. It should give good accuracy on the training set, as well as good generalization ability for unseen patterns. This best predictor is not easily found. Second, when choosing a single predictor and discarding the other candidates, we are throwing away valuable information that can lead to further improvements in the accuracy of the task at hand. This led researchers to think of ways to combine multiple predictors instead of using a single predictor.

Predictor Ensembles have been a hot research topic in the past few years [28, 30]. In this approach, a group of predictors are trained and used instead of just employing the best predictor. The outputs of the individual predictors are combined together, using simple averaging or voting for example, to produce the ensemble output. This technique has been proved, both theoretically and empirically, to outperform the single predictor approach [88, 53]. Using multiple predictors can get around a single predictor overfitting the data and can decrease the variance in its predictions. However, for the ensemble to produce good performance, the component predictors not only need to be accurate, but they also need to be diverse i.e. their generalisation errors should be as least correlated as possible [51, 17]. This is intuitive, because nothing can be gained from using predictors that give identical predictions. Researchers have developed many ways that are capable of producing accurate diverse predictor ensembles [28, 30, 17]. The most popular methods are bagging [14], boosting [43, 44], error-correcting output codes [27], random subspace method [56], cross-validation ensembles [63], and many other techniques. Some of these methods are generic i.e. can be used for classification and/or regression and with any predictor type, and some are tailored specifically for a specific predictor or used only for classification or regression. A brief description of the important techniques will be given below, however this will be preceded by an analysis of ensembles in a regression context that highlights the importance of diversity in the ensemble.

## 2.3 Diversity in Predictor Ensembles

Ensembles of identical predictors are of no use, because averaging or combining their outputs does not change it. For ensembles to be better than a single predictor, the individual predictors should have uncorrelated errors regarding the test data. Krogh & Vedelsby [63] proved that, at a single data point, the square error of the ensemble is guaranteed to be less than or equal to the average square error of the component predictors:

$$(f_{ens} - d)^2 = \sum_i w_i (f_i - d)^2 - \sum_i w_i (f_i - f_{ens})^2, \tag{2.1}$$

where $f_{ens}$is the ensemble output and equals $\sum_i w_i \, f_i$, $f_i$ is the $i^{th}$ predictor output, $w_i$ is the $i^{th}$ predictor weight such that $\sum_i w_i = 1$, and $d$ is the actual value. As can be seen in equation 2.1, the ensemble square error will be less than the individual square errors, only provided that they have diverse errors. The term on the right is called the *Ambiguity Term*, as it signifies the disagreement between the component predictors and the whole ensemble. The proof for equation 2.1 can be found in [63], but a simpler proof is also provided in [17], which is described below.

$$\begin{aligned}
\sum_i w_i (f_i - d)^2 &= \sum_i w_i (f_i - f_{ens} + f_{ens} - d)^2 \\
&= \sum_i w_i [(f_i - f_{ens})^2 + (f_{ens} - d)^2 + 2(f_i - f_{ens})(f_{ens} - d)] \\
&= \sum_i w_i (f_i - f_{ens})^2 + \sum_i w_i (f_{ens} - d)^2 + \\
&\quad 2 \sum_i w_i (f_i - f_{ens})(f_{ens} - d) \qquad (2.2) \\
&= \sum_i w_i (f_i - f_{ens})^2 + (f_{ens} - d)^2 + \\
&\quad 2(f_{ens} - d) \sum_i w_i (f_i - f_{ens}) \qquad (2.3)
\end{aligned}$$

The rightmost term in 2.3 vanishes to $zero$ as the sum of the weights $\sum_i w_i = 1$. This leads to the result in 2.1 above. This result was quite encouraging to the ensemble learning community, and it lead researchers into thinking of different ways to create diverse ensemble predictors. The next two sections describe two of the most spread ensemble creation algorithms, namely *bagging* and *boosting*. They both rely on the idea of sub-sampling the training set by creating a different training set for each predictor, but they differ in the way this set is sampled.

## 2.4 Bagging

Bagging (an acronym for **B**ootstrap **agg**regat**ing**) is one way of creating diverse predictor ensemble , and it can accomplish this through creating different training sets for each predictor. Bagging is based on the idea of bootstrap sampling [14]. A different training set is presented to each predictor, by sampling $m$ samples with replacement from the original training set of $m$ patterns. Each sample is called a *bootstrap sample* [38, 53], and that is where the algorithm got its name from. By sampling with replacement, some patterns will be repeated more than once, others will be absent, however, on the average, 63.2% of the original training set will be represented. After training all the individual predictors, the final ensemble output is the simple average of the component predictors outputs. Algorithm 1 describes the bagging procedure.

**Algorithm 1** BAGGING

**Inputs:**

a set $S$ of $m$ training patterns: $S = \{(\mathbf{x_i}, y_i),\ i = 1, \ldots, m\}$, where $\mathbf{x}_i$ is the $i^{th}$vector of features and $y_i$ is the $i^{th}$target value,
LEARN: a learning algorithm ,
$K$: the number of predictors to return,

**Outputs:**

an ensemble $E$ of $K$ predictors: $E = \{C_k,\ k = 1, \ldots, K\}$

**Procedure:**

1: **for** $k = 1 : K$ **do**
2: create $T_k$ {a bootstrap training set of size $m$}
3: $C_k$ :=LEARN$(T_k)$ {train the predictor}
4: **end for**

Bagging has been shown to perform better than single predictors [14, 86, 67, 99, 7, 29, 79, 88, 50]. It was also shown that it can perform better than other ensemble learning techniques, specially in noisy environments. The only requirement for bagging to produce good diverse ensembles is the instability of the training algorithm [14]. As long as the bootstrap sampling from the original training causes the training algorithm to produce nonidentical predictors, then bagging is guaranteed, on the average, to produce better results.

The main advantage of bagging is its simplicity in application and its parallelizability. The individual predictors are independent of each other, and so they

can be created simultaneously. In addition, it is applicable to both classification and regression, and does not need any change in the base learning algorithm. In the case of classification, simple majority voting can be used, and in case of regression a simple average of the outputs is employed.

## 2.5 Boosting

Boosting [43, 44], like bagging, relies on the concept of under-sampling the original training set. However, unlike bagging, the sampling process is adaptive and depends on the performance of previously created predictors. In the bagging process, all the patterns in the original set are equally weighted i.e. they have the same chance of being picked up in any predictor's bootstrap sample. On the other hand, in boosting, the weight of each pattern changes dynamically as the algorithm runs, according to the performance of past predictors. Boosting tends to give larger weights on patterns that are misclassified by, or have high prediction error from, previous predictors. Those *hard* patterns have a higher chance of being picked up by future predictors. Therefore, later predictors focus on these hard patterns in an attempt to learn them more accurately. Algorithm 2 describes a variant of boosting algorithm used for regression, called AdaBoost.R, detailed in [36, 88]. AdaBoost stands for **Ada**ptive **Boost**ing, where it adaptively adjusts the distribution of the original training patterns. It starts by setting all the patterns with equal probability of being chosen in the bootstrap sample. Then, for each predictor, it generates a different training set by sampling, with replacement, from the original set, according to the weight distribution of the patterns. For each created predictor, the output of each pattern is computed, and a relative loss function is computed. The weight of each pattern is then adapted according to its difficulty i.e. the more incorrect the predictor was, the larger the weight it takes. The weights are then renormalized to remain a valid distribution.

Boosting has been shown to outperform single predictors and most other ensemble creation methods [43, 44, 37, 86, 67, 99, 7, 29, 79, 88, 50]. It can generate, on average, very accurate ensembles using many learning algorithms (decision trees, regression trees, neural networks, ... etc) for both classification and regression. However, it can suffer sometimes from overfitting, resulting in bad ensembles, even worse than a single one, specially in noisy datasets. Moreover, the individual predictors are dependent on one another, and therefore they have to be trained in sequence, and this prevents the algorithm from being parallelized.

**Algorithm 2** ADABOOST.R

**Inputs:**

a set $S$ of $m$ training patterns: $S = \{(\mathbf{x_i}, y_i),\ i = 1, \ldots, m\}$, where $\mathbf{x}_i$ is the $i^{th}$vector of features and $y_i$is the $i^{th}$target value,

LEARN: a learning algorithm ,

$K$: the number of predictors to return

**Outputs:**

an ensemble $E$ of $K$ predictors: $E = \{C_k,\ k = 1, \ldots, K\}$

**Initialize:**

$w_i^{(1)} := 1/\sum_{i=1}^{m} w_i$ {initialize weights of training set}

**Procedure:**

**for** $k := 1, \ldots, K$ **do** {loop on number of predictors}

construct a training set $T_k$ using the distribution$w^{(k)}$

$C_k :=$LEARN$(T_k)$ {train predictor $C_k$ using bootstrap sample $T_k$}

$\hat{y}_i^{(k)} := C_k(\mathbf{x}_i)$ for all $i,\ i = 1, \ldots, m$

$L_i := \frac{|\hat{y_i}^{(k)} - y_i|}{D}$ for all $i$, where $D := \max_{i=1\ldots m} |\hat{y_i}^{(k)} - y_i|$ {loss for each pattern}

$\bar{L} := \sum_{i=1}^{m} L_i w_i^{(k)}$ {average loss function}

$\beta_k := \frac{\bar{L}}{1-\bar{L}}$ {confidence measure}

$w_i^{(k+1)} := w_i^{(k)} \beta_k^{(1-L_i)}$

$w_i^{(k+1)} := \frac{w_i^{(k+1)}}{\sum_i w_i^{(k+1)}}$ {normalize weight so that it remains a distribution}

**end for**

## 2.6 Other Ensemble Creating Techniques

Many more ensemble creating algorithms have been investigated by researchers, however the most two important are bagging and boosting. Bagging and boosting try to create diverse ensembles by manipulating the distribution of the training set presented to each individual predictor. Other techniques try to achieve the same objective by manipulating the output class labels [27], the input features [56], or by using algorithm-specific property that injects randomness into the different predictors [82, 99, 29]. As indicated, some of these techniques are applied to specific algorithms and can not be used by any general learning algorithm, some are used only for classification and can not be used for regression, and some are general to be used with any algorithm or application.

## 2.7 Summary

Ensemble learning employs a committee of predictors for solving the problem at hand instead of just using one predictor. A group of predictors was proved to be better than single predictors for many practical problems. For the ensemble to be accurate, the individual predictors need to be both accurate and different. Many ways are used to create accurate diverse ensembles, the two major of which are bagging and boosting.

# Chapter 3

# Feature Subset Selection

## 3.1 Introduction

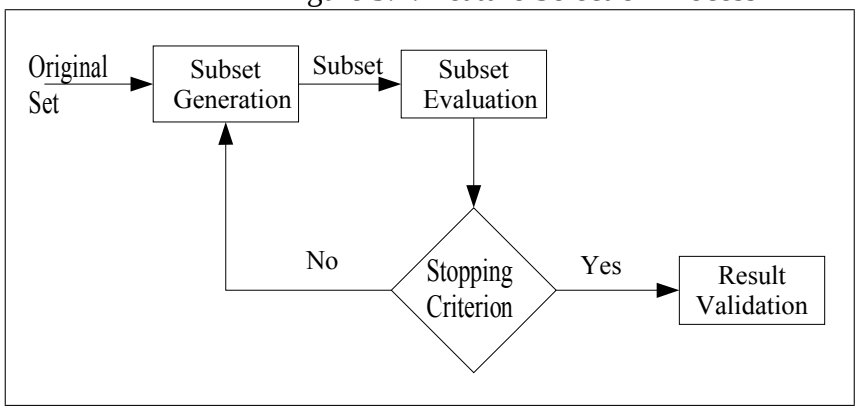
Feature selection is a very important part of the preprocessing phase in machine learning [59, 28, 61] and statistical pattern recognition [102, 58, 53, 66]. In many real world situations, we are faced with problems having hundreds or thousands of features, some of which are irrelevant to the problem at hand. Feeding learning algorithms with all the features can result in a deteriorating performance, as the algorithm can get stuck trying to figure out which features are useful and which are not. Therefore, feature selection is employed as a preliminary step, to selected a subset of the input features, that contains potentially more useful features. In addition, feature selection tends to reduce the dimensionality of the feature space, avoiding the well-known dimensionality curse problem [53].

Feature selection is a non-trivial procedure, as the search space grows exponentially with the number of features e.g. for $n$ features there are $2^n$different subsets that should be examined and evaluated. Many different techniques have been proposed to solve this difficult problem. An overview of the feature selection process is given below, followed by a description of sequential feature selection techniques that are used in this work. They are among the most robust and stable feature selection algorithms.

## 3.2 Feature Selection Process

Feature selection consists of the four major steps [66] in figure 3.1. Subset generation is a search procedure through the feature space that produces candidate

Figure 3.1: Feature Selection Process




feature subsets, based on a certain search strategy. These feature subsets are then evaluated, and if the stopping criterion is not yet reached, other potential subsets are produced. When the stopping criterion is satisfied, the final selection is validated.

### 3.2.1 Subset Generation

Subset generation is a process of heuristic search, producing at each state a candidate solution i.e. feature subset. This search is decided by two factors: the starting point and the search technique. The starting point may be an empty set, to which features are then added, the so called *forward search*. Alternatively, it may start with the full set of features, and bad features are then removed (*backward search*), or with an initial subset to which features are added/removed (*forward-backward*). In addition, there are several search techniques, including complete, sequential, and random search. In complete search, the optimal solution is guaranteed to be found. These include exhaustive search, branch and bound search [74], and beam search [32]. Sequential search, however, follow the greedy hill-climbing approach, and thus are not guaranteed to pick the optimal feature subset. The sequential steps may be taken by adding/removing one feature at a time (sequential forward/backward selection [58]), by adding/removing $l$ in one step and removing/adding $r$ in the other step, $l > r$ (plus-$l$ take away $r$ [91]), or by adding/removing variable number of features, according to the performance of the currently selected feature subset (sequential floating search [84]). The random search, on the other hand, adds/removes features randomly. The most obvious ex-

amples for random search are Genetic Algorithm approaches for feature selection [89, 47, 76].

### 3.2.2 Subset Evaluation

After generating candidate feature subsets, they have to be evaluated to select the best solution. There are two approaches for feature subset evaluation: the filter and the wrapper models [61]. In the filter model, each feature subset is evaluated based on the intrinsic data between the features, independently of the learning algorithm. The wrapper model, on the other hand, employs the learning algorithm as a black box in the evaluation process. The feature subset is evaluated based on its performance, using the learning algorithm, on a validation set, or using $k$-fold cross-validation [59, 60] on the training set. In $k$-fold cross-validation, the training set is partitioned into $k$ equal subsets, and the learning algorithm is evaluated $k$ independent times. In each time, one of the $k$ subsets is used for testing, while using the other $k-1$ subsets for training. The final evaluation is an average of the $k$ runs. The wrapper model is more computationally expensive than the feature model, however it usually gives better results.

### 3.2.3 Stopping Criteria

The stopping criteria are checked after each candidate subset is generated and evaluated, and the search procedure stops when they are satisfied. The criteria may include: the end of the search process, some limit on the feature subset size, the number of iterations is reached, or deterioration of performance on the addition/deletion of any further feature to the subset.

### 3.2.4 Result Validation

The best selected feature subset from the above steps has to be validated. The validation can use prior knowledge about the problem, in case it is available. Otherwise, the performance of the selected set can be evaluated using the present training set, or a separate validation set, or again using $k$-fold cross-validation.

## 3.3 Sequential Feature Selection Algorithms

As described above, many techniques have been used for efficient feature selection. Sequential feature selection, though considered sub-optimal, is among the most robust and widespread feature selection algorithms [59]. Two versions of sequential feature selection will be described here: Sequential Forward Selection (*SFS*) and Sequential Floating Forward Selection (*SFFS*). The subset evaluation follows the wrapper approach, in which $k$-fold cross-validation is used to estimate the feature subset performance. The training set is divided up into $k$ equal parts. The training operation is performed $k$ times, where each part is used once as a test set while training using the other $k-1$ parts. The final error estimation is the average of the $k$ error estimates computed [60]. The stopping criteria employed is the deterioration of the subset performance i.e. the algorithm stops when the addition of new features results in a reduced performance.

### 3.3.1 Sequential Forward Selection

*SFS* is considered among the simplest approaches for feature selection. It starts with an empty feature set, and adds one feature at a time. The feature added is the one that optimizes the objective function value of the currently selected subset including the newly added feature. Addition of new features continues until no further improvement in performance can be achieved. Performance is measured using 10-fold cross-validation on the training set. Algorithm 3 outlines the *SFS*.

### 3.3.2 Sequential Floating Forward Selection

Pudil et al. [84] introduced a modification on the *SFS* algorithm, called the floating search methods. The difference added to the *SFS* is a backward elimination step after the addition of each new feature. Here, the forward addition is unconditional i.e. the best new feature is added to the current subset regardless of the new subset performance. However, the backward elimination step continues to remove features from the subset as long as this improves the performance. It was shown that *SFFS* performs very well, compared to other sequential algorithms [58]. Algorithm 4 outlines the *SFFS*.

**Algorithm 3** Sequential Forward Selection

**Input:**

$S = \{(\mathbf{x}_i, y_i), i = 1, \ldots, m\}$ {training set of $m$ patterns and $k$ features}

$A$ {a learning algorithm}

**Output:**

$J_{best} = \{j, j \in \{1, \ldots, k\}\}$ {the best feature subset}

**Procedure:**

$J_{best} := \phi$ {initialize to empty set}

$\eta_{best} := eval(J_{best}, S, A)$ {evaluate the subset using 10-fold cross-validation using the algorithm $A$ and the training set $S$}

**while** (true) **do**

  **for** $k = 1, \ldots, K$ **do** {for all the features}

    $J := J_{best} + k$ {add the feature to the subset}

    $\eta_k := eval(J, S, A)$ {evaluate its performance}

  **end for**

  $(\eta, k) := \max_k \eta_k$ {get the feature that achieves maximum performance}

  **if** $\eta > \eta_{best}$ **then** {if the best feature improves the performance}

    $J_{best} := J_{best} + k$ {add it to the subset}

    $\eta_{best} = \eta$ {update $\eta_{best}$}

  **else**

    return $J_{best}$ {exit and return the best subset feature}

  **end if**

**end while**

**Algorithm 4** Sequential Floating Forward Selection

**Input:**

$S = \{(\mathbf{x}_i, y_i), i = 1, \ldots, m\}$ {training set of $m$ patterns and $k$ features}

$A$ {a learning algorithm}

**Output:**

$J_{best} = \{j,\ j \in \{1, \ldots, k\}\}$ {the best feature subset}

**Procedure:**

$J_{best} := \phi$ {initialize to empty set}

$\eta_{bestf} := eval(J_{best}, S, A)$ {evaluate the subset using 10-fold cross-validation using the algorithm $A$ and the training set $S$}

**while** (true) **do**

{Step 1: unconditional forward selection step}

**for** $k = 1, \ldots, K$ **do** {for all the features}

$J := J_{best} + k$ {add the feature to the subset}

$\eta_k := eval(J, S, A)$ {evaluate its performance}

**end for**

$(\eta_f, k_{best}) := \max_k \eta_k$ {get the feature that achieves maximum performance}

$J_{best} := J_{best} + k_{best}$ {add the best feature to the subset}

$\eta_{bestb} = \eta_f$ {best $\eta$ in the backward step that follows}

$\eta_{bestf} = \eta_f$ {save the current performance}

**while** $\eta_{bestb} \geq \eta_f$ **do**

{Step 2: the conditional backward elimination step}

**for** $k = 1, \ldots, length(J_{best})$ **do**

$J := J_{best} - k$ {remove feature from the subset}

$\eta_k := eval(J, S, A)$ {evaluate the new subset}

**end for**

$(\eta, k_{best}) := \max_k \eta_k$ {get the feature that achieves maximum performance}

**if** $\eta > \eta_{bestb}$ **then**

$J_{best} := J_{best} - k_{best}$ {remove the worst feature}

$\eta_{bestb} = \eta$ {update $\eta_{bestb}$}

**end if**

**end while**

{check if the last step deteriorated the performance}

**if** $\eta_{bestb} < \eta_{bestf}$ **then**

undo the last addition

return $J_{best}$

**end if**

**end while**

## 3.4 Summary

Feature selection is considered an important preprocessing step in machine learning problems. It tries to focus the attention of the learning algorithm on a small subset of features of potential importance to the problem at hand. Numerous approaches have been used to get this small useful feature subset, of which sequential feature selection techniques are among the most stable.

# Chapter 4

# Literature Survey

## 4.1 Introduction

Ensemble learning is a powerful tool in the field of machine learning. For a predictor ensemble to produce better performance, the constituent predictors have to be accurate and diverse. Many directions have been investigated in order to achieve diverse accurate ensembles. The approach focused on here is the manipulation of input features to the individual predictors. The idea of partitioning the input features among individual predictors arises from the idea of feature selection. In feature selection problems, there exists a large number of features available that can not all be used in the learning process. Some way of choosing the most representative subset of these features must be employed. The disadvantage of feature subset selection is that some features that may seem less important, and are thus discarded, may bear valuable information.

This is where Feature Subset Ensemble ($FSE$) comes into play. It simply partitions the input features among the individual predictors in the ensemble. Hence, no information is discarded. It serves at uitilizing all the available information in the training set, and at the same time not overloading a single predictor with all the features, as this may lead to poor learning.

Some attempts have been made to find a systematic way of distributing the input features among the different predictors. These techniques used random selection [55, 8, 56, 18], traditional feature selection algorithms [4, 49], genetic algorithm approach [78, 48, 77], manual selection of features [23], grouping based on the correlation properties between features [65, 93, 80], and algorithm-specific feature manipulation [99, 29, 16]. Most of these methods achieved encouraging

results and proved useful in the sample datasets they were applied to. These directions will be discussed in detail below.

## 4.2 Random Feature Subset Selection Approach

Ho [55, 56] proposed a technique called *Random Subspace Method (RSM)*. In this technique, a subset of features was randomly selected for each predictor. She used C4.5 decision trees [85] as the base predictor. The number of features selected for each predictor was half the total number of features. Experiments were undertaken to compare the RSM to bagging, boosting and single tree predictors employing all features. Four publicly available datasets from Project StatLog [12] were used. In each ensemble technique, a decision forest was grown up to $100$ decision trees. RSM showed better performance than bagging, boosting, and single tree predictor. In addition, further experiments were conducted to assess the effect of the feature subset size on the ensemble performance. She pointed out that using half the number of features produced acceptable results. The technique can be used with any training algorithm other than decision trees, and can be applied to either regression or classification. However, the feature subset size should not be limited to half the number of features, as this depends on the correlation and redundancy between features.

Bay [8] introduced a similar idea, called *Multiple Feature Subsets (MFS)*. In this method, he applied random feature selection to nearest neighbor (NN) predictors. Each NN predictor was trained using a random subset of features. Bay noticed that other ensemble learning techniques, e.g. bagging or boosting, that depend on subsampling the training set, failed to improve NN ensembles. Therefore, he worked on another method that manipulated input features instead of input patterns. Bay used two sampling functions: sampling with replacement and sampling without replacement. In sampling with replacement, a given feature can be replicated within the same predictor. In sampling without replacement, however, a given feature can not be assigned more than once to the same predictor. Each of the NN predictors used the same number of randomly selected features. The algorithm has two parameters, the feature subset size, and the ensemble size. The latter was fixed to 100 predictors, and the former is chosen using 10-fold cross-validation on the training set using 10 evenly spaced values. Bay developed two versions of his approach: MFS1 (using sampling with replacement) and MFS2 (using sampling without replacement). The output of the predictors were combined using simple voting. These two versions were compared to four other

algorithms: nearest neighbor (NN), $k$ nearest neighbor (kNN), nearest neighbor with forward (FSS), and backward (BSS) sequential selection [102]. The value of $k$ in kNN was computed using 10-fold cross-validation on the training data. The feature selection used cross-validation for accuracy estimation, which is known as the wrapper approach [59, 61]. Bay used 25 publicly available classification datasets from UCI repository [10] to evaluate the compared algorithms. Bay took the average of 30 independent runs for each algorithm, in each one $\frac{2}{3}$ of the patterns were selected as the training set and the remaining used as the test set. Bay reported that the two versions of his approach, MFS1 and MFS2, performed better than the other algorithms in all but two datasets. Bay's approach was applied to classification only, nevertheless it can be applied to both classification and regression. In fact, it is the same idea of Random Subspace Method, but applied to a different predictor type.

Bryll et al. [18] also used a similar approach, which they called *Attribute Bagging (AB)*. In this approach, each predictor is trained on a subset of randomly selected features (without replacement). They proposed a framework, in which the size of the feature subset is determined first, and this parameter is problem dependent. Then, various subsets of that size are evaluated using the wrapper method [61], and only the best of these subsets are used for voting. Bryll et al. evaluated their method on a hand-pose recognition dataset [97] using OC1 decision tree algorithm [72]. The dataset consists of 29 attributes, 20 classes, 934 training patterns, and 908 testing patterns. They tested different subset sizes and found that using $\frac{1}{3}$ of the attributes with an ensemble of 25 predictors gave the best results. They tried both unweighted and weighted voting of the individual predictors, where the weights were proportional to their accuracy on the training set. The results reported were the average of 30 independent runs. Bryll et al. compared their approach to single OC1 trees, bagged OC1 ensembles, ID3, RIEVL and others. Their algorithm showed better results than all of these algorithms. AB is a generic approach that can be used with any learning algorithm, and for both classification and regression. However, it is not flexible enough, as its parameters, the attribute subset size and the ensemble size, are hand selected and are not computed automatically from the training set.

The methods discussed above are nearly similar in that they assign features randomly to each individual predictor. They differ in the way their parameters (subset and ensemble sizes) are estimated. In addition, they were tested only on classification problems not regression tasks.

## 4.3 Traditional Feature Subset Selection Approach

Alkoot and Kittler [4] proposed an approach that uses traditional feature selection algorithms in order to maximize the overall ensemble performance. They proposed three different variations for building the ensemble: the parallel system, the serial system, and the optimized conventional system. In the parallel system, each *expert* (the term they used for a *predictor*) is allowed, in turn, to take one feature such that the overall ensemble performance is optimised on a validation set. In the serial system, in contrast, the first expert is allowed to take all the features that achieve the maximum ensemble accuracy on the validation set. If some features remain, a second expert is used, and so on. The optimized conventional system builds each expert independently, and then features are added/deleted from the ensemble as long as the ensemble performance is increased. Three different predictor algorithms were used: gaussian, $k$ Nearest Neighbor ($k$NN), and Nearest Neighbor (NN). Two classification datasets were employed: Breast Cancer Wisconsin [10] from UCI repository, and Seismic from the University of Surrey. These approaches were compared only to single predictor, and no comparison to other ensemble learning techniques was carried out. Alkoot and Kittler reported that their algorithm performed, on the average, better than single predictor approach.

Günter and Bunke [49] proposed an ensemble creation technique based on feature selection algorithms. They tested their method in the context of handwritten word recognition, using Hidden Markov Model (*HMM*) recognizer [69] as the base predictor. In their approach, each predictor is given a well performing set of features using any existing feature selection algorithm. The only modification that must be made to the feature selection algorithm is to prevent identical feature subsets from being returned. Therefore, the feature selection algorithm tries to get the best unrepeated feature subset with respect to a given objective, and given a list of previously generated feature subsets. Günter and Bunke used two well known algorithms: floating sequential forward and backward search algorithms [84]. Each predictor uses one of the two floating search algorithms to get a unique feature subset. Two objective functions were experimented: the recognition performance on the validation set, and another one that combines a measure of the ensemble diversity with the above accuracy. Günter and Bunke compared their approach to floating forward search, floating backward search, and random subspace method [56] using $50\%$ of the features. The ensemble size employed was 10 predictors. They reported that their approach outperformed the other techniques on the used dataset. This approach is generic, and can be used with any feature selection algo-

rithm other than the floating search algorithm, and on regression or classification tasks. However, it was not compared to other ensemble creating techniques such as bagging or boosting.

## 4.4 Genetic Algorithm Approach

Opitz [78] presented a Genetic Algorithm (*GA*) approach for ensemble creation, called Genetic Ensemble Feature Selection (*GEFS*). Opitz noted that GA can be used to search through the large space of feature subsets, and to select the best of such subsets to create a powerful predictor ensemble, such that those subsets create a diverse ensemble. He argued that this task is an enormous problem, and can be tackled using global optimization techniques such as *GA*s. In this technique, a startup population of Artificial Neural Networks (*ANN*s) is initially trained using randomly selected feature subsets (with replacement) with random subset sizes. Then, the *GA* is run on this population, using the genetic operators of mutation and cross-over, producing more fit children in each new population according to a fitness function. The fitness function employed combines the individual network's accuracy with its disagreement (diversity) with the entire ensemble. Experiments were performed using 21 classification datasets from the University of Wisconsin Machine Learning Repository and the UCI data set repository [10]. The results reported are the average of five independent iterations using standard 10-fold cross-validation on the datasets. For each fold an ensemble of 20 networks is created, for a total of 200 networks per dataset, each trained using standard back-propagation learning algorithm. Opitz compared his approach to single network, bagging, and boosting (AdaBoost). The results reported that it compared favorably to bagging and boosting, as it outperformed bagging in 15 datasets, tied in 6; and outperformed boosting in 16, tied in 4, and lost in 1. Obviously, the algorithm is generic and can be used with any predictor type other than Artificial Neural Networks, and also for regression or classification.

Guerra-Salcedo and Whitley [48] proposed another *GA*-based approach for ensemble creation. However, they used table-based predictors, namely KMA [31] and Euclidean Decision Tables (*EDT*)[47]. They applied the CHC genetic search algorithm [39]. Experiments were carried out using three classification datasets from Project StatLog [12] and UCI repository [10]. The datasets were called Segment, LandSat, and DNA and had 19, 36, and 180 features respectively. In their experiments, they compared different combinations of ensemble building techniques with feature selection techniques. For ensemble building, they used all the

training set, bagging, and two versions of boosting. For feature subset selection for each predictor, they applied CHC genetic search, Random Subspace Method [56], and all available features. The results reported were the average of 10 independent runs over the 3 datasets. In each run, CHC was operated 50 times using 50 different random splits of the training set into training and validation sets, and generated 50 good feature subsets. The fitness function employed used only the accuracy of the individual predictors on the validation set, yet it did not consider any diversity measure inside the ensemble. The 50 generated feature subset were distributed over 50 predictors. Guerra-Salcedo and Whitley reported that their approach exhibited encouraging results and outperformed, on the average, the random subspace method and using all the features.

Oliveira et al. [77] also proposed a *GA*-based ensemble creation technique. Their technique also used a *GA* to find good feature subsets, however they employed a hierarchical two-phase approach to ensemble creation. In the first phase, a set of good predictors are generated using Multi-Objective Genetic Algorithm (*MOGA*) search[76]. The base predictors used were Artificial Neural Networks (*ANN*), however any type of predictor can be used. The second phase searches through the space created by the different combinations of these good predictors, again using *MOGA*, to find the best possible combination i.e. the best ensemble. The objective function in the first phase was the accuracy of the *ANN* measured using sensitivity analysis [71]. In the second phase, two objective functions were used: one measuring the generalization error of the ensemble, and the other measuring its ambiguity [63]. The method was experimented in the context of handwritten digit recognition, using NIST SD19 database. Oliveira et al. compared their method to single *ANN* predictors, using three different feature sets. They reported that their method outperformed the single *ANN* and used about 5 predictors in the ensemble, however no comparison to other ensemble creating method was carried out.

## 4.5 Manual Feature Subset Selection Approach

Cherkauer [23] introduced a system called PLANNETT (Person-Level Artificial Neural Networks for ExtraTerrestrial Terrain classification), which combines ANNs in order to achieve better accuracy in the difficult task of recognizing volcanoes in radar images of planet Venus. Experiments were carried out using four images from the Magellan space probe containing together 163 volcanoes that are labeled by human experts [90]. A total set of 119 features are extracted from each part of

the image that contain a potential volcano. These 119 features were divided into 8 hand-selected feature subsets, each one given four ANNs. The ANNs given the same subset differ in the number of hidden nodes, having either 0, 5, 10, or 20 hidden nodes. The 32 networks are trained using standard backpropagation. The final output of the ensemble is the simple average of the 32 ANNs. The PLANNETT system replaces the existing classification module in the JARTOOL system developed at NASA's Jet Propulsion Laboratory [19]. Cherkauer reported that his new classification module, PLANNETT, outperformed the old one, achieving the accuracy of a human planetary geologist. This method is problem-specific and can not be applied systematically to any other dataset.

## 4.6 Correlation-Based Feature Subset Selection Approach

Liao and Moody [65] proposed a technique called *Input Feature Grouping*. The first step in the technique is to group the input features into clusters based on their mutual information, such that features in each group are greatly correlated to each other, and are as little correlated with features in other groups as possible. In the second step, each member of the ensemble is given a representative of each feature cluster. Liao and Moody applied their technique on an economic forecasting problem having 9 input features. A hierarchical clustering algorithm [40] was used to cluster the input features. The inputs were divided into four groups. Eighteen different ANNs were constructed, each given one feature from each of the four groups. They compared their method with three other ensemble techniques: baseline, random selection, and bagging. In the baseline method they trained the networks using all the features, and they only differed in their initial weights. Twenty independent runs were performed for each of the four methods. Liao and Moody reported that their approach outperformed the other three methods.

Tumer and Oza [93, 80] presented an approach called *Input Decimation Ensembles (IDE)*. Their method is only applicable to classification tasks. For a classification problem with $L$ class labels, it constructs $L$ predictors. Each predictor is given a subset of the input features, such that these features are the most correlated with that class. The ensemble final output is the average of the individual predictors output. They compared their method to a single predictor with all input features, and to Principal Component Analysis (*PCA*) ensembles [53], a well-known dimensionality reduction technique. They employed three real world [93]

and three synthetic [80] datasets. Artificial Neural Networks were used as the base predictors, trained using standard backpropagation. Tumer and Oza reported that their method outperformed the single predictor and the *PCA* ensemble, specially in datasets with large number of input features. However, they did not compare their method to other standard ensemble techniques.

## 4.7 Algorithm-Specific Feature Manipulation

Zheng and Webb [99] introduced an approach specific to the decision trees, specially C4.5 [85] decision trees, and called it *Stochastic Attribute Selection Committee (SASC)*. The key idea is to create diverse predictors by manipulating the attributes available at each decision in the tree. For each node, a subset of the attributes (features) is stochastically selected, and that node is restricted to make its decision from these attributes. They carried out experiments on 40 natural domain classification datasets from UCI repository [10], and compared their approach to single C4.5 trees, bagging, and boosting. Two stratified 10-fold cross-validations were run for each algorithm, and the average of the 20 runs were recorded. Zheng and Webb reported that SASC outperformed the single tree and bagging, on average. Nevertheless, boosting was superior to both SASC and bagging. They argue that their method, like bagging, is amenable to being parallelized, unlike boosting in which each tree depends on previous ones. In addition, bagging and SASC are more stable than boosting. Zheng and Webb investigated the idea further, by combining SASC with bagging to form SASCBAG [98], combining SASC with boosting to form SASCB [101], or combining both bagging and boosting with SASC to form SASCMB [100]. All these methods were applied to C4.5 decision trees, and required modification to the base tree induction algorithm.

Dietterich [29] proposes a related approach to generate diverse C4.5 decision forests. The method, at each internal node in the tree, identifies the best 20 possible splits, and chooses one at random. Experiments were undergone on 33 classification datasets from UCI repository [10]. The randomized C4.5 trees were compared to single C4.5 trees, bagging, and boosting. He used ensembles of 200 trees for both randomized C4.5 and bagging, and 100 for boosting. The three ensemble techniques outperformed, as expected, the single tree. Bagging and randomized C4.5 gave quite similar results, while boosting was superior in most of the datasets.

Breiman [16] presented an approach quite near to SASC, called *Random Forests*. In this approach, bagging is combined with random feature selection. Each tree

in the ensemble is trained using a different bootstrap sample of the training set, and the remaining out-of-bag patterns are used as a validation set. There are two versions of the method regarding the random feature selection at each node. The first, called *Forest-RI*, selects a random subset of features to split on. The second version, *Forest-RC*, takes a random combination of randomly selected inputs, and makes the split on these generated features. The base tree predictor used was *CART* [13]. For classification, Breiman compared his method to a single tree, and AdaBoost using 20 classification datasets. For regression, Random Forests were compared to bagging and adaptive bagging [15] using 8 regression datasets. He reported that his method compared favorably to AdaBoost (in classification) and to adaptive bagging (in regression).

## 4.8 Summary

We investigated several techniques for creating diverse ensembles, which all relied on manipulating input features. All, except two, of these techniques were implemented and tested on classification tasks, with some of them applicable only to classification. Some of them were algorithm-specific i.e. can be used only with certain type of learning algorithm. They were all reported to outperform single predictor, and other ensemble techniques, including bagging and boosting, specially in datasets with many input features. This is due to the fact that feature distribution among predictors tends to reduce the dimensionality of the problem.

Table 4.1 presents a summary of the techniques discussed. They are grouped into categories according to the techniques used for distributing the features among predictors in the ensemble. The employed learning algorithm is listed for each technique, together with the kind of task experimented, *C* for Classification and *R* for Regression. The applicability of the technique is denoted in the column labeled *Generic*, with *yes* for completely generic techniques that can be used with any learning algorithm for either classfication or regression.

Table 4.1: Summary of Feature Distribution Techniques

| **Category** | **Method** | **Learning Algorithm** | **Task** | **Generic** |
|---|---|---|---|---|
| Random Feature Selection | RSM [56] | C4.5 trees | C | yes |
| | MFS [8] | NN | C | yes |
| | AB [18] | OC1 trees | C | yes |
| Traditional Feature Selection | [4] | NN, kNN and Guassian | C | yes |
| | [49] | HMM | C | yes |
| GA Feature Selection | GEFS [78] | ANN | C | yes |
| | [47] | KMA and EDT | C | yes |
| | [77] | ANN | C | yes |
| Manual FS | PLANNETT [23] | ANN | C | yes |
| Correlation Feature Selection | IFG [65] | ANN | R | yes |
| | IDE [93, 80] | ANN | C | C only |
| Algorithm-Specific FS | SASC, SASCBAG, SASCB, and SASCMB [99, 98, 101, 100] | C4.5 trees | C | C4.5 only |
| | Randomized C4.5 [29] | C4.5 trees | C | C4.5 only |
| | Random Forests [16] | CART | C & R | CART only |

# Chapter 5

# *FSE* Comparison and Variants

## 5.1 Introduction

We adopted two approaches in this work. The first is to compare basic Feature Subset Ensembles (*FSE*) based on random subset selection of features to three types of single-predictor systems and two well known ensemble systems. This comparison focused on regression datasets. The comparison also tried to figure out the effect of training set size on the accuracy of these techniques. Secondly, we tried to introduce some variants to the basic random subset selection algorithm, in an attempt to increase its accuracy and reliability. These included changing the predictors weights, using bagging, boosting, and forward selection algorithms from within the random feature selection. Then, we tried to investigate other systematic feature partitioning techniqus, in addition to the random feature selection. These included techniques based on feature correlation or on feature relevance.

## 5.2 Datasets

Our focus was on regression tasks, so we used six different regression datasets. Three of them were synthetic, and three were real-world datasets. Table 5.1 gives a summary of the datasets, showing the number of features and number of available examples for each. The datasets were chosen such that they have a substantial number of features, in addition to a large number of patterns. This aids in performing tests that can be near ground truth. The datasets are:

- **bank32nh:** a dataset from the Delve repository [1] synthetically generated

Table 5.1: Dataset Summary. Type '*S*' refers to synthetic datasets, and '*R*' refers to real-world datasets.

| Dataset | No. of Features | No. of Patterns | Type |
|---|---|---|---|
| bank32nh | 32 | 8,192 | *S* |
| ailerons | 40 | 13,750 | *S* |
| syn | 80 | 10,000 | *S* |
| house16h | 16 | 22,784 | *R* |
| comp | 21 | 8,192 | *R* |
| pole | 48 | 15,000 | *R* |

from a simulation of how bank-customers choose their banks. Tasks are based on predicting the fraction of bank customers who leave the bank because of full queues. It consists of 8192 patterns and 32 features.

- **ailerons:** a synthetic dataset from the Weka Project [95]. This data set addresses a control problem, namely flying an F16 aircraft. The attributes describe the status of the aeroplane, while the goal is to predict the control action on the ailerons of the aircraft. It consists of 40 features and 13750 patterns.

- **syn:** a synthetically generated dataset. It consists of 80 features and 10,000 patterns. The dataset is generated such that it contains 40 irrelevant features (with zero weights in the final target) and 40 relevant features (with weights chosen from a unifrom distribution $U \sim (-1, 1)$). The features are also drawn from a multivariate normal distribution such that there is some correlation among the features. A noise term is added to the target, drawn from a normal distribution $N \sim (0, 1)$. The target output is a linear combination of the inputs and the weights, in addition to the noise term.

- **house16h:** a real-world dataset from the Delve repository [1]. It predicts median house prices from the 1990 US census data. It consists of 22,784 patterns and 16 features.

- **comp:** a natural domain dataset from the Delve repository [1]. It predicts a computer system activity from system performance measures. It consists of 8,192 patterns and 21 features.

- **pole:** a natural domain dataset from the Weka Project [95]. The data describes a telecommunication problem. It consists of 15,000 patterns and 48 features.

## 5.3 Comparison

In an attempt to better realize the usefulness of *FSE*, we performed extensive experiments to assess their performance on six different regression datasets. The algorithm used was similar to Ho's Random Subspace Method (*RSM*) [56], which is also quite similar to Attribute Bagging (AB) [18] and Multiple Feature Subsets (MFS) [8]. Algorithm 5 describes the basic Feature Subset Ensembles (*FSE*) algorithm, the version used in this work. In this algorithm, the number of predictors in the ensemble as well as the number of features per predictor are given as inputs. The sampling function to be used, i.e. sampling with or without replacement, is also specified to the algorithm. The final ensemble output is the simple avarage of the individual predictors outputs, i.e. they all have equal weights. This algorithm was compared to five different other algorihtms, three single-predictor and two ensemble techniques. It was compared to: a single predictor using all the features, a single predictor using Sequential Forward Selection (*SFS*) (algorithm 3), a single predictor using Sequential Floating Forward Selection (*SFFS*) (algorithm 4), bagging ensembles using all features (algorithm 1), and boosting (ADABOOST.R) ensembles with all features (algorithm 2).

All these algorithms are generic, in the sense that they can be used with any predictor type. Therefore, to gain further insight into their operation in regression tasks, we used three different predictor types; one linear and two non-linear. The linear is the famous Least Square Error predictor (*LSE*) [70]. The other two non-linear predictors are Artificial Neural Networks (*ANN*) [53] and *CART* Regression Trees [13].

## 5.4 *FSE* Variations

We tried out several variations on the basic *FSE* trying to add more accuracy and stability. First, we tried two sampling functions: with and without replacement. Second, we tried to change the weighting scheme, to give higher emphasis on the more accurate predictors. Two weighting schemes were added beside the normal equal weighting: weight according to training error and weight according

**Algorithm 5** Feature Subset Ensembles $FSE$

**Inputs:**

training set $S = \{(\mathbf{x}_i, d_i),\, i = 1, \ldots, M\}$ where $\mathbf{x}_i = \{x_{i,j},\, j = 1, \ldots, N\}$ is the set of features, $M$ is the number of patterns, and $N$ is the number of features

$train$ : a function that produces a predictor given a training set

$rep$ : determines whether to sample with or without replacement

$K$ : the number of predictor to return in the ensemble

$n$ : the number of features in each predictor

**Outputs:**

an ensemble $P = \{p_i,\, i = 1, \ldots, K\}$

**Procedure:**

**for** $k := 1 : K$ **do** {loop on each predictor}

$f_k := getSubset\,(rep,\, n)$ {get a random feature subset of size $n$}

$P_k = train\,(S,\, f_k)$ {train the predictor using the feature subset chosen and the required training algorithm}

**end for**

to predictor relevance. Third, we tried to mix up bagging and boosting with the *FSE*. Predictors were given random feature subsets as usual, however they are not trained using the whole training set, but rather on a subsample thereof. Fourth, in an effort to increase the diversity among different predictors, we employed feature selection within *FSE*. Individual predictors are given random feature subsets, but they are not all used, instead sequential feature selection is applied to this subset to select the best components out of it.

### 5.4.1 Sampling Functions

The sampling funciton originally used in most of the previous methods, specially $RSM$ [56] and $AB$ [18], was sampling without replacement. In this method, each predictor is given a subset of features by sampling without replacement from the pool of features i.e. when a feature is chosen in the subset for a certain predictor, it can not be further chosen for the *same* predictor, while it can be chosen in other predictors. On the other hand, $MFS$ [8] used sampling with replacement in addition to sampling with replacement. In this technique, a given feature can be repeated in the subset chosen for a given predictor. Effectively, this is a way

of reducing the number of features chosen as these replicated features add no new information to the predictor. We experimented with both sampling techniques, in order to see the effect of the sampling function on the output of the ensembles.

### 5.4.2 Weighting Functions

Many of the ensemble techniqes relied on equal weighting of the individual predictors [30]. The predictors have equal votes (in case of classification) or equal weights in averaging (in case of regression). Furthermore, some ensemble techniques assigned weights to component predictors relative to their performance, either on the training set or on some validation set [18, 44]. This weighting mechanism proved useful in some situations, specially in boosting, to give larger emphasis on better predictors.

We tried out the above two weighting approaches: equal weighting and weighting according to predictor performance on the training set. In equal weighting, each predictor $k$ was assigned a weight $w_k = \frac{1}{K}$, where $K$ is the ensemble size. In the second method, each predictor's weight was computed using the softmax formula: $w_k = \frac{\exp(-e_k)}{\sum_{i=1}^{K} \exp(-e_i)}$, where $e_k$ is the training error for predictor $k$.

In addition to these two techniques, we used a third weighting mechanism. This relied on the relevance of the features selected for each predictor. The features in the dataset were ranked according to their relevance [59], employing a variant of sequential forward selection (*SFS*) algorithm with 10-fold cross-validation. The algorithm stops when all the features are selected in the subset, and bases its estimate of the subset performance on the result of 10-fold cross-validation [60]. Then, each predictor is given a weight according to the ranks of the features in its feature subset. The weight given to predictor $k$ is defined by $w_k = \frac{w'_k}{\sum_{i=1}^{K} w'_i}$ where $w'_k = \sum_{i=1}^{n} r_i$, $r_i$ is the rank of feature $i$, $n$ is number of features in each subset, and $K$ is the total number of predictors. This weighting scheme measures the strength of each predictor in terms of the strength of features it has. It assigns larger weights to hopefully more relevant predictors, those that have the most relevant features.

### 5.4.3 Training Set Subsampling

Training set subsampling has beeen proved effective in creating accurate diverse ensembles. The two most successful ensemble techniques, bagging and boosting, are examples of training set subsampling. Therefore, we tried to embed both of

bagging and boosting into *FSE*, as this has the potential to increase the diversity of the produced ensembles. This seemed attractive, as it combined two orthogonal approaches, feature set subsampling and training set subsampling, and they both perform well independently. Hence, we tried to test their behaviour when working together.

When using bagging within *FSE*, each predictor gets its own feature subset, and then trains on an independent bootstrap sample drawn from the original training set. The advantage of this approach, like bagging, is that the individual predictors are completely independent of each other, and can be created and trained in parallel.

On the other hand, using boosting within *FSE* is a bit more complicated. The version of boosting implemented is ADABOOST.R described in algorithm 2. After providing each predictor with its feature subset, they are trained in sequence, with each one given a different sample of the training set based on the performance of the previous predictors. Those patterns that exhibit higher error with the previous predictor are given higher probability of beging represented in the sample of the new predictor. Thus, harder patterns are given more emphasis than easier ones and the predictor can focus its attention to learning those hard-to-learn paterns.

### 5.4.4 Embedded Feature Selection

We tried yet another technique to help increase diversity of *FSE*. In using randomly selected feature subset for each predictor, features can get mingled together in a way that might worsen that predictor's performance. Some researchers already used other feature selection techniques for building predictor ensembles, e.g. genetic algorithms [48, 77, 78] and other mechanisms [4, 49]. Hence, we thought of using feature selection after the random feature sampling. In this approach, and after giving each predictor its random share of the features, a feature selection algorithm is applied to choose the best out of those features. Two algorithms were tried: Sequential Forward Selection $SFS$ (algorithm 3) and Sequential Floating Forward Selection $SFFS$ (algorithm 4). The former is a simple and well-established feature selection algorithm. The latter has been proved more accurate and stable [58]. The versions of the algorithms employed use 10-fold cross-validation for error estimation and stop when the performance starts to deteriorate.

### 5.4.5 Feature Subset Selection Criteria

In addition to the above variations to the basic *FSE*, we tried other systematic feature subset selection criteria. The basic *FSE* uses random feature subsets for each predictor. We thought that having a more advanced feature selection criteria that can make use of the knowledge about the features, their relevance and correlation, has the potential of providing better results. Two criteria were tried out for feature subset selection: feature relevance and feature correlation.

#### 5.4.5.1 Feature Relevance Criteria

Each feature has some inherent strength with respect to the dataset it represents. Knowing those strong features can help us partition the features into strong diverse subsets. The features relevances are determined as described earlier in section 5.4.2 using a variant of sequential forward selection $SFS$ algorithm. Then, this ranked feature list is divided into two equal lists: strong features list and weak features list. Two techniques are employed to select feature subsets for the predictors, which we will call pure and hybrid techniques. In the pure technique, each predictor takes its subset randomly from either the strong list or the weak list. Half the predictors take all their features from the strong list, and the rest take theirs from the weak list. On the other hand, each predictor in the hybrid technique takes half its subset from the strong list and the other half from the weak list.

#### 5.4.5.2 Feature Correlation Criteria

Correlation between features is a major issue in mahine learning. Using highly correlated features adds little information to the learning process. We tried to have a measure of the corrlation between features, and wanted to roughly divide them into a group correlated features and antoher of uncorrelated features. We employed a heuristic approach to achieve this objective. First, the correlation correlation coefficient matrix is calculated for the features from the training set. Then, the features are ranked descendingly according to the abosolute value of their mutual correlation. This is achieved by choosing the maximum correlation value, extracting the two features intersecting in this value, and adding them to the output list. Then, the next highest value is found, and so on. After that, the feature list is again divided into two equal lists: the correlated list and the uncorrelated list. Likewise, two techniques are used to select feature subsets for the predictors from these two lists. In the pure technique, half the predictors take all their features

from the strong list and the rest take theirs from the weak list. In the hybrid, on the other hand, each predictor chooses a mixture of features from the two lists.

## 5.5 Summary

We followed two approaches in this work. First, comparing basic *FSE* to other techniques, both single-predictor and ensemble-based. Second, we added several variants to the basic *FSE* method in an attempt to improve its performance. These variations aim at enhancing the performance and accuracy of basic *FSE*. They included using another sampling function, weighting functions, adding training set subsampling, adding embedded feature selection, and using systematic feature selection methods other than the random feature selection.

# Chapter 6

# Experiments and Results

## 6.1 Introduction

We carried out extensive experiments in two directions: comparison of the *FSE* with other ensemble and non-ensemble techniques, and testing the variations introduced to the basic *FSE*. Three predictor types were used, and several runs were averaged to provide the final results stated below. The results emphasized the superiority of basic *FSE* with some predictor types, in addition to the effectiveness of the *FSE* variants suggested in this work in improving its performance.

## 6.2 Experimental Setup

### 6.2.1 Development Tools

We used Mathworks Matlab 7 R14 as the programming environment for developing the algorithms. We implemented *FSE* together with its variants, *SFS*, *SFFS*, bagging, and boosting. Matlab has built-in implementations for *LSE*, *ANN*, and *CART* trees predictors. Simulations were carried out on a Compaq Proliant ML server with four Pentium Xion processors at 1.4GHz.

### 6.2.2 Predictors Types

#### 6.2.2.1 Least Squares Error (*LSE*)

The *LSE* [70] just tries to find the weight vector that minimizes the square error in mapping the input patterns matrix onto the target vector. Given $X \in R^{M \times N}$ is

the input matrix with $M$ patterns and $N$ features, $d \in R^{M\times 1}$is the target vector, *LSE* finds the weight vector $w \in R^{N\times 1}$ such that $w = (X^T X)^{-1} X^T d$, where $X^T$ means the transpose of matrix $X$. This is equivalent to finding the projection of the target vector onto the column space of the input matrix. It is clear that *LSE* has no parameters to be set, and is considered the fastest predictor type used in this work.

#### 6.2.2.2 Artificial Neural Networks (*ANN*)

Artificial Neural Networks are well known general purpose predictors [53]. They have been successfully used for both classification and regression tasks. There are many types of *ANN* with respect to their training methodology, network structure, number of layers, and interconnections between layers and neurons. The type of networks used here is Feedforward Single Hidden Layer Multilayer Perceptron ANNs. They contain an input layer, a single hidden layer, and one output layer with no feedback connections between neurons. The size of the input layer equals the number of features in the dataset, and the output layer contains a single neuron. The number of hidden nodes was chosen to be 5 neurons. This size seemed sufficient for the training set size to prevent overfitting. The training algorithm employed is the Levenberg-Marquardt backpropagation algorithm [46] with default training settings.

#### 6.2.2.3 Classification and Regression Trees (*CART*)

*CART* (Classification And Regression Trees) [13] are also general purpose predictors used for both classification and regression. Given a training set, the *CART* algorithm builds a binary tree splitting the space according to the training data. Then, it can get the predicted value of any unknown input by following the path from the root to the leaves of the tree.

### 6.2.3 Performance Measure

The normal performance measure employed in regression is the mean square error. However, we noticed that this measure depends on the mean of the dataset values, and lower means square error in one dataset may be worse than a higher mean square error in another dataset. Therefore, we adopted another performance measure, namely Normalized Mean Squared Error (*NMSE*), which is defined by $NMSE = \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i \hat{y}_i^2} \times 100\,\%$, where $\hat{y}_i$ is the predicted value and $y_i$ is the target

value. This measure normalizes the normal means square error by the square of the target value, and so give an error estimate relative to the mean of the dataset.

## 6.3 Results and Discussions

### 6.3.1 Comparisons

To be able to get a fair comparison between *FSE* and other techniques, we performed more than one run on the datasets. Ten independent runs for each dataset were performed. In each of the runs, a random subset is chosen from the dataset for training, with the remaining used for testing. The training set size was chosen to be 200 patterns. For each predictor type, *FSEs* were compared to a single predictor using all the features, *SFS*, *SFFS*, bagging, and boosting.

Four different ensemble sizes were attempted: 10, 25, 50, and 100. In addition, five different feature subset sizes were tried for FSEs. We tried $\{0.1, 0.3, 0.5, 0.7, 0.9\} \times N$, where $N$ is the number of features in the dataset. Hence, 20 different combinations were performed for FSE; four different ensemble sizes and five different subset sizes for each ensemble size. We experimented with different subset sizes as the effective number of features is dataset dependent, and can not be fixed in advance.

For these set of experiments, we employed the basic FSE algorithm, where equal weighting is used and features are selected for each predictor using sampling without replacement. This means that a given feature can not be replicated within the same predictor, but can be selected for different predictors. Comparison results using LSE predictors are shown in tables 6.1-6.2, where $K$ respresents the number of individual predictors, and $n$ denotes the number of features in each individual predictor of the ensemble. Bold values represent the best result in each technique. Table 6.3 represents a summary for LSE predictors, where the best result for each of *FSE*, bagging and boosting is shown. Tables 6.4-6.9 give the corresponding results for *ANN* and *CART* predictors.

The results reveal some interesting points. First, they show the superiority of *FSE* over the other techniques when using the linear *LSE* predictor. The *FSE* was followed by the sequential selection methods then came the bagging and boosting. Second, they indicate, as expected, that bagging and boosting outperform the other methods when using non-linear predictors (*ANN* and *CART*). This can lend itself to the non-linear and unstable nature of these predictors. This nature produces more uncorrelated diverse ensembles, which helps increase their accuracy [14].

For the *LSE* predictor, *FSE* produced the best results in four out of the six datasets, and came the third in the remaining two datasets. Bagging was better than boosting in all but one dataset. Sequential selection techniques were also better than bagging in five datasets. Boosting was the worst technique in all the datasets.

This picture changes considerably when using non-linear predictors such as *ANN* and *CART*. Bagging and boosting led, on the average, all other techniques in these experiments. For the *ANN* predictors, boosting was the best predictor in four datasets, *FSE* in one, and *SFS* in one. In five datasets *FSE* was the third, and in all the datasets bagging was the second. The sequential selection techniques did not prove useful with this kind of predictor, but they were, on the average, better than the single predictor.

For *CART* predictors, *FSE* was the best in two, and the third in the remaining. Boosting was the best in two, the second in two and the third in two. Bagging was the first in two datasets and the second in the rest. Sequential selection methods lagged the other ensemble techniques.

Table 6.1: Results for basic $FSE$s using $LSE$ predictors, where $K$ is the ensemble size and $n$ is the feature subset size

| | FSE | | | | |
|---|---|---|---|---|---|
| Dataset | | $K$ | | | |
| | $n$ | 10 | 25 | 50 | 100 |
| bank32nh | 4 | 60.52 | 61.48 | 62.23 | 62.59 |
| | 10 | 52.19 | 52.76 | 50.44 | 51.55 |
| | 16 | 45.06 | 45.77 | 44.92 | 44.31 |
| | 22 | 40.13 | 39.35 | 39.85 | 39.47 |
| | 28 | 38.83 | 38.65 | 38.64 | **38.37** |
| ailerons | 4 | 9.54 | 9.78 | 9.93 | 9.66 |
| | 12 | 7.73 | 7.36 | 7.27 | 7.21 |
| | 20 | 5.46 | 5.70 | 5.55 | 5.53 |
| | 28 | 4.71 | 4.66 | 4.56 | 4.59 |
| | 36 | 4.41 | 4.43 | **4.34** | 4.37 |
| syn | 8 | 71.56 | 71.86 | 70.10 | 70.96 |
| | 24 | 48.18 | 47.25 | 46.89 | 46.36 |
| | 40 | 40.85 | 39.78 | 39.55 | 39.10 |
| | 56 | 40.02 | 39.10 | 38.66 | **38.58** |
| | 72 | 43.55 | 43.09 | 42.72 | 42.72 |
| house16h | 2 | 39.01 | 37.82 | 37.68 | 37.83 |
| | 5 | 35.08 | 34.25 | 34.75 | 33.81 |
| | 8 | 36.79 | 35.22 | 33.69 | **33.59** |
| | 11 | 35.77 | 35.80 | 35.97 | 36.13 |
| | 14 | 39.35 | 38.90 | 39.63 | 39.02 |
| comp | 3 | 34.49 | 35.73 | 35.39 | 34.39 |
| | 8 | 18.73 | 18.06 | 17.67 | 18.72 |
| | 11 | 14.45 | 10.99 | 9.96 | 10.05 |
| | 15 | 7.05 | 6.18 | 6.15 | 6.16 |
| | 19 | **4.33** | 4.35 | 4.40 | 4.35 |
| pole | 55 | 69.91 | 69.23 | 67.81 | 68.98 |
| | 15 | 50.44 | 49.90 | 50.15 | 49.27 |
| | 24 | 43.28 | 41.44 | 41.49 | 41.25 |
| | 33 | 41.47 | 40.71 | **40.46** | 40.51 |
| | 43 | 44.73 | 44.26 | 44.50 | 44.40 |

Table 6.2: Results for other techniques using LSE predictors

| Dataset | $K$ | Bagging | Boosting | Single | SFS | SFFS |
|---|---|---|---|---|---|---|
| bank32nh | 10 | 40.47 | **51.46** | 39.71 | 38.01 | 38.05 |
| | 25 | 40.01 | 57.87 | | | |
| | 50 | 39.81 | 63.42 | | | |
| | 100 | **39.79** | 66.55 | | | |
| ailerons | 10 | 4.42 | **5.57** | 4.43 | 4.35 | 4.35 |
| | 25 | **4.36** | 6.48 | | | |
| | 50 | 4.39 | 7.25 | | | |
| | 100 | 4.37 | 7.48 | | | |
| syn | 10 | 53.62 | 59.30 | 47.81 | 42.50 | 43.46 |
| | 25 | 49.37 | 56.28 | | | |
| | 50 | 48.99 | 54.26 | | | |
| | 100 | **48.65** | **53.72** | | | |
| house16h | 10 | 42.69 | **88.96** | 41.79 | 42.98 | 42.98 |
| | 25 | 41.72 | 120.7 | | | |
| | 50 | 40.86 | 110.9 | | | |
| | 100 | **40.68** | 118.2 | | | |
| comp | 10 | 4.46 | **4.68** | 4.36 | 3.77 | 3.77 |
| | 25 | 4.47 | 4.77 | | | |
| | 50 | 4.45 | 4.95 | | | |
| | 100 | **4.44** | 5.21 | | | |
| pole | 10 | 58.76 | 61.38 | 47.43 | 40.87 | 41.41 |
| | 25 | 53.26 | 47.04 | | | |
| | 50 | 59.58 | 43.55 | | | |
| | 100 | **51.58** | **43.45** | | | |

Table 6.3: Comparison results summary using LSE predictors

| | Dataset | | | | | |
|---|---|---|---|---|---|---|
| Alg. | bank32nh | ailerons | syn | house16h | comp | pole |
| $FSE$ | 38.37 | **4.34** | **38.58** | **33.59** | 4.33 | **40.46** |
| Bagging | 39.79 | 4.36 | 48.65 | 40.68 | 4.44 | 51.58 |
| Boosting | 51.46 | 5.57 | 53.72 | 88.96 | 4.68 | 43.45 |
| Single | 39.71 | 4.43 | 47.81 | 41.79 | 4.36 | 47.43 |
| $SFS$ | **38.01** | 4.35 | 42.50 | 42.98 | **3.77** | 40.87 |
| $SFFS$ | 38.05 | 4.35 | 43.46 | 42.98 | 3.77 | 41.41 |

Table 6.4: Results for basic $FSE$s using $ANN$s predictors, where $K$ is the ensemble size and $n$ is the feature subset size

| | FSE | | | | |
|---|---|---|---|---|---|
| Dataset | | $K$ | | | |
| | $n$ | 10 | 25 | 50 | 100 |
| bank32nh | 4 | 76.85 | 84.98 | 65.91 | 61.70 |
| | 10 | 68.58 | 59.21 | 54.21 | 51.54 |
| | 16 | 82.30 | 56.62 | 50.71 | 44.92 |
| | 22 | 62.35 | 46.83 | 41.71 | 42.79 |
| | 28 | 62.08 | 42.46 | 41.61 | **40.81** |
| ailerons | 4 | 37.71 | 12.86 | 11.86 | 10.86 |
| | 12 | 26.73 | 15.25 | 10.31 | 10.67 |
| | 20 | 18.71 | 13.30 | 15.96 | 7.90 |
| | 28 | 20.26 | 8.33 | 7.19 | 5.66 |
| | 36 | 6.08 | 8.09 | 6.76 | **4.82** |
| syn | 8 | 68.72 | 112.2 | 47.58 | 47.06 |
| | 24 | 80.40 | 38.97 | 33.74 | 33.56 |
| | 40 | 36.19 | 30.13 | 28.11 | 27.18 |
| | 56 | 31.39 | 27.39 | 26.30 | 25.53 |
| | 72 | 29.48 | 26.24 | 25.44 | **24.68** |
| house16h | 2 | 33.82 | 38.92 | 19167 | **43.78** |
| | 5 | 162.0 | 80.84 | 43.06 | 50.98 |
| | 8 | 234.1 | 89.67 | 57.44 | 50.61 |
| | 11 | 251.1 | 87.68 | 64.18 | 58.68 |
| | 14 | 246.9 | 75.20 | 83.39 | 70.85 |
| comp | 3 | 7.47 | 3.31 | 3.37 | 2.09 |
| | 7 | 7.43 | 2.82 | 1.75 | 1.53 |
| | 11 | 2.05 | 1.64 | 1.09 | 0.68 |
| | 15 | 0.85 | 0.56 | 0.55 | 0.36 |
| | 19 | 0.51 | 5.99 | 0.29 | **0.28** |
| pole | 5 | 2257 | 41.64 | 328.8 | 28.66 |
| | 15 | 219.9 | **25.15** | 27.78 | 30.38 |
| | 24 | 153.0 | 62.45 | 26.63 | 29.80 |
| | 33 | 81.39 | 36.31 | 29.27 | 33.15 |
| | 43 | 63.32 | 37.25 | 27.83 | 29.63 |

Table 6.5: Results for other techniques using ANNs predictors

| Dataset | $K$ | Bagging | Boosting | Single | SFS | SFFS |
|---|---|---|---|---|---|---|
| bank32nh | 10 | 47.02 | 51.99 | 113.9 | 181.9 | 64.07 |
| | 25 | 41.75 | 42.99 | | | |
| | 50 | 41.41 | 39.83 | | | |
| | 100 | **38.82** | **38.78** | | | |
| ailerons | 10 | 7.30 | 5.71 | 325.1 | 15.13 | 12.86 |
| | 25 | 5.47 | 4.94 | | | |
| | 50 | 4.98 | 4.42 | | | |
| | 100 | **4.52** | **4.21** | | | |
| syn | 10 | 30.35 | 31.88 | 76.84 | 65.89 | 83.14 |
| | 25 | 26.77 | 27.06 | | | |
| | 50 | 26.18 | 26.18 | | | |
| | 100 | **25.30** | **25.20** | | | |
| house16h | 10 | 76.32 | 33.89 | 971.3 | 92.48 | 103.4 |
| | 25 | 70.29 | 26.64 | | | |
| | 50 | 49.85 | 27.29 | | | |
| | 100 | **47.77** | **24.03** | | | |
| comp | 10 | 0.94 | 0.45 | 3.67 | 0.40 | 0.72 |
| | 25 | 0.51 | 0.24 | | | |
| | 50 | 0.51 | 0.19 | | | |
| | 100 | **0.35** | **0.18** | | | |
| pole | 10 | 42.05 | **676.2** | 139.0 | 11.66 | 18.96 |
| | 25 | 33.67 | 761.5 | | | |
| | 50 | **27.29** | 4112 | | | |
| | 100 | 30.17 | 2435 | | | |

Table 6.6: Comparison results summary using *ANN* predictors

| | Dataset | | | | | |
|---|---|---|---|---|---|---|
| Alg. | bank32nh | ailerons | syn | house16h | comp | pole |
| FSE | 40.81 | 4.82 | **24.68** | 43.78 | 0.28 | 25.15 |
| Bagging | 38.82 | 4.52 | 25.30 | 47.77 | 0.35 | 27.29 |
| Boosting | **38.78** | **4.21** | 25.20 | **24.03** | **0.18** | 676.2 |
| Single | 113.9 | 325.1 | 76.84 | 971.3 | 3.67 | 139.0 |
| SFS | 181.9 | 15.13 | 65.89 | 92.48 | 0.40 | **11.66** |
| SFFS | 64.07 | 12.86 | 83.14 | 103.4 | 0.72 | 18.96 |

Table 6.7: Results for basic $FSE$s using $CART$ predictors, where $K$ is the ensemble size and $n$ is the feature subset size

| | FSE | | | | |
|---|---|---|---|---|---|
| Dataset | | $K$ | | | |
| | $n$ | 10 | 25 | 50 | 100 |
| bank32nh | 4 | 65.89 | 64.23 | 62.83 | 61.85 |
| | 10 | 58.05 | 55.34 | 54.95 | 55.08 |
| | 16 | 52.31 | 51.05 | 50.73 | 50.10 |
| | 22 | 53.82 | 50.32 | **49.90** | 49.43 |
| | 28 | 59.68 | 57.38 | 56.87 | 56.65 |
| ailerons | 4 | 10.86 | 9.96 | 9.84 | 9.74 |
| | 12 | 8.35 | 8.17 | 7.87 | 7.69 |
| | 20 | 6.98 | 6.90 | 6.69 | 6.60 |
| | 28 | 6.81 | 6.57 | 6.43 | **6.43** |
| | 36 | 7.36 | 7.44 | 7.29 | 7.20 |
| syn | 8 | 85.09 | 81.06 | 79.86 | 78.95 |
| | 24 | 76.50 | 70.80 | 70.09 | **69.21** |
| | 40 | 75.99 | 71.98 | 69.90 | 69.38 |
| | 56 | 79.76 | 78.27 | 75.29 | 75.69 |
| | 72 | 101.2 | 100.3 | 97.88 | 97.72 |
| house16h | 2 | 42.07 | 40.31 | 39.94 | 39.53 |
| | 5 | 38.27 | 36.85 | **35.84** | 35.89 |
| | 8 | 38.44 | 37.39 | 37.56 | 36.70 |
| | 11 | 41.98 | 40.63 | 40.68 | 39.71 |
| | 14 | 48.65 | 46.95 | 47.34 | 46.82 |
| comp | 3 | 1.84 | 1.83 | 1.59 | 1.48 |
| | 7 | 0.44 | 0.39 | 0.43 | 0.43 |
| | 11 | 0.18 | 0.20 | 0.20 | 0.19 |
| | 15 | 0.21 | 0.18 | 0.18 | **0.17** |
| | 19 | 0.22 | 0.22 | 0.22 | 0.22 |
| pole | 5 | 58.04 | 53.63 | 54.52 | 54.74 |
| | 15 | 36.75 | 32.91 | 32.56 | 31.82 |
| | 24 | 21.48 | 19.80 | 20.42 | 19.20 |
| | 33 | 14.91 | 13.51 | 13.07 | 13.02 |
| | 43 | 12.74 | 11.75 | **11.72** | 11.75 |

Table 6.8: Results for other techniques using CART predictors

| Dataset | $K$ | Bagging | Boosting | Single | SFS | SFFS |
|---|---|---|---|---|---|---|
| bank32nh | 10 | 46.51 | 51.41 | 75.98 | 68.21 | 68.49 |
| | 25 | 44.22 | 46.37 | | | |
| | 50 | 43.43 | 44.84 | | | |
| | 100 | **43.32** | **44.03** | | | |
| ailerons | 10 | 5.69 | 6.27 | 8.71 | 9.21 | 9.22 |
| | 25 | 5.41 | 5.53 | | | |
| | 50 | 5.24 | 5.30 | | | |
| | 100 | **5.20** | **5.27** | | | |
| syn | 10 | 73.03 | 77.19 | 130.5 | 141.1 | 100.8 |
| | 25 | 69.07 | 68.66 | | | |
| | 50 | 68.25 | 65.32 | | | |
| | 100 | **67.08** | **63.44** | | | |
| house16h | 10 | 40.20 | 43.49 | 59.18 | 61.48 | 52.92 |
| | 25 | 37.70 | 41.39 | | | |
| | 50 | 37.33 | **40.64** | | | |
| | 100 | **37.07** | 40.69 | | | |
| comp | 10 | 0.18 | 2.79 | 0.30 | 0.26 | 0.26 |
| | 25 | 0.18 | 3.47 | | | |
| | 50 | 0.17 | 3.30 | | | |
| | 100 | **0.17** | **3.10** | | | |
| pole | 10 | 10.04 | 23.48 | 14.57 | 20.43 | 20.39 |
| | 25 | 9.49 | 13.41 | | | |
| | 50 | 9.45 | 8.92 | | | |
| | 100 | **9.20** | **7.88** | | | |

Table 6.9: Comparison results summary using *CART* predictors

| | Dataset | | | | | |
|---|---|---|---|---|---|---|
| Alg. | bank32nh | ailerons | syn | house16h | comp | pole |
| $FSE$ | 49.90 | 6.43 | 69.21 | **35.84** | **0.17** | 11.72 |
| Bagging | **43.32** | **5.20** | 67.08 | 37.07 | 0.17 | 9.20 |
| Boosting | 44.03 | 5.27 | **63.44** | 40.64 | 3.10 | **7.88** |
| Single | 75.98 | 8.71 | 130.5 | 59.18 | 0.30 | 14.57 |
| $SFS$ | 68.21 | 9.21 | 141.1 | 61.48 | 0.26 | 20.43 |
| $SFFS$ | 68.49 | 9.22 | 100.8 | 52.92 | 0.26 | 20.39 |

### 6.3.2 *FSE* Variants

We introduced many novel variations to the basic *FSE* algorithm. They included changing the sampling functions, weighting functions, using training set subsampling, and embedded feature selection. In addition, new systematic feature selection techniques were proposed. The results of such variations are summarized below. The *FSE* was run, like the above experiments, using 4 ensemble sizes (10, 25, 50, 100) and five different subset sizes. Ten independent runs were performed and averaged. The best of the 20 averaged results for each dataset are shown in the tables below, compared with the basic *FSE* results. Datasets that exhibited enhanced performance are shown in **bold face**.

#### 6.3.2.1 Sampling Function

The default $FSE$ used sampling without replacement, in which a given feature can not be replicated for a given predictor but can be used by different predictors. We ran experiments to test the effect of sampling with replacement on its performance. For each predictor, features are allowed to be replicated within its subset. Table 6.10 summarizes the results of this variation and compares them with the basic *FSE* results for the three predictor types. Bold values represent the best value for each dataset.

The results show that this variation was more apparent in the *LSE* predictors, where the performance was better in half the datasets. For the *ANN* and *CART*, it was less useful as it improved the performance in only two of the datasets. This shows that it is can be more useful with linear predictors than non-linear ones.

Table 6.10: Summary for variation of *FSE* using sampling with replacement

| | *FSE* | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|---|
| *LSE* | Basic | 38.37 | 4.34 | 38.58 | 33.59 | 4.33 | 40.46 |
| | Samp. | 41.03 | 4.89 | **38.47** | **31.40** | 13.09 | **40.05** |
| *ANN* | Basic | 40.81 | 4.82 | 24.68 | 43.78 | 0.28 | 25.15 |
| | Samp. | 42.51 | 6.38 | 26.48 | **28.30** | 0.97 | **24.71** |
| *CART* | Basic | 49.90 | 6.43 | 69.21 | 35.84 | 0.17 | 11.72 |
| | Samp. | **49.20** | 6.43 | **69.04** | 36.17 | 0.19 | 15.61 |

#### 6.3.2.2 Weighting Function

Two more weighting functions were experimented instead of the equal weighting used with basic FSEs. The first is weighting according to training error of individual predictors, and the second is weighting according to feauture relevance. These two weighting functions are used with the default sampling function (sampling without replacement) of $FSE$s. A summary of this variation is presented in table 6.11.

The relevance weighting provided better results in three datasets for the *LSE* predictors, in three for the *ANN*, and in five for the *CART*. The error weithing, on the other hand, enhanced the performance in one dataset for *LSE*, in four for the *ANN*, and in five for the *CART*. Therefore, weighting according to training error proved more useful in non-linear predictors while relevance weighting was better for linear predictors.

Table 6.11: Summary for variation of *FSE* using different weighting functions. "Wt R" refers to relevance weighting, while "Wt E" is error weighting

| | *FSE* | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|---|
| *LSE* | Basic | 38.37 | 4.34 | 38.58 | 33.59 | 4.33 | 40.46 |
| | Wt R | 38.54 | **4.34** | 38.64 | **33.48** | 4.41 | **40.30** |
| | Wt E | 38.42 | 4.37 | **38.50** | 33.99 | 4.35 | 40.51 |
| *ANN* | Basic | 40.81 | 4.82 | 24.68 | 43.78 | 0.28 | 25.15 |
| | Wt R | **39.54** | 5.30 | 24.92 | **30.07** | 0.31 | **23.06** |
| | Wt E | **39.97** | 5.71 | 24.76 | **28.73** | **0.27** | **23.46** |
| *CART* | Basic | 49.90 | 6.43 | 69.21 | 35.84 | 0.17 | 11.72 |
| | Wt R | **49.30** | **6.39** | 69.61 | **35.57** | **0.17** | **11.61** |
| | Wt E | **49.61** | **6.36** | **69.16** | 35.99 | **0.17** | **11.47** |

#### 6.3.2.3 Training Set Subsampling

Bagging and boosting were implemented within the basic FSE in an attempt to increase the diversity of the produced ensembles. Table 6.12 provides a summary for these results.

Training set subsampling did not exhibit any accuracy improvement in linear *LSE* predictor. However, bagging was better in one dataset for *ANN* and in all the datasets for *CART*. Boosting also was better in one dataset for *ANN* and in five datasets for *CART*. This clearly shows that subsampling the training set is more useful for non-linear predictors than for linear ones.

Table 6.12: Summary for variation of *FSE* using training set subsampling. "Bag" refers to using bagging, while "Bst" refers to using boosting

| | *FSE* | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|---|
| *LSE* | Basic | 38.37 | 4.34 | 38.58 | 33.59 | 4.33 | 40.46 |
| | Bag | 38.48 | 4.39 | 39.33 | 55.16 | 4.68 | 46.03 |
| | Bst | 51.60 | 5.52 | 43.30 | 100.20 | 4.79 | 43.17 |
| *ANN* | Basic | 40.81 | 4.82 | 24.68 | 43.78 | 0.28 | 25.15 |
| | Bag | 42.97 | **4.46** | 39.61 | 67.36 | 0.37 | 44.00 |
| | Bst | 41.84 | **4.77** | 40.13 | 72.98 | 0.37 | 74.61 |
| *CART* | Basic | 49.90 | 6.43 | 69.21 | 35.84 | 0.17 | 11.72 |
| | Bag | **42.13** | **5.09** | **66.17** | **34.47** | **0.16** | **10.98** |
| | Bst | **49.08** | **5.33** | **63.57** | 37.05 | **0.17** | **11.28** |

#### 6.3.2.4 Embedded Feature Selection

Sequential feature selection techniques were employed within the basic FSE to choose the best feature subset for each predictor. Table 6.13 shows the summary of these experiments.

The results clearly show that the embedded feature selection was very useful for the linear *LSE* predictors. Embedded *SFS* was better in four datasets while embedded *SFFS* was better in five of the six datasets. Likewise, it was also useful for the nonlinear predictors. For *ANN*, it improved the performace in three datasets for each of *SFS* and *SFFS*, while in *CART* it improved the performance in two datasets for each of the feature selection algorithms.

Table 6.13: Summary for variation of *FSE* using embedded feature selection. "SFS" refers to using *SFS* algorithm, while "SFFS" refers to using *SFFS* algorithm

| | *FSE* | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|---|
| *LSE* | Basic | 38.37 | 4.34 | 38.58 | 33.59 | 4.33 | 40.46 |
| | SFS | **36.21** | **4.14** | 38.82 | 51.11 | **3.82** | **38.76** |
| | SFFS | **36.57** | **4.32** | **37.54** | 53.83 | **3.79** | **39.13** |
| *ANN* | Basic | 40.81 | 4.82 | 24.68 | 43.78 | 0.28 | 25.15 |
| | SFS | **39.74** | **2.15** | 28.69 | **42.73** | 0.32 | **12.07** |
| | SFFS | 41.73 | **4.53** | 27.56 | **38.16** | 0.41 | **10.03** |
| *CART* | Basic | 49.90 | 6.43 | 69.21 | 35.84 | 0.17 | 11.72 |
| | SFS | **49.77** | 7.56 | 85.38 | 38.89 | **0.17** | 12.67 |
| | SFFS | 68.61 | 7.21 | 100.1 | 39.04 | **0.16** | **11.19** |

#### 6.3.2.5 Feature Subset Selection Criteria

Two new criteria, other than random selection, are used in the process of selecting feature subsets for individual predictors. These are: feature relevance and feature correlation criteria.

**6.3.2.5.1 Feature Relevance Criteria** Features are divided according to their relevance into two lists: strong list and weak list. Features are drawn from these two lists using two techniques: either all features come in turn from one of them (the pure technique), or features are selected from both lists (the hybrid technique). In the pure technique, it is apparent that not all the subset sizes are possible, as each predictor is given its subset from half the original number of features, therefore only subset sizes less than or equal to half the total number of features are available. Table 6.14 provides a summary of the results.

It is clear from the results that the pure technique did not prove much useful, as it failed to enhance the results in any dataset. However, the hybrid technique was more successful. It produced better results in two datasets using *LSE*, in two using *ANN*, and in five using *CART*.

Table 6.14: Summary for variation of *FSE* using feature relevance criteria. "P" refers to using the pure technique, while "H" refers to the hybrid one

| | *FSE* | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|---|
| *LSE* | Basic | 38.37 | 4.34 | 38.58 | 33.59 | 4.33 | 40.46 |
| | P | 44.49 | 5.33 | 42.73 | 48.62 | 19.15 | 45.55 |
| | H | **38.13** | **4.34** | 39.52 | 52.24 | **3.78** | **39.05** |
| *ANN* | Basic | 40.81 | 4.82 | 24.68 | 43.78 | 0.28 | 25.15 |
| | P | 45.42 | 7.96 | 47.13 | 46.10 | 3.05 | 70.86 |
| | H | **39.84** | **4.54** | 41.20 | 51.66 | 0.63 | 64.06 |
| *CART* | Basic | 49.90 | 6.43 | 69.21 | 35.84 | 0.17 | 11.72 |
| | P | 54.42 | 8.51 | 70.25 | 36.10 | 0.79 | 24.32 |
| | H | **49.26** | 6.82 | **68.51** | **35.43** | **0.16** | **10.55** |

**6.3.2.5.2 Feature Correlation Criteria** Features here are roughly divided into two lists: correlated and uncorrelated lists. The two techniques defined above are also used here to select feature subsets for predictors. Table 6.15 gives a summary.

The pure technique, like the relevance criteria, did not enhance the results in any dataset. The hybrid technique, on the other hand, was more effective. It managed to produce better results in two dataset for *LSE*, in one for *ANN*, and in five dataset for *CART*.

Table 6.15: Summary for variation of *FSE* using feature correlation criteria. "P" refers to using the Pure technique, while "H" refers to the hybrid one

| | *FSE* | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|---|
| *LSE* | Basic | 38.37 | 4.34 | 38.58 | 33.59 | 4.33 | 40.46 |
| | P | 45.01 | 5.10 | 43.84 | 49.10 | 22.09 | 44.03 |
| | H | **38.11** | 4.35 | 39.42 | 51.19 | 4.49 | **39.26** |
| *ANN* | Basic | 40.81 | 4.82 | 24.68 | 43.78 | 0.28 | 25.15 |
| | P | 46.02 | 8.11 | 47.08 | 47.74 | 3.06 | 70.65 |
| | H | **38.91** | 5.06 | 40.99 | 59.16 | 1.30 | 64.45 |
| *CART* | Basic | 49.90 | 6.43 | 69.21 | 35.84 | 0.17 | 11.72 |
| | P | 54.92 | 8.49 | 69.94 | 36.25 | 0.76 | 24.30 |
| | H | **49.39** | 6.81 | **68.79** | **35.57** | **0.17** | **10.53** |

## 6.4 Summary

Feature Subset Ensembles aim at utilizing all the available features by partitioning them among the available predictors. We presented here an extensive comparison with other single-predictor and ensemble approaches using regression datasets for the first time. In addition, we investigated several suggested variations to the basic *FSE* algorithm, and new systematic partitioning methods. The results proved the effectiveness of the basic *FSE* and the success of the added variants in improving its performance even better.

A summary of all the results is provided here for convenience. Tables 6.17-6.19 brief the results for all the methods used. The naming of $FSEs$ is according to table 6.16. The results clearly show the success of the variations introduced into the basic *FSE*. For the *LSE* predictors, *FSE* variants outperformed the other techniques in six datasets, while *SFS* was the best in only one. In the six datasets in which *FSE* was superior, the embedded feature selection gave the best results in four , while sampling with replacement produced the best in one.

For the *ANN* predictors, boosting was the leader. It outperformed the others in three datasets, while *FSE* was the best in the other datasets. This comes as no surprise, because boosting was reported to outperform other ensemble methods. For the *CART* predictors, boosting again produced good results, but in only two of the datasets. *FSE* variants were the best in four datasets. The bagging within *FSE* proved very useful, as it produced those four best results. Again this seems intuitive, as training set subsampling works better for non-linear unstable predictors.

Table 6.16: $FSEs$ variations

| Name | Description |
|---|---|
| $FSE$-00 | Basic $FSE$ with random partitioning, sampling without replacement, and equal weighting |
| $FSE$-01 | $FSE$ with sampling with replacement |
| $FSE$-02 | $FSE$ with relevance weighting |
| $FSE$-03 | $FSE$ with training error weighting |
| $FSE$-04 | $FSE$ with bagging |
| $FSE$-05 | $FSE$ with boosting |
| $FSE$-06 | $FSE$ with $SFS$ |
| $FSE$-07 | $FSE$ with $SFFS$ |
| $FSE$-08 | $FSE$ with pure feature relevance criteria |
| $FSE$-09 | $FSE$ with hybrid feature relevance criteria |
| $FSE$-10 | $FSE$ with pure feature correlation criteria |
| $FSE$-11 | $FSE$ with hybrid feature correlation criteria |

Table 6.17: Summary for all methods using $LSE$ predictors

| Method | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|
| Single | 39.71 | 4.43 | 47.81 | 41.79 | 4.36 | 47.43 |
| $SFS$ | 38.01 | 4.35 | 42.50 | 42.98 | **3.77** | 40.87 |
| $SFFS$ | 38.05 | 4.35 | 43.46 | 42.98 | 3.77 | 41.41 |
| Bagging | 39.79 | 4.36 | 48.65 | 40.68 | 4.44 | 51.58 |
| Boosting | 51.46 | 5.57 | 53.72 | 88.96 | 4.68 | 43.45 |
| $FSE$-00 | 38.37 | 4.34 | 38.58 | 33.59 | 4.33 | 40.46 |
| $FSE$-01 | 41.03 | 4.89 | 38.47 | **31.40** | 13.09 | 40.05 |
| $FSE$-02 | 38.54 | 4.34 | 38.64 | 33.48 | 4.41 | 40.30 |
| $FSE$-03 | 38.42 | 4.37 | 38.50 | 33.99 | 4.35 | 40.51 |
| $FSE$-04 | 38.48 | 4.39 | 39.33 | 55.16 | 4.68 | 46.03 |
| $FSE$-05 | 51.60 | 5.52 | 43.30 | 100.20 | 4.79 | 43.17 |
| $FSE$-06 | **36.21** | **4.14** | 38.82 | 51.11 | 3.82 | **38.76** |
| $FSE$-07 | 36.57 | 4.32 | **37.54** | 53.83 | 3.79 | 39.13 |
| $FSE$-08 | 44.49 | 5.33 | 42.73 | 48.62 | 19.15 | 45.55 |
| $FSE$-09 | 38.13 | 4.34 | 39.52 | 52.24 | 3.78 | 39.05 |
| $FSE$-10 | 45.01 | 5.10 | 43.84 | 49.10 | 22.09 | 44.03 |
| $FSE$-11 | 38.11 | 4.35 | 39.42 | 51.19 | 4.49 | 39.26 |

Table 6.18: Summary for all methods using $ANN$ predictors

| Method | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|
| Single | 113.9 | 325.1 | 76.84 | 971.3 | 3.67 | 139.0 |
| $SFS$ | 181.9 | 15.13 | 65.89 | 92.48 | 0.40 | 11.66 |
| $SFFS$ | 64.07 | 12.86 | 83.14 | 103.4 | 0.72 | 18.96 |
| Bagging | 38.82 | 4.52 | 25.30 | 47.77 | 0.35 | 27.29 |
| Boosting | **38.78** | 4.21 | 25.20 | **24.03** | **0.18** | 676.2 |
| $FSE$-00 | 40.81 | 4.82 | **24.68** | 43.78 | 0.28 | 25.15 |
| $FSE$-01 | 42.51 | 6.38 | 26.48 | 28.30 | 0.97 | 24.71 |
| $FSE$-02 | 39.54 | 5.30 | 24.92 | 30.07 | 0.31 | 23.06 |
| $FSE$-03 | 39.97 | 5.71 | 24.76 | 28.73 | 0.27 | 23.46 |
| $FSE$-04 | 42.97 | 4.46 | 39.61 | 67.36 | 0.37 | 44.00 |
| $FSE$-05 | 41.84 | 4.77 | 40.13 | 72.98 | 0.37 | 74.61 |
| $FSE$-06 | 39.74 | **2.15** | 28.69 | 42.73 | 0.32 | 12.07 |
| $FSE$-07 | 41.73 | 4.53 | 27.56 | 38.16 | 0.41 | **10.03** |
| $FSE$-08 | 45.42 | 7.96 | 47.13 | 46.10 | 3.05 | 70.86 |
| $FSE$-09 | 39.84 | 4.54 | 41.20 | 51.66 | 0.63 | 64.06 |
| $FSE$-10 | 46.02 | 8.11 | 47.08 | 47.74 | 3.06 | 70.65 |
| $FSE$-11 | 38.91 | 5.06 | 40.99 | 59.16 | 1.30 | 64.45 |

Table 6.19: Summary for all methods using $CART$ predictors

| Method | bank32nh | ailerons | syn | house16h | comp | pole |
|---|---|---|---|---|---|---|
| Single | 75.98 | 8.71 | 130.5 | 59.18 | 0.30 | 14.57 |
| $SFS$ | 68.21 | 9.21 | 141.1 | 61.48 | 0.26 | 20.43 |
| $SFFS$ | 68.49 | 9.22 | 100.8 | 52.92 | 0.26 | 20.39 |
| Bagging | 43.32 | 5.20 | 67.08 | 37.07 | 0.17 | 9.20 |
| Boosting | 44.03 | 5.27 | **63.44** | 40.64 | 3.10 | **7.88** |
| $FSE$-00 | 49.90 | 6.43 | 69.21 | 35.84 | 0.17 | 11.72 |
| $FSE$-01 | 49.20 | 6.43 | 69.04 | 36.17 | 0.19 | 15.61 |
| $FSE$-02 | 49.30 | 6.39 | 69.61 | 35.57 | 0.17 | 11.61 |
| $FSE$-03 | 49.61 | 6.36 | 69.16 | 35.99 | 0.17 | 11.47 |
| $FSE$-04 | **42.13** | **5.09** | 66.17 | **34.47** | **0.16** | 10.98 |
| $FSE$-05 | 49.08 | 5.33 | 63.57 | 37.05 | 0.17 | 11.28 |
| $FSE$-06 | 49.77 | 7.56 | 85.38 | 38.89 | 0.17 | 12.67 |
| $FSE$-07 | 68.61 | 7.21 | 100.1 | 39.04 | **0.16** | 11.19 |
| $FSE$-08 | 54.42 | 8.51 | 70.25 | 36.10 | 0.79 | 24.32 |
| $FSE$-09 | 49.26 | 6.82 | 68.51 | 35.43 | 0.16 | 10.55 |
| $FSE$-10 | 54.92 | 8.49 | 69.94 | 36.25 | 0.76 | 24.30 |
| $FSE$-11 | 49.39 | 6.81 | 68.79 | 35.57 | 0.17 | 10.53 |

# Part II

# Video Traffic Forecasting

# Chapter 7

# Video Network Traffic Forecasting

## 7.1 Introduction

Multimedia network applications have gained much attention in the past years. Video-on-demand, teleconferencing, *IP* telephony, and online gaming are just some of its applications. Multimedia applications need special considerations when being transmitted over the network, due to its delay sensitive nature. Some of the applications, e.g. video-on-demand and remote streaming, can tolerate relatively high end-to-end delay to allow for edge buffering, a technique used to alleviate the possibility of delay jitter. However, other applications are real-time and can not withstand this delay, e.g. telephony and teleconferencing. Therefore, destination-side buffering can not be employed here. Other controls have to be used, namely Quality-of-Service *(QoS)* controls. They assign for each traffic type a certain bandwidth to provide proper delivery to the destination. Nevertheless, they need to know, even roughly, the amount of bandwidth that will be required for the multimedia application. Hence, a robust solution for estimating multimedia traffic beforehand has to be created.

A major fraction of network traffic involves multimedia applications. Many research papers considered the problem of predicting network traffic. Most of the multimedia traffic comprises video traffic. Video traffic is encoded in many formats to convert the huge constant bit rate (*CBR*) traffic into manageable variable bit rate (*VBR*) traffic that can be handled efficiently through the network. *MPEG* (Moving Picture Expert Group) standard is the most widespread compression and coding method for video streams. Hence, many techniques for predicting network traffic have been applied to the problem of predicting *MPEG*-coded video traffic.

## 7.2 MPEG Overview

MPEG is the video compression standard of choice in present applications. MPEG started out with MPEG-1 (1992), a standard that aimed to provide video coding for Video CD-ROMs (*VCDs*) at 1.5 Mbps. MPEG-2 (1995) was designed later to provide higher bit rates suitable for High Definition Television (HDTV) at up to 20 Mbps. Later, MPEG-4 (1999) was designed to provide higher quality at bit rates as low as 10 Kbps.

MPEG divides the video stream into a number of pictures, called frames. Two different types of compression methods are used to compress these frames. Intra-frame compression methods can be used to reduce spatial redundancy within the same frame. Inter-frame compression is employed to reduce the temporal redundancy between consecutive frames. These techniques are used to produce three different types of frames:

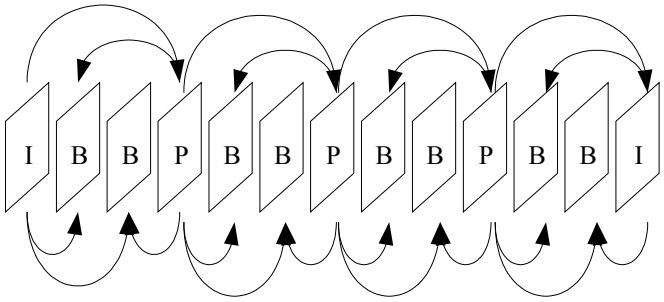
- ***I*-frames:** These are intra-frame compressed frames, where only information within the same frame is used. *I*-frames are created with techniques similar to JPEG-encoded images. They are considered key frames in the sequence, allow for random access in the stream, and act as the base for other compressed frames.

- ***P*-frames:** These are predictive frames, where inter-frame motion compensated compression is used. They depend on the previous *I*- or *P*-frame.

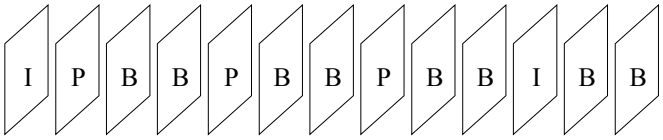
- ***B*-frames:** These are bi-directional frames, where bi-directional motion compensated inter-frame compression is used. The depend on both the previous and next *I*- or *P*-frames, however, they are not used in subsequent predictions. They provide the highest compression of all the frame types.

The coded frames are then packed into a deterministic periodic sequence, known as Group of Pictures *(GOP)*. The MPEG specifications do not define the exact order of the three frame types. However, a typical GOP frame is as shown in figure 7.1. The GOP in the figure consists of the sequence "IBBPBBPBBPBB". Each GOP is characterized by two quantities: $M$, the distance between two successive *I*-frames; and $N$, the distance between two successive *P*-frames. The GOP in the figure, for example, has $M = 12$ and $N = 3$. As shown, every B-frame depends on the previous and next *I*- or *P*-frame (as expressed by the arrows), and each *P*-frame depends on the previous *I*- or *P*-frame. The order of transmission is also

Figure 7.1: MPEG GOP sequence




(a) GOP logical structure




(b) GOP order of transmission

shown in the figure, where the *I*- and *P*-frames must be transmitted before their dependent *B*-frames.

Each of the sequences of *I*-, *B*-, and *P*-frames have special characteristics. *I*-frames exhibit the least compression ratios, however, they are the most important in the GOP, as the remaining frames rely on it. On the other hand, *P*- and *B*-frames show more compression, but they are less important in the GOP. In addition, they have different statistical properties, which are used for their prediction. Therefore, when predicting MPEG-coded video, each type of frame is handled separately.

## 7.3 Literature Survey

Most of the research achieved in multimedia traffic forecasting can be classified into two broad categories. The first deals with the development of stochastic source models [6, 11, 45, 54, 64]. These approaches model video sources using Hidden Markov chains, statistical techniques, and correlations functions [41]. The

second category involves empirical modeling and prediction approaches. There is not much work done in this category. Many of the proposed empirical models are linear. For example, Adas [2, 3] proposes two linear models: a Wiener-Hopf model and a Normalized Least Mean Square (*NLMS*) adaptive model. Yoo [96] develops an adaptive traffic prediction scheme for VBR MPEG video sources that focuses on predicting the effects of scene changes. Both Adas's and Yoo's approaches are adaptive, in the sense that they continually update the predictor weight coefficients with time. Chodorek and Chodorek [24] develop a linear predictor for MPEG-coded video traffic based on partitioning of the phase-space into sub-regions.

Recognizing that nonlinearities in the traffic dynamics are present, a number of researchers considered models such as neural networks. Chang and Hu [20] implemented a model called Pipelined Recurrent Neural Network (*PRNN*) for the adaptive traffic prediction of MPEG video signals over ATM networks. Doulamis et al. [33, 34] investigate the application of neural networks for nonlinear traffic prediction of VBR MPEG-coded video sources. In a recent work by Doulamis et al. [35], the authors propose an adaptable neural network architecture that can operate off-line as well as on-line. In addition, in a recent paper, Bhattacharya et al. [9] proposed a feed-forward and a recurrent neural network model (also see [81]) for the multi-step-ahead prediction (MSP) of MPEG4-coded video source traffic.

## 7.4 Summary

Multimedia applications are among the most important applications on computer networks nowadays. Of these applications, real time applications are the most crucial. They can not tolerate large delays during transmission. Therefore, strict *QoS* measures are applied to guarantee smooth transfer of data. These measures require accurate prediction for the traffic flow expected out of such applications.

# Chapter 8

# Sparse Basis Selection Approach to Video Traffic Forecasting

## 8.1 Introduction

Atiya [5] proposed a novel algorithm for adaptive learnng using Sparse Basis Selection. We applied this algorithm to adaptive video traffic forecasting. The main feature of the algorithm is its adpatability. It can adapt to changing video characteristics, such as changing scene types, or when a new video is presented. The algorithm proposed is a novel technology developed in the signal processing comunity, called Sparse Basis Selection [22, 25, 52, 62, 75, 68, 94]. It is based on having a large dictionary of basis signals. Assuming that the basis is combined linearly, a selection algorithm chooses the most effective small set of basis signals. This technology has almost been confined to the signal processing community, and has applications in signal representation and data compression. The machine learning community has started exploring these techniques in the last few years, including exploring its relationship with support vector machines [83, 92, 73, 62].

## 8.2 Sparse Basis Selection

Consider that we would like to represent a signal or a group of signals linearly in terms of a number of basis functions. The traditional approach has been to consider a fixed family of basis functions of varying parameters (such as sinusoids with varying frequencies, or Gaussian functions of varying centers). It would be desirable to have the signal represented by as small set of basis functions as

possible. In the sparse basis selection problem we consider a very large dictionary of possible basis functions (to have a large choice). For example, we can have a number of functions, such as sinusoids, Gaussians, sinc functions, wavelets, etc. From these we select a set of basis functions as small as possible that can represent the given signal(s) well.

This topic has generated a lot of research activity since the mid nineties. The main contributions in the early phase of developing the idea can be found in [25, 68, 75, 21]. Most of the work has focused on developing efficient algorithms for selecting the best basis functions from among the dictionary functions. Because of the combinatorial nature of the problem, it has been shown to be *NP*-hard, as demonstrated by Natarajan [75]. Most developed methods produce suboptimal solutions. A group of methods is based on first selecting the best vector, then the vector that together with the previously selected produces the best set and so on. Such approach is termed *the forward greedy selection approach*. Another approach, *the backward greedy selection approach*, starts with the full dictionary, and sequentially eliminates one vector after the other [26].

The problem with the forward selection approach is that once a vector is chosen, we are stuck with it and it can not be removed. On the other hand, backward elimination algorithms are computationaly expensive. A good alternative would be a *forward-backward* greedy approach [73]. In this method, a random subset of vectors is intilly chosen. Then, at each step, the worst vector is removed from this subset and the best vector from the remaining pool is added to the subset. Atiya followed this approach and devised a computationally efficient technique to perform the forward-backward elimination step.

Despite the fact that sparse basis selection produces a linear combination of the chosen vector subset, it can be used to produce non-linear prediciton models. This can be accomplished through employing non-linear transformations of the inputs as additional inputs. A number of inputs are extracted from the original time series, such as differences, second derivatives, moving averages, absolute differences, ...etc. Then, non-linear functions can be applied to these inputs, such as logarithms, sigmoids, square roots, exponentials, ...etc. By forming a linear combination of these non-linear transformations of the inputs, we are effectively producing a non-linear model.

## 8.3 Adaptive Sparse Basis Selection

Consider a pool of $K$ possible inputs. Let $x(i) \in \mathcal{R}^{1\times K}$ and $y(i) \in \mathcal{R}$ represent respectively the training set input vectors and target outputs (the values to be forecasted), where $i$ indexes time. Let the size of the training set be $N$. Let us arrange the input vectors in a matrix $X \in \mathcal{R}^{N\times K}$ with the rows being $x(i)$, and let us also arrange the target outputs $y(i)$ in a column vector $y$. In the sparse basis selection problem the goal is to find a small number $J << K$ of effective inputs that lead to minimum error. Let $S \subset \{1, 2, \ldots, K\}$ denote the set of selected inputs and let $\bar{S} \equiv \{1, 2, \ldots, K\} - S$ represent the remaining inputs that were not selected. The developed algorithm is a forward/backward approach. The main steps are given as

- Initialize with any $J$ randomly selected inputs $S$.
- Remove the element of $S$ that increases the resulting error the least amount.
- Add the element from $\bar{S}$ to $S$ that reduces the resulting error the most.
- Repeat the past two steps until convergence i.e until the removed vector is the same as the added vector.

The algorithm proposes a faster approach to update the currently selected vectors and their weights as new data points arrive. It is an adaptive algorithm, in that as new values arrive they increase the accuracy of the current model in a clever way without having to re-calculate everything from scratch. A detailed description of the algorithm follows.

### 8.3.1 Preliminaries

Consider a forward approach, at any step assume we have selected a number of inputs (corresponding to the columns $X_S$) with $|S| < J$, and we would like to select a vector from $\bar{S}$ to add to $S$. The least square error solution for the weight vector is given as

$$w = ({X_S}^T X_S)^{-1} {X_S}^T y \tag{8.1}$$

and the corresponding error is

$$E = y^T \mathcal{P}^{\perp}_{X_S} y = y^T (I - X_S({X_S}^T X_S)^{-1} {X_S}^T) y \tag{8.2}$$

where the matrix $\mathcal{P}^{\perp}_{X_S}$ represents the projection on the null space of $X_S$. An efficient way to update the solution vector and compare the contribution of each vector in $\bar{S}$ to determine which one to be added to $S$, is to consider the projection of $X_{\bar{S}}$ onto the null space of $X_S$, defined as

$$Z = \mathcal{P}^{\perp}_{X_S} X_{\bar{S}} \tag{8.3}$$

It can be shown that the best element (in terms of minimizing error) to move from $\bar{S}$ to $S$ is the one corresponding to column $z$ of $Z$ that achieves maximum

$$\kappa(v) = \frac{|z^T y|^2}{\|z\|^2} \tag{8.4}$$

A proof will be given in Section 8.3.2.1. Natarajan [75] derived a clever algorithm to update the projected matrix $Z$ every forward selection step to reflect the added column in $X_S$.

### 8.3.2 The Forward Update

Assume that at time $n-1$ we already have a selected set of inputs $S$. Also, assume that new measurements are now available at time $n$, and that we would like to update the set $S$ and its corresponding weight vector $w$. To simplify the notation, let

$$U \equiv X_S \tag{8.5}$$

$$V \equiv X_{\bar{S}} \tag{8.6}$$

To make the method adaptive and have the recent data have a larger influence on the selected vectors, we use a discounted error function:

$$E(n) = \sum_{i=1}^{n} \alpha^{n-i} [u^T(i) w - y(i)]^2 \tag{8.7}$$

where $u^T(i)$ is the $i^{th}$ row of $U$, and $\alpha$ is the discount weight. It is in the range from 0 to 1 (usually close to 1).
Let $R$ denote the discount matrix, being an $n \times n$ diagonal with $R_{ii} = \alpha^{n-i}$ The projection matrix is given by

$$\mathcal{P}_S = RU(U^T RU)^{-1} U^T R \tag{8.8}$$

and the minimum error is given by

$$E(n) \equiv y^T \mathcal{P}_S^{\perp} y = y^T \Big[ R - RU(U^T RU)^{-1} U^T R \Big] y \tag{8.9}$$

These formulas are given in a straightforward manner by adapting least square theory to the case of a discounted error function.

When new input/output measurements arrive at time $n$, two tasks have to be performed. The first task is to examine each of the unselected vectors (the columns of $V$), and find the one vector leading to maximum reduction in error. This, of course, will be done in a computationally efficient recursive manner. The second task, once determinining the best vector from $V$ to add to $U$, is to update the relevant matrices and variables to reflect this new addition. The algorithms solving both tasks are described in the next two subsections:

#### 8.3.2.1 Determining Best Column Vector to Add

The main approach we follow is to compute the formula 8.4 for each of the columns $v$ of $V$ in a recursive manner. This will be described here. Let us compute the new error at time $n$ as a function of the error and other variables that are available at time $n-1$. Define

$$D(n-1) = (U^T RU)^{-1} \tag{8.10}$$

Upon moving forward in time, and getting a new training data point, each of the selected basis vectors (the columns of $U$) gets augmented by a new element. Denote the new $U$ matrix as

$$U_{\text{new}} = \begin{pmatrix} U \\ u^T(n) \end{pmatrix} \tag{8.11}$$

with $U$ being the old one of time $n-1$, and $u^T(n)$ being the new added row in $U$. The discount matrix of time $n$ can be shown to be given by

$$R_{\text{new}} = \begin{pmatrix} \alpha R & 0 \\ 0^T & I \end{pmatrix} \tag{8.12}$$

where $R$ is the old discount matrix and $0 = (0 \;\; 0 \;\; \ldots \;\; 0)^T$. The new $D(n)$ can be written as

$$D(n) \equiv (U_{\text{new}}^T R_{\text{new}} U_{\text{new}})^{-1} = \Big( \alpha U^T RU + u(n) u^T(n) \Big)^{-1} \tag{8.13}$$

Using the small-rank adjustment inversion formula [57], we get

$$D(n) = \frac{1}{\alpha}(U^T RU)^{-1} - \frac{\frac{1}{\alpha^2}(U^T RU)^{-1}u(n)u^T(n)(U^T RU)^{-1}}{1 + u^T(n)(U^T RU)^{-1}u(n)/\alpha} \tag{8.14}$$

For abbreviation, let

$$b(n) = (U^T RU)^{-1}u(n) \tag{8.15}$$

Then

$$D(n) = \frac{1}{\alpha}D(n-1) - \frac{b(n)b^T(n)}{\alpha^2 + \alpha b^T(n)u(n)} \tag{8.16}$$

The $D(n)$ is a key variable in the adaptive update. The other two key quantities are:

$$C(n) = X^T RX \tag{8.17}$$

$$d(n) = X^T Ry \tag{8.18}$$

(remember $X$ is the matrix consisting of all vectors including selected and unselected ones). Upon moving forward in time, all these quantities get updated in a straightforward manner as follows:

$$C(n) = \alpha C(n-1) + x(n)x^T(n) \tag{8.19}$$

$$d(n) = \alpha d(n-1) + y(n)x(n) \tag{8.20}$$

where $x^T(n)$ is the new row added to matrix $X$ corresponding to time $n$ data point.

These matrices will serve multiple purposes. To be able to extract components of these, define $C_{QZ}(n)$ as the submatrix of $C(n)$ consisting of rows indexed by set $Q$ and columns indexed by set $Z$. Of course, each of $Q$ or $Z$ can be a single number. An analogous definition will apply to the vector $d_Q(n)$, and generally this convention will be used for any matrix or vector.

Assume that we consider adding to the pool of selected inputs an input $i_v$ from $\bar{S}$ (corresponding to column of $v$ of matrix $V$) We can rewrite Eq. 8.4 as:

$$p(v) \equiv \frac{v^T \mathcal{P}_{\bar{\mathcal{S}}}^{\perp} y}{v^T \mathcal{P}_{\bar{\mathcal{S}}}^{\perp} v} \tag{8.21}$$

(noting that $(\mathcal{P}_{\mathcal{S}}^{\perp})^2 = \mathcal{P}_{\mathcal{S}}^{\perp}$), and we also have

$$E'(n) = E(n) - p(v) \tag{8.22}$$

where $E(n)$ and $E'(n)$ are respectively the error functions before and after adding the new column. We get

$$p(v) = \frac{v^T R y - v^T R U D(n) U^T R y}{v^T R v - v^T R U D(n) U^T R v} \tag{8.23}$$

We solve for (8.23) in a computationally efficient manner, as follows. First we update the $C(n)$ and $d(n)$ matrices with the new data point at $n$ as in (8.44) and (8.45). Then we compute:

$$A_1 = D(n) C_{Si_v} \tag{8.24}$$

$$A_2 = D(n) d_S(n) \tag{8.25}$$

We get

$$p(v) = \frac{d_{i_v}(n) - C_{Si_v}^T A_2}{g(v)} \tag{8.26}$$

where

$$g(v) = C_{i_v i_v}(n) - C_{Si_v}^T A_1 \tag{8.27}$$

One can see that the above method avoided any computations of order $N$. The largest computation is of order $K^2$ (in the update of Eq. 8.44), even after computing $p(v)$ for all possible vectors $v$. Another note is that one can use the fact that $D(n)$ is a rank-one adjustment of the previous $D$, whether $D$ of the previous data point $n-1$ 8.14, or $D$ resulting from a column addition (Eq. 8.32 next Subsection) or deletion (Eq. 8.38 next Section). This can make the computations in (8.47) and (8.25) of order $J(K-J)$ rather than $J^2(K-J)$ (with cycling over all $v$'s). We will not go into the details of this technique, to avoid being too much sidetracked into algorithmic details.

Once $p(v)$ is computed for all $v$, then the vector $v$ leading to minimum $p(v)$ will be chosen to be added to the set $S$.

#### 8.3.2.2 Updating the Matrices

Assume column $v$ of matrix $V$ delivered the lowest resulting error. It is now required to augment the set of selected columns $U$ with the new addition $v$, and recompute all necessary matrices and quantities. Let $U'$ be the new augmented $U$ matrix. Thus,

$$U' = \begin{pmatrix} U & v \end{pmatrix} \tag{8.28}$$

Then, the matrix $D(n)$ can be updated recursively, as will be shown next. Let $D'(n)$ be the updated matrix. It is given by

$$D'(n) = \begin{pmatrix} U^T RU & U^T RV \\ v^T RU & v^T Rv \end{pmatrix} \tag{8.29}$$

By the block-partitioned matrix inversion formula, we get

$$D'(n) = \begin{pmatrix} (U^T RU - U^T Rvv^T RU)^{-1} & -\frac{(U^T RU)^{-1} - U^T Rv}{c} \\ -\frac{v^T RU(U^T RU)^{-1}}{c} & \frac{1}{c} \end{pmatrix} \tag{8.30}$$

where

$$c = v^T Rv - v^T RU(U^T RU)^{-1} U^T Rv \tag{8.31}$$

The upper left submatrix in the above matrix can be evaluated using the small-rank adjustment inversion formula. We get

$$D'(n) = \begin{pmatrix} (U^T RU)^{-1} + \frac{(U^T RU - U^T Rvv^T RU)^{-1}}{c} & -\frac{(U^T RU)^{-1} - U^T Rv}{c} \\ -\frac{v^T RU(U^T RU)^{-1}}{c} & \frac{1}{c} \end{pmatrix} \tag{8.32}$$

All quantities in the above formula are available as they have been computed in the previous subsection. We get:

$$D'(n) = \begin{pmatrix} D(n) + \frac{A_1 A_1^T}{g(v)} & \frac{A_1}{g(v)} \\ \frac{A_1^T}{g(v)} & \frac{1}{g(v)} \end{pmatrix} \tag{8.33}$$

where every $A_1$ and $v$ in the formula pertain to the vector that was selected to move to chosen set.

### 8.3.3 The Backward Update

In the backward update we eliminate a column vector from the matrix of selected vectors $U$. By eliminating one vector at time, and checking the effect of each elimination on the error, one can determine the best vector to eliminate. To derive

the backward update recursively, we utilize some of Reeves's formulas [87] for the backward basis update. Reeves has developed one of the most efficient backward greedy algorithms. Reeves showed that upon eliminating column number $k$ in $U$, the error function becomes

$$E_k(n) = E(n) + \frac{\left|\left[(U^T RU)^{-1} U^T Ry\right]_k\right|^2}{\left[(U^T RU)^{-1}\right]_{kk}} \tag{8.34}$$

where $E_k(n)$ denotes the error after eliminating the $k^{th}$ column, $E(n)$ is the error before eliminating any column, the subscript indexing in the RHS denotes the vector or matrix row/column number (note that we have added the discounting effect to Reeves's formulas). From the previous section, one see that all the quantities needed here are readily available. For example

$$\left[(U^T RU)^{-1}\right]_{kk} = D_{kk}(n) \tag{8.35}$$

where $D_{kk}(n)$ is $k^{th}$ diagonal element of $D(n)$. Also,

$$\begin{aligned} \left[(U^T RU)^{-1} U^T Ry\right]_k &= \left[D(n) d_S(n)\right]_k \\ &= \left[A_2\right]_k \end{aligned} \tag{8.36}$$

We have to compute $E_k(n)$ for each $k = 1, \ldots, J$, and obtain the $k$ that results in minimum $E_k(n)$. Once decided which column $k$ to eliminate, the matrices and quantities used can be updated in a straightforward manner. We will follow closely Reeves's derivation, adjusting it when needed to reflect the discounting. Let $D(n)$ and $D'(n)$ denote respectively the old value and the updated value of $(U^T RU)^{-1}$. Let us partition $D(n)$ as follows

$$D(n) = \Pi \begin{pmatrix} G_k & g_k \\ g_k^T & D_{kk}(n) \end{pmatrix} \Pi^T \tag{8.37}$$

where $\Pi$ represents a permutation matrix that moves $k^{th}$ column and $k^{th}$ row to the last column and row respectively. The column vector $g_k \in \mathcal{R}^{(J-1)\times 1}$ represents the $k^{th}$ column of $D(n)$ with the $k^{th}$ element of that column excluded (that is without the diagonal element $D_{kk}(n)$). Using the block matrix inversion formula we get

$$D'(n) = G_k - \frac{g_k g_k^T}{D_{kk}(n)} \tag{8.38}$$

### 8.3.4 A Summary of the Algorithm

The algorithm with its forward update and backward update components can be summarized in the following steps:

1. *Initialize:* Start at time $n = n_0$. Choose the set $S$ randomly as any $J$ numbers in $\{1, \ldots, K\}$. Let $\bar{S} = \{1, \ldots, K\} - S$.

2. Compute initial variables:

$$D(n) = (X_S^T R X_S)^{-1} \tag{8.39}$$

$$C(n) = X^T R X \tag{8.40}$$

$$d(n) = X^T R y \tag{8.41}$$

3. For $n = n_0 + 1$ to $N$ do:

   (a) Update the matrices:

$$D(n) = \frac{1}{\alpha} D(n-1) - \frac{b(n) b^T(n)}{\alpha^2 + \alpha u^T(n) b(n)} \tag{8.42}$$

   where

$$b(n) = D(n-1) u(n) \tag{8.43}$$

$$C(n) = \alpha C(n-1) + x(n) x^T(n) \tag{8.44}$$

$$d(n) = \alpha d(n-1) + y(n) x(n) \tag{8.45}$$

   (b) For $I$ cycles do Steps c) then d):

   (c) *Backward recursion:* Compute

$$\hat{k} = \text{argmax}_k \frac{\left[D(n) d_S(n)\right]_k}{D_{kk}(n)} \qquad k \in S \tag{8.46}$$

   Remove $\hat{k}$ from $S$: $S \equiv S - \{\hat{k}\}$, $\bar{S} = \bar{S} \cup \{\hat{k}\}$. Update $D(n)$ according to Eq. 8.37 and 8.38.

(d) *Forward recursion:* For all $k \in \bar{S}$:

$$A_{1k} = D(n)C_{Sk} \tag{8.47}$$

$$A_2 = D(n)d_S(n) \tag{8.48}$$

$$g_k = C_{kk}(n) - C_{Sk}^T A_{1k} \tag{8.49}$$

$$\hat{k} = \text{argmax}_k \frac{d_k(n) - C_{Sk}^T A_2}{g_k} \tag{8.50}$$

Move $\hat{k}$ from $\bar{S}$ to $S$: $\bar{S} \equiv \bar{S} - \{\hat{k}\}, \quad S = S \cup \{\hat{k}\}$. Update $D(n)$ as follows:

$$D(n) \equiv= \begin{pmatrix} D(n) + A_{1\hat{k}} A_{1\hat{k}}^T / g_{\hat{k}} & A_{1\hat{k}}/g_{\hat{k}} \\ A_{1\hat{k}}^T / g_{\hat{k}} & 1/g_{\hat{k}} \end{pmatrix} \tag{8.51}$$

## 8.4 Experiments and Results

### 8.4.1 Video Streams

In our experiments with Atiya's algorithm, we used MPEG-4 coded video streams available from the Technical University of Berlin [42]. There is a choice of high/medium/low quality movie traces, and we chose the high quality versions. The traces provide the sizes (in bytes) of $I$-frames, $B$-frames, and $P$-frames. The traces were divided into three independent time series: $I$-frames, $B$-frames, and $P$-frames, with different characteristics. Each time series was predicted independently of other series.

### 8.4.2 Input Dictionaries

We used the following inputs for the I-frame prediction model (note that the numbering of the B-frames represents its order in the GOP):
1. first backward difference $b(t) = f(t) - f(t-1)$ where $b(t)$ is the input and $f(t)$ is the frame size
2. absoulute value of (1)
3. exponential of (1) divided by the mean of (2)
4. exponential of (2) divided by the mean of (2)
5. sigmoid of (1) appropriately scaled, where sigmoid is the function $1/(1+e^{-x})$

6. second backward difference $b(t) = f(t) - 2f(t-1) + f(t-2)$
7. spike-detecting input

$$b(t) = \begin{cases} 1 & \text{if } f(t) - f(t-1) > mean|f(t) - f(t-1)| \\ -1 & \text{if } f(t) - f(t-1) < -mean|f(t) - f(t-1)| \\ 0 & \text{otherwise} \end{cases}$$

8. another spike-detecting input

$$b(t) = \begin{cases} 1 & \text{if } f(t) - f(t-1) > std(f(t) - f(t-1)) \\ -1 & \text{if } f(t) - f(t-1) < -std(f(t) - f(t-1)) \\ 0 & \text{otherwise} \end{cases}$$

9. current frame minus the running average of past frames
10. a binarization of (9)
11-15. difference between two moving average windows, small window with sizes 1, 3, 5, 8, and 12 and large window with sizes 2, 9, 15, 24, and 36
16. mean of previous 12 frames (including I-, B- and P-frames)
17. mean of previous 2 B-frames
18. mean of previous 4 B-frames
19. mean of previous (B3, B4, B7 and B8) frames
20. mean of previous (B1, B2, B5 and B6) frames
21. mean of previous 8 B-frames
22. previous P-frame
23. mean of previous 3 P-frames
24. previous I-frame minus mean of 2 previous B-frames
25-48. logarithm of (1-24) appropriately scaled
49-72. exponential of (1-24) appropriately scaled
73-96. square root of (1-24) appropriately scaled
97-120. sigmoid of (1-24) appropriately scaled

The inputs for the P-frame model were as follows:
1-16. the same as for the I-frames above but applied on P-frames
17. previous P-frame minus mean of P-frames
18. difference between previous 2 P-frames
19. mean of previous 2 P-frames
20. mean of previous 2 I-frames
21. mean of previous 2 B-frames

22. mean of previous 4 B-frames
23. mean of previous 8 B-frames
24. previous I-frame minus mean of previous 2 B-frames
25-48. logarithm of (1-24) appropriately scaled
49-72. exponential of (1-24) appropriately scaled
73-96. square root of (1-24) appropriately scaled
97-120. sigmoid of (1-24) appropriately scaled

The inputs for the B-frame model were as follows:
1-16. the same as for the I-frames above but applied on B-frames
17. previous P-frame minus mean of P-frames
18. mean of previous 2 P-frames
19. mean of previous 3 P-frames
20. mean of previous (B1,B2) for B1 and B2 frame types, similar input for other B-frame types
21. mean of (B5,B6) for B1 and B2 frames and vice versa, mean of (B7,B8) for B3 and B4 frames and vice versa
22. previous I-frame minus mean of I-frames
23. mean of previous 2 B-frames
24. mean of previous 2 I-frames
25-48. logarithm of (1-24) appropriately scaled
49-72. exponential of (1-24) appropriately scaled
73-96. square root of (1-24) appropriately scaled
97-120. sigmoid of (1-24) appropriately scaled

### 8.4.3 Simulation Results

We used three of the traces: Aladdin, The Firm, and Star Wars IV as a kind of training set, to adjust some parameters of the algorithm, such as the adaption rate $\alpha$, the number of selected basis $J$, and the number of iterations for the backward forward algorithm $NITER$. Based on these runs we selected these parameters as follows: $alpha = 0.999,\ J = 5,\ NITER = 30$. We also used the training set to develop the pool of inputs. Some of the inputs are initially screened for predictive power using the training set, and are added to the pool if we felt they could be useful.

We compared the results to those of the neural network prediction model developed by Bhattacharya et al. [9]. We wish to note that if the algorithm is applied starting at time $n = 1$ we will encounter matrix ill-conditioning problems. The

reason is that in the outer product matrix $U^T RU$ becomes close to singular. To avoid that, we start at some initial time $n_0$ rather than time $n = 1$. In that case we will have to obtain the matrices initially at time $n_0$ by brute force rather than by the recursive approach (see the algorithm in Section 8.3.4 Eq. 8.39-8.41). Subsequently, we adapt them according to our recursive algorithm starting $n = n_0 + 1$ till the end of the trace (Step 3 in Section 8.3.4). For every video trace we used $n_0$ equals 500 for the I-frame time series, and equals 2000 for the P-frame and B-frame time series (the total size of the I-frame, P-frame, and B-frame time series for one trace is typically around 7000, 20000, 60000). It is always safer to have $n_0$ large enough. Because no prediction is performed in the interval 1 to $n_0$, we tried to have $n_0$ the smallest possible value that would still be sufficient to avoid the ill-conditioning problem. We wanted the excluded initial part as small as possible to have the comparison with [9] be as fair as possible.

We used a similar error measure as in [9], namely the normalized mean square error (it is essentially $SNR^{-1}$):

$$NMSE = \frac{\sum_{n=n_0}^{N} (y_n - \hat{y}_n)^2}{\sum_{n=n_0}^{N} y_n^2} \tag{8.52}$$

where $y_n$ is the actual time series value, $\hat{y}_n$ is the forecast, and $N$ is the length of the trace. We used the following traces as out-of-sample test data for the proposed approach: Die Hard III, Jurassic Park I, Lecture Room, Mr Bean, Silence of the Lambs, Simpsons, and Skiing [42]. Tables 8.1, 8.2, and 8.3 show the prediction results for respectively the I-frame model, the P-frame model, and the B-frame model, for all traces including the training and out of sample traces. Also included in the table are the results whenever available of [9], for comparison. Figures 8.1-8.3 show the prediction versus actual value for a portion of Die Hard III's I-frame, P-frame, and B-frame time series respectively.

### 8.4.4 Discussion of the Results

One can see that for most traces the proposed approach produces considerably better prediction results than the results in [9]. The improvement is more pronounced for the P-frame and B-frame models. The exception is Lecture Room's I-frame results, where the proposed approach gives 58.3% error. The reason is the ill-conditioning problem. It was under control for all other traces, but in Lecture Room apparently $n_0$ was not suffiient. In real forecasting situations we can use $n_0$ as large as we want, and therefore the ill-conditioning problem can be completely avoided.

Table 8.1: Prediction Performance for the I-Frame Model (in *NMSE*)

| **Trace** | **NMSE**(%) | [9] (%) |
|---|---|---|
| Aladdin* | **2.50** | 2.6 |
| Die Hard III | **1.89** | 2.9 |
| Jurassic Park I | **0.75** | 0.8 |
| Lecture Room | 58.3 | 0.2 |
| Mr Bean | 2.22 | - |
| Silence of the Lambs | **1.61** | 3.6 |
| Simpsons | 1.33 | - |
| Skiing | **1.12** | 2.0 |
| Starwars IV* | **1.27** | 1.5 |
| The Firm* | 1.28 | - |

* indicates traces used for training.

Table 8.2: Prediction Performance for the P-Frame Model (in *NMSE*)

| **Trace** | **NMSE**(%) | [9](%) |
|---|---|---|
| Aladdin* | **9.67** | 10.3 |
| Die Hard III | **4.10** | 9.0 |
| Jurassic Park I | **3.36** | 4.0 |
| Lecture Room | **1.36** | 6.9 |
| Mr Bean | 6.15 | - |
| Silence of the Lambs | **4.05** | 11.0 |
| Simpsons | 25.20 | - |
| Skiing | **2.33** | 6.2 |
| Starwars IV* | **9.05** | 9.2 |
| The Firm* | 7.71 | - |

* indicates traces used for training.

Table 8.3: Prediction Performance for the B-Frame Model (in *NMSE*)

| **Trace** | **NMSE**(%) | [9](%) |
|---|---|---|
| Aladdin* | 9.25 | 8.2 |
| Die Hard III | **1.51** | 4.0 |
| Jurassic Park I | **1.70** | 2.2 |
| Lecture Room | **0.72** | 28.3 |
| Mr Bean | 3.08 | - |
| Silence of the Lambs | **1.53** | 27.8 |
| Simpsons | 10.95 | - |
| Skiing | **1.10** | 14.7 |
| Starwars IV* | **2.97** | 3.5 |
| The Firm* | 1.23 | - |

* indicates traces used for training.

Figure 8.1: *I*-frame Prediction for Die Hard III

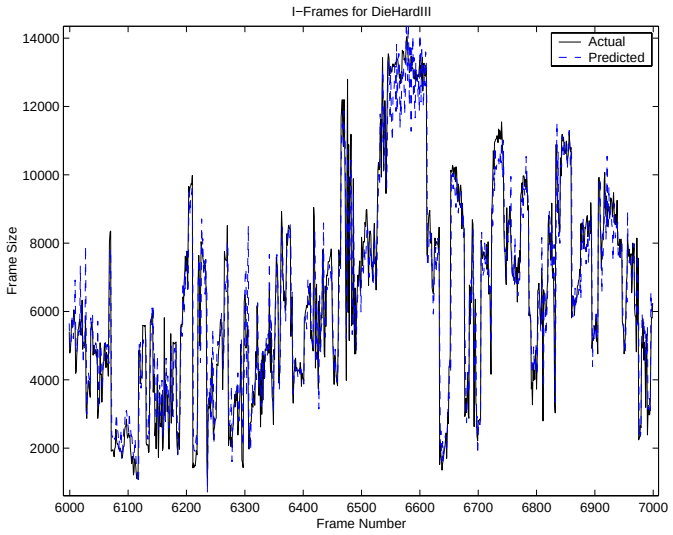


Figure 8.2: *B*-frame Prediction for Die Hard III

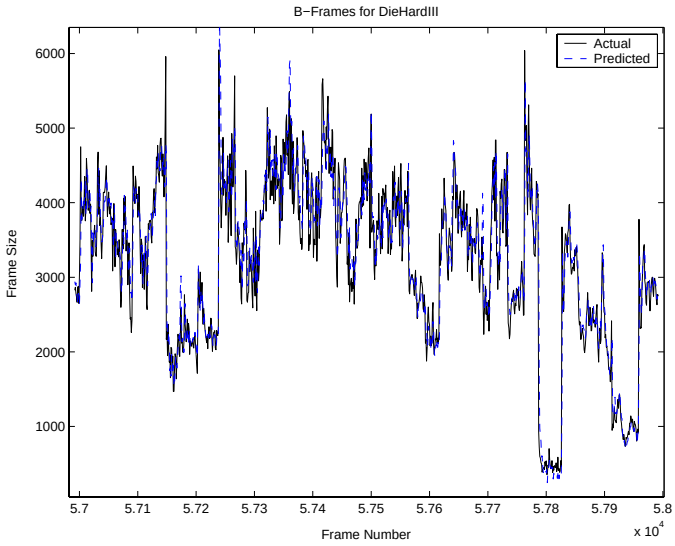
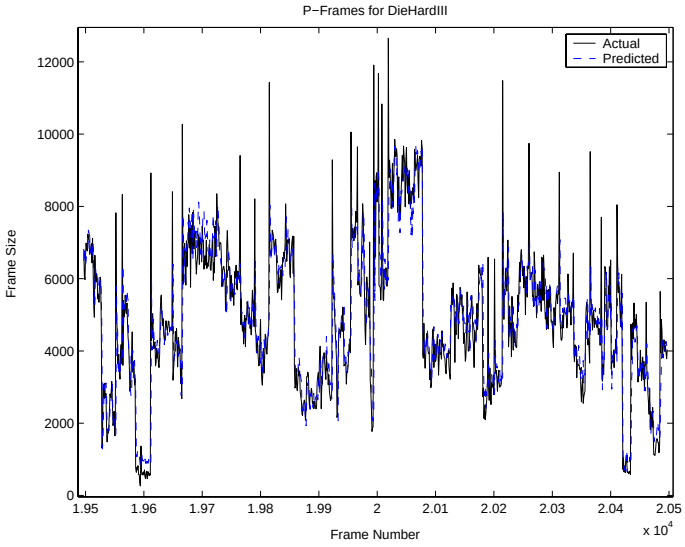



Figure 8.3: *P*-frame Prediction for Die Hard III



## 8.5 Summary

Sparse basis selection is a robust technique developed in the signal processing community, and was adapted recently to the machine learning arena. We used a novel sparse learning algorithm to adaptively predict video traffic. Adaptability is in both changing weights of bases selected, as well as the selected bases themselves. With the addition of each new data point, the set of selected bases is efficiently altered based on the new information available. This technique proved much successful and better than currently used prediction methods.

# Chapter 9

# Predictor Ensembles Approach to Video Traffic Forecasting

## 9.1 Introduction

Ensemble learning techniques exhibited superior performance in many calssification and regression tasks. However, it has not been tried in the problem of video traffic forecasting. We try to focus on the application of video traffic prediction using predictor ensembles. This is done by considering the video traffic problem as a normal regression task. We tried Artificial Neural Network ensembles, as they already showed good performance in video traffic prediction [9]. Three ensemble techniques were used: bagging, boosting, and basic $FSE$.

## 9.2 Experimental Setup

### 9.2.1 Datasets

Video traces from Technical University of Berlin [42] were employed in these experiments. We used a subset of the input dictionaries extracted in the experiments of Sparse Basis Selection. We discarded the non-linear transformations of the generated features, as the *ANN* is already a non-linear predictor and can adjust its weights to cope with the input features. Hence, each dataset is composed of 24 features. The features for the *I*-frames were as follows:
1. first backward difference $b(t) = f(t) - f(t-1)$ where $b(t)$ is the input and $f(t)$ is the frame size

2. absoulute value of (1)
3. exponential of (1) divided by the mean of (2)
4. exponential of (2) divided by the mean of (2)
5. sigmoid of (1) appropriately scaled, where sigmoid is the function $1/(1+e^{-x})$
6. second backward difference $b(t) = f(t) - 2f(t-1) + f(t-2)$
7. spike-detecting input

$$b(t) = \begin{cases} 1 & \text{if } f(t) - f(t-1) > mean|f(t) - f(t-1)| \\ -1 & \text{if } f(t) - f(t-1) < -mean|f(t) - f(t-1)| \\ 0 & \text{otherwise} \end{cases}$$

8. another spike-detecting input

$$b(t) = \begin{cases} 1 & \text{if } f(t) - f(t-1) > std(f(t) - f(t-1)) \\ -1 & \text{if } f(t) - f(t-1) < -std(f(t) - f(t-1)) \\ 0 & \text{otherwise} \end{cases}$$

9. current frame minus the running average of past frames
10. a binarization of (9)
11-15. difference between two moving average windows, small window with sizes 1, 3, 5, 8, and 12 and large window with sizes 2, 9, 15, 24, and 36
16. mean of previous 12 frames (including I-, B- and P-frames)
17. mean of previous 2 B-frames
18. mean of previous 4 B-frames
19. mean of previous (B3, B4, B7 and B8) frames
20. mean of previous (B1, B2, B5 and B6) frames
21. mean of previous 8 B-frames
22. previous P-frame
23. mean of previous 3 P-frames
24. previous I-frame minus mean of 2 previous B-frames

and for the *P*-frames:
1-16. the same as for the I-frames above but applied on P-frames
17. previous P-frame minus mean of P-frames
18. difference between previous 2 P-frames
19. mean of previous 2 P-frames
20. mean of previous 2 I-frames
21. mean of previous 2 B-frames

22. mean of previous 4 B-frames
23. mean of previous 8 B-frames
24. previous I-frame minus mean of previous 2 B-frames

and for the *B*-frames:
1-16. the same as for the I-frames above but applied on B-frames
17. previous P-frame minus mean of P-frames
18. mean of previous 2 P-frames
19. mean of previous 3 P-frames
20. mean of previous (B1,B2) for B1 and B2 frame types, similar input for other B-frame types
21. mean of (B5,B6) for B1 and B2 frames and vice versa, mean of (B7,B8) for B3 and B4 frames and vice versa
22. previous I-frame minus mean of I-frames
23. mean of previous 2 B-frames
24. mean of previous 2 I-frames.

We used the following movie traces: Aladdin, Die Hard III, Starwars IV, The Firm, Jurassic Park I, Lecture Room, Silence of The Lambs, Skiing, , Mr Bean, and Simpsons. The first four traces were used during training for adjusting the *ANN* parameters, and the remaining six were used as the test set. Each video trace was divided into three different sequences: $I$-, $B$-, and $P$-frames, and each was predicted individually. The typical sizes for these traces are 7000, 20000, and 60000 for the *I*-, *P*- and *B*-frames respectively. The training set size used was 2000, 3500 and 2500 for the *I*-, *P*- and *B*-frames respectively. The rest of each trace is used as the test set.

### 9.2.2 Predictor Setup

We used Artificial Neural Networks for these experiments. The type of $ANNs$ used was Recurrent Multilayer Perceptron (*RMLP*). They are like normal multilayer perceptron networks, with the exception that each neuron contains feedback connection from its output. This helps to add memory capabilities to the network. They are proved very powerful in modeling complex dynamic systems [81]. The number of input nodes are the number of features selected into each predictor, and the output layer consists of a single neuron. The hidden node contains 3 neurons. The networks were trained using the Levenberg-Marquardt backpropagation algorithm [46] with default training settings.

We experimented with three ensemble techniques in addition to the single pre-

dictor. The ensemble methods were: bagging, boosting, and basic $FSE$ i.e. $FSE$ with random feature subsets and equal weighting. For the ensemble techniques, we tried two ensemble sizes: 50 and 100. For the $FSE$, we tested for 4 feature subset sizes: 7, 12, 17, and 21. The performance measure used is also the $NMSE$. The reported results are the average of two independent runs on each video trace.

## 9.3 Results and Discussion

Tables 9.1-9.3 show the detailed prediction results for $I$-, $B$-, and $P$-frames for the used movie traces. A summary of the results is presented in tables 9.4-9.6 where the best result of each ensemble technique is presented. A comparison with the sparse basis method and Bhattacharya's approach [9] is presented in tables 9.7-9.9, where only the best ensemble approach is depicted for each trace.

The ensemble results show some points. First, the basic *FSE* was the superior, on the average, in the three ensemble techniques used. It was also better than the single predictor. *FSE* was the best in five datasets for the *I*-frames, in seven datasets for the *B*-frames, and in six datasets for the *P*-frames. It tied with bagging in one dataset for *I*-frames, and in three for the *P*-frames.

Second, the boosting results were worse than bagging for most of the video traces. Boosting may sometimes suffer from overfitting as a result of focusing on hard examples, which may happen to be noise. It seems that these video traces are highly unstable, and focusing on such hard patterns leads boosting ensembles to overfit the data. Third, bagging was the second best technique. It lead in four datasets for the *I*-frames, in two for *B*-frames, and in one for the *P*-frames.

The comparison with the sparse and Bhattacharya's techniques also show the success of the ensemble approach in video traffic prediction. For the *I*-frames, the ensemble techniques produced comparable, but generally worse, results, as it outperformed in only two video traces. For the *B*-frames, the ensemble approach was apparently more effective. It was the superior in seven traces and the second best in one. It was also sucessful for the *P*-frames, as it produced the best results in three traces and was the second best in another two.

Table 9.1: Results for $I$-frame prediction, where $K$ is the ensemble size and $n$ is the feature subset size

| | FSE | | | Bagging | | Boosting | |
|---|---|---|---|---|---|---|---|
| Dataset | $K$ | | | $K$ | | $K$ | |
| | $n$ | 50 | 100 | 50 | 100 | 50 | 100 |
| Aladdin | 7 | 5.47 | 5.50 | | **4.96** | **8.31** | 8.64 |
| | 12 | 5.09 | 5.02 | 5.05 | | | |
| | 17 | 4.96 | **4.91** | | | | |
| | 21 | 5.22 | 5.17 | | | | |
| Die Hard III | 7 | 6.34 | 6.52 | | **5.79** | **9.31** | 11.49 |
| | 12 | 5.53 | 5.55 | 6.01 | | | |
| | 17 | **5.32** | 5.37 | | | | |
| | 21 | 5.43 | 5.41 | | | | |
| Jurassic Park I | 7 | 1.93 | 1.84 | | **1.56** | **3.48** | 3.49 |
| | 12 | 1.65 | 1.65 | 1.57 | | | |
| | 17 | 1.58 | 1.58 | | | | |
| | 21 | 1.59 | **1.58** | | | | |
| Lecture Room | 7 | 0.16 | 0.16 | | 0.16 | **0.18** | 0.19 |
| | 12 | 0.16 | 0.16 | **0.15** | | | |
| | 17 | 0.16 | 0.16 | | | | |
| | 21 | 0.16 | **0.15** | | | | |
| Mr Bean | 7 | 6.39 | 6.39 | | 8.77 | **10.38** | 10.89 |
| | 12 | 6.75 | **6.36** | **8.35** | | | |
| | 17 | 8.44 | 8.10 | | | | |
| | 21 | 9.29 | 9.36 | | | | |
| Silence of The Lambs | 7 | 0.16 | 0.16 | | 0.17 | **2.19** | 2.39 |
| | 12 | 0.14 | 0.14 | **0.16** | | | |
| | 17 | 0.14 | 0.14 | | | | |
| | 21 | 0.14 | **0.14** | | | | |
| Simpsons | 7 | 2.90 | **2.88** | | 3.86 | 3.17 | **2.98** |
| | 12 | 2.95 | 2.99 | **3.82** | | | |
| | 17 | 3.28 | 3.14 | | | | |
| | 21 | 3.74 | 3.16 | | | | |
| Skiing | 7 | 4.77 | 4.82 | | **4.00** | **5.37** | 6.03 |
| | 12 | 4.19 | 4.14 | 4.09 | | | |
| | 17 | 4.05 | 4.04 | | | | |
| | 21 | **4.03** | 4.07 | | | | |
| Star Wars IV | 7 | 4.15 | 4.11 | | 3.25 | 12.34 | **11.88** |
| | 12 | 3.59 | 3.64 | **3.23** | | | |
| | 17 | 3.36 | 3.33 | | | | |
| | 21 | **3.29** | 3.31 | | | | |
| The Firm | 7 | 4.05 | 4.13 | | **3.31** | 11.38 | **11.17** |
| | 12 | 3.58 | 3.61 | 3.26 | | | |
| | 17 | 3.39 | 3.45 | | | | |
| | 21 | **3.39** | 3.40 | | | | |

Table 9.2: Results for $B$-frame prediction, where $K$ is the ensemble size and $n$ is the feature subset size

| | FSE | | | Bagging | | Boosting | |
|---|---|---|---|---|---|---|---|
| Dataset | $K$ | | | $K$ | | $K$ | |
| | $n$ | 50 | 100 | 50 | 100 | 50 | 100 |
| Aladdin | 7 | 7.22 | 7.05 | | | | |
| | 12 | 5.89 | 6.06 | 5.88 | **5.75** | **8.60** | 8.84 |
| | 17 | **5.44** | 5.56 | | | | |
| | 21 | 5.84 | 5.57 | | | | |
| Die Hard III | 7 | 1.17 | 1.16 | | | | |
| | 12 | 1.13 | **1.13** | 1.14 | **1.22** | 2.37 | **2.18** |
| | 17 | 1.14 | 1.16 | | | | |
| | 21 | 1.15 | 1.16 | | | | |
| Jurassic Park I | 7 | 7.59 | 7.46 | | | | |
| | 12 | **7.34** | 7.44 | **7.72** | 7.76 | **12.98** | 13.53 |
| | 17 | 7.57 | 7.61 | | | | |
| | 21 | 8.03 | 7.98 | | | | |
| Lecture Room | 7 | 0.10 | 0.10 | | | | |
| | 12 | 0.08 | 0.09 | 0.09 | **0.09** | **0.12** | 0.14 |
| | 17 | **0.08** | 0.09 | | | | |
| | 21 | 0.09 | 0.09 | | | | |
| Mr Bean | 7 | 5.80 | 5.83 | | | | |
| | 12 | 5.21 | 5.43 | 4.92 | **4.74** | 7.35 | **6.36** |
| | 17 | 5.20 | 5.28 | | | | |
| | 21 | **5.10** | 5.12 | | | | |
| Silence of The Lambs | 7 | 0.28 | 0.28 | | | | |
| | 12 | 0.27 | **0.26** | 0.31 | **0.30** | **1.85** | 1.98 |
| | 17 | 0.29 | 0.28 | | | | |
| | 21 | 0.31 | 0.29 | | | | |
| Simpsons | 7 | 2.95 | 2.99 | | | | |
| | 12 | 2.93 | **2.93** | **2.94** | 3.07 | **3.67** | 3.96 |
| | 17 | 2.93 | 2.97 | | | | |
| | 21 | 3.02 | 2.97 | | | | |
| Skiing | 7 | 1.93 | 1.92 | | | | |
| | 12 | 1.84 | 1.84 | 1.82 | **1.80** | **1.99** | 1.99 |
| | 17 | **1.81** | 1.82 | | | | |
| | 21 | 1.84 | 1.85 | | | | |
| Star Wars IV | 7 | 0.48 | **0.47** | | | | |
| | 12 | 0.50 | 0.53 | 1.28 | **1.07** | 1.03 | **0.97** |
| | 17 | 0.74 | 0.65 | | | | |
| | 21 | 0.88 | 0.82 | | | | |
| The Firm | 7 | 0.20 | **0.20** | | | | |
| | 12 | 0.22 | 0.23 | 0.26 | **0.25** | 0.33 | **0.30** |
| | 17 | 0.31 | 0.34 | | | | |
| | 21 | 0.40 | 0.45 | | | | |

Table 9.3: Results for $P$-frame prediction, where $K$ is the ensemble size and $n$ is the feature subset size

| | FSE | | | Bagging | | Boosting | |
|---|---|---|---|---|---|---|---|
| Dataset | $K$ | | | $K$ | | $K$ | |
| | $n$ | 50 | 100 | 50 | 100 | 50 | 100 |
| Aladdin | 7 | 11.60 | 11.54 | | **11.31** | **22.79** | 23.11 |
| | 12 | 11.35 | **11.31** | 11.35 | | | |
| | 17 | 11.41 | 11.43 | | | | |
| | 21 | 11.50 | 11.49 | | | | |
| Die Hard III | 7 | 5.77 | 5.76 | | 5.59 | **16.11** | 17.02 |
| | 12 | 5.63 | 5.61 | **5.59** | | | |
| | 17 | 5.59 | **5.58** | | | | |
| | 21 | 5.59 | 5.59 | | | | |
| Jurassic Park I | 7 | 4.54 | 4.54 | | **4.50** | **13.00** | 14.19 |
| | 12 | 4.45 | **4.45** | 4.51 | | | |
| | 17 | 4.48 | 4.48 | | | | |
| | 21 | 4.52 | 4.53 | | | | |
| Lecture Room | 7 | 0.46 | 0.46 | | 0.41 | **0.52** | 0.61 |
| | 12 | 0.43 | 0.43 | **0.41** | | | |
| | 17 | 0.42 | **0.41** | | | | |
| | 21 | 0.40 | 0.41 | | | | |
| Mr Bean | 7 | 8.81 | 8.79 | | **10.29** | **43.54** | 45.32 |
| | 12 | 9.17 | **9.13** | 10.95 | | | |
| | 17 | 9.44 | 9.52 | | | | |
| | 21 | 9.73 | 9.74 | | | | |
| Silence of The Lambs | 7 | 1.18 | 1.19 | | 1.12 | 9.83 | **9.49** |
| | 12 | 1.15 | 1.14 | **1.12** | | | |
| | 17 | 1.13 | 1.13 | | | | |
| | 21 | 1.12 | **1.12** | | | | |
| Simpsons | 7 | 5.79 | 5.76 | | 5.70 | 14.69 | **13.88** |
| | 12 | 5.63 | 5.64 | **5.68** | | | |
| | 17 | 5.64 | **5.63** | | | | |
| | 21 | 5.66 | 5.65 | | | | |
| Skiing | 7 | 7.44 | 7.43 | | **7.04** | **11.97** | 12.35 |
| | 12 | 7.22 | 7.18 | 7.05 | | | |
| | 17 | 7.12 | **7.08** | | | | |
| | 21 | 7.08 | 7.09 | | | | |
| Star Wars IV | 7 | 7.93 | 7.92 | | 7.95 | 166.4 | **155.9** |
| | 12 | **7.83** | 7.85 | **7.92** | | | |
| | 17 | 7.92 | 7.92 | | | | |
| | 21 | 7.98 | 7.97 | | | | |
| The Firm | 7 | 9.11 | 9.08 | | 8.96 | **58.42** | 66.93 |
| | 12 | 8.96 | 8.97 | **8.96** | | | |
| | 17 | 8.96 | **8.95** | | | | |
| | 21 | 8.96 | 8.97 | | | | |

Table 9.4: $I$-frame results summary

| | Algorithm | | | |
|---|---|---|---|---|
| Dataset | Single | $FSE$ | Bag | Boost |
| Aladdin | 5.75 | **4.91** | 4.96 | 8.31 |
| Die Hard II | 5.57 | **5.32** | 5.79 | 9.31 |
| Jurassic Park I | 1.66 | 1.58 | **1.56** | 3.48 |
| Lecture Room | 0.18 | **0.15** | **0.15** | 0.18 |
| Mr Bean | 10.34 | **6.36** | 8.35 | 10.38 |
| Silence of The Lambs | 0.16 | **0.14** | 0.16 | 2.19 |
| Simpsons | 4.43 | **2.88** | 3.82 | 2.98 |
| Skiing | 4.29 | 4.03 | **4.00** | 5.37 |
| Starwars IV | 3.40 | 3.29 | **3.23** | 11.88 |
| The Firm | 4.00 | 3.39 | **3.31** | 11.17 |

Table 9.5: $B$-frame results summary

| | Algorithm | | | |
|---|---|---|---|---|
| Dataset | Single | $FSE$ | Bag | Boost |
| Aladdin | 5.88 | **5.44** | 5.75 | 8.60 |
| Die Hard II | 1.18 | **1.13** | 1.22 | 2.18 |
| Jurassic Park I | 8.91 | **7.34** | 7.72 | 12.98 |
| Lecture Room | 0.11 | **0.08** | 0.09 | 0.12 |
| Mr Bean | 5.29 | 5.10 | **4.74** | 6.36 |
| Silence of The Lambs | 0.34 | **0.26** | 0.30 | 1.85 |
| Simpsons | **2.76** | 2.93 | 2.94 | 3.67 |
| Skiing | 1.88 | 1.81 | **1.80** | 1.99 |
| Starwars IV | 0.73 | **0.47** | 1.07 | 0.97 |
| The Firm | 1.02 | **0.20** | 0.25 | 0.30 |

Table 9.6: $P$-frame results summary

| | Algorithm | | | |
|---|---|---|---|---|
| Dataset | Single | $FSE$ | Bagg | Boost |
| Aladdin | 11.63 | **11.31** | **11.31** | 22.79 |
| Die Hard II | 5.63 | **5.58** | 5.59 | 16.11 |
| Jurassic Park I | 4.81 | **4.45** | 4.50 | 13.00 |
| Lecture Room | 0.44 | **0.41** | **0.41** | 0.52 |
| Mr Bean | 9.79 | **9.13** | 10.29 | 43.54 |
| Silence of The Lambs | 1.15 | **1.12** | **1.12** | 9.49 |
| Simpsons | 5.77 | **5.63** | 5.68 | 13.88 |
| Skiing | 7.37 | 7.08 | **7.04** | 11.97 |
| Starwars IV | 8.09 | **7.83** | 7.92 | 155.9 |
| The Firm | 8.98 | **8.95** | 8.96 | 58.42 |

Table 9.7: $I$-frame comparison with Sparse Basis Selection and [9]'s method. **Bold** entries are better than one method, and ***bold italic*** are better than the two methods.

| | Algorithm | | |
|---|---|---|---|
| Dataset | Ensemble | [9] | Sparse |
| Aladdin | 4.91 | 2.6 | 2.50 |
| Die Hard II | 5.32 | 2.9 | 1.89 |
| Jurassic Park I | 1.56 | 0.8 | 0.75 |
| Lecture Room | ***0.15*** | 0.2 | 58.3 |
| Mr Bean | 6.36 | - | 2.22 |
| Silence of The Lambs | ***0.14*** | 3.6 | 1.61 |
| Simpsons | 2.88 | - | 1.33 |
| Skiing | 4.00 | 2.0 | 1.12 |
| Starwars IV | 3.23 | 1.5 | 1.27 |
| The Firm | 3.31 | - | 1.28 |

Table 9.8: $B$-frame comparison with Sparse Basis Selection and [9]'s method. **Bold** entries are better than one method, and ***bold italic*** are better than the two methods.

| | Algorithm | | |
|---|---|---|---|
| Dataset | Ensemble | [9] | Sparse |
| Aladdin | ***5.44*** | 8.2 | 9.25 |
| Die Hard II | ***1.13*** | 4.0 | 1.51 |
| Jurassic Park I | 7.34 | 2.2 | 1.70 |
| Lecture Room | ***0.08*** | 28.3 | 0.72 |
| Mr Bean | 4.74 | - | 3.08 |
| Silence of The Lambs | ***0.26*** | 27.8 | 1.53 |
| Simpsons | ***2.93*** | - | 10.95 |
| Skiing | **1.80** | 14.7 | 1.10 |
| Starwars IV | ***0.47*** | 3.5 | 2.97 |
| The Firm | ***0.20*** | - | 1.23 |

Table 9.9: $P$-frame comparison with Sparse Basis Selection and [9]'s method. **Bold** entries are better than one method, and ***bold italic*** are better than the two methods.

| | Algorithm | | |
|---|---|---|---|
| Dataset | Ensemble | [9] | Sparse |
| Aladdin | 11.31 | 10.3 | 9.67 |
| Die Hard II | **5.58** | 9.0 | 4.10 |
| Jurassic Park I | 4.45 | 4.0 | 3.36 |
| Lecture Room | ***0.41*** | 6.9 | 1.36 |
| Mr Bean | 9.13 | - | 6.15 |
| Silence of The Lambs | ***1.12*** | 11.0 | 4.05 |
| Simpsons | **5.63** | - | 25.20 |
| Skiing | 7.04 | 6.2 | 2.33 |
| Starwars IV | ***7.83*** | 9.2 | 9.05 |
| The Firm | 8.95 | - | 7.71 |

## 9.4 Summary

Video traffic prediction is an important and hard problem. We tried to apply the ensemble technique, for the first time, to this application. We used basic *FSE*, bagging, and boosting. *FSE* managed to outperform the other techniques, bagging came second, and boosting was the worst. The produced results were encouraging. They were generally worse in *I*-frame prediction, but generally superior in *B*- and *P*-frame forecasting.

# Part III

# Conclusion

# Chapter 10

# Conclusions and Future Work

## 10.1 Introduction

This work studied the approach of ensemble learning in a regression context. In this method, the learning model is composed of a group of predictors instead of a single predictor. The individual predictors should both be diverse and accurate to provide enhanced performance. Many methods have been proposed to build such diverse, of which bagging and boosting are the most popular and successful. These two methods are based on subsampling the training set while giving all the features to all predictors. A new approach, Feature Subset Ensembles (*FSE*), that is based on subsampling the available features is extensively studied in this work. It is comprehensively compared to bagging, boosting, and sequential selection algorithms using six regression datasets. In addition, several effective variations are introduced to the basic *FSE* algorithm.

We further discussed an important problem in current network multimedia applications, namely video network traffic forecasting. It is of crucial importance to enhancing the quality of service of video transmission over the network. Two approaches were used to tackle that problem. The first is using a novel sparse basis selection algorithm to adaptively predict video traffic. The second is using the ensemble approach and treating the problem as a conventional regression task. The two approaches provided encouraging results and outperformed previously used methods.

## 10.2 Conclusions

This work provided a comparison of basic *FSE* with bagging, boosting, and sequential selection algorithms. The results showed the success of this technique, both with linear and non-linear predictors. The comparison also asserted the effectiveness of bagging and boosting as two powerful ensemble techniques, and that bagging can be sometime inferior to boosting but is more stable and resilient to overfitting the training data.

The variations added to the basic *FSE* were successful in enhancing its accuracy, which enabled it to outperform both bagging and boosting on the six used datasets. The embedded feature selection was particularly successful with linear *LSE* predictor, while bagging within *FSE* was helpful with non-linear *CART* trees.

The approach of sparse basis selection to video traffic prediction was also very successful. It applied a novel sparse learning algorithm that recursively updated the selected bases and their weights on the arrival of new data. It produced results superior to currently used techniques.

This work also applied the ensemble approach to video traffic forecasting. We tried basic *FSE*, bagging and boosting. The *FSE* produced the best results, followed by bagging and then boosting. This emphasized the effectiveness of *FSE* as a robust ensemble technique. The generated results were encouraging, and outperformed other techniques, including the sparse approach, in many video traces.

## 10.3 Contributions

This work introduced many valuable contributions in the area of ensemble learning as well as in the area of video traffic forecasting.

- It provides the first comprehensive comparison of bagging, boosting and *FSE* in the context of regression problems. The experiments used both linear and non-linear predictors and employed six datasets with different characteristics.
- It introdced many valuable variations to the basic FSE algorithm that managed to improve its performance for both linear and non-linear predictors.
- These experiments asserted the effectiveness of *FSE* and its variants in providing accurate ensembles that can outperform the well-known bagging and boosting.

- It introduced the use of sparse basis selection approach to the problem of video traffic forecasting. It employed a novel adaptive approach to solve this problem, that not only changes the weights of selected bases adaptively upon the arrival of new data points, but also changes the selected bases themselves.
- The sparse basis selection approach provided excellent results on ten different video traces and outperformed other available methods.
- It introduced the use of ensemble learning approach to the video traffic prediction. It used bagging, boosting, and basic FSE, and treated the problem as a regular regression job. *FSE*, again, was the superior ensemble technique.
- The ensemble approach achieved acceptable results, specially in the *B*- and *P*-frame prediction.

## 10.4 Future Work

Many interesting points arose from this work that deserve further investigation.

- Experimenting with more predictor types, and checking the relation of predictor overfitting to the ensemble performance.
- Adding more enhancements to *FSE* that can even improve its accuracy.
- Introducing an automated technique for choosing the best *FSE* parameters (size of ensemble and size of feature subset).
- Checking combinations of introduced enhancements, e.g. using training set subsampling with embedded feature selection.
- Enhancing the video traces dataset and improving the effectiveness of extracted features.
- Improving video traffic prediction accuracy using other, possibly more powerful, ensemble approaches or using different predictors.

# Bibliography


[1] Delve: Collections of data for developing, evaluating, and comparing learning methods, Computer Science Department, University of Toronto, http://www.cs.toronto.edu/∼delve/index.html, 2004.

[2] A. M. Adas. Supporting real-time VBR using dynamic reservation based on linear prediction. Technical Report GIT-CC-95/26, Georgia Institute of Technology, August 1995.

[3] Abdelnaser M. Adas. Using adaptive linear prediction to support real-time VBR video under RCBR network service model. *IEEE/ACM Transactions on Networks*, 6(5):635–644, 1998.

[4] Fuad M. Alkoot and Joseph Kittler. Feature selection for an ensemble of classifiers. In *Proceedings of the* $4^{th}$ *Multiconference on Systematics, Cybernatics, and Informatics*, pages 379–384, Orlando, Florida, 2000.

[5] Amir F. Atiya. New adaptive sparse basis selection algorithm. Technical report, Cairo University, Giza, Egypt, May 2004.

[6] J. J. Bae and T. Suda. Survey of traffic control schemes and protocols in ATM networks. In *Proceedings of the IEEE ...*, volume 79, pages 170–189, 1991.

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

[8] Stephen D. Bay. Combining nearest neighbor classifiers through multiple feature subsets. In *Proceedings of the* $17^{th}$ *International Conference on Machine Learning*, pages 37–45, Madison, WI, 1998.

[9] Aninda Bhattacharya, Alexander G. Parlos, and Amir F. Atiya. Prediction of MPEG-coded video source traffic using recurrent neural networks. *IEEE Transactions On Signal Processing*, 51(8):2177–2190, August 2003.

[10] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, University of California, Irvine, Dept. of Information and Computer Sciences, http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[11] P. Bocheck and S. F. Chang. A content based video traffic model using camera operations. In *Proceedings of IEEE International Conference on Image Processing*, pages 817–820, Lausanne, Switzerland, September 1996.

[12] P. Brazdil. Project StatLog, LIACC, University of Porto, http://www.liacc.up.pt/ML/statlog/datasets.html, 1999.

[13] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman and Hall, 1984.

[14] Leo Breiman. Bagging predictors. *Machine Learning*, 24:124–140, 1996.

[15] Leo Breiman. Using adaptive bagging to debias regression. Technical Report TR 547, University of California, Berkeley, California, USA, 1999.

[16] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[17] Gavin Brown, Jeremy Wyatt, Tachel Harris, and Xin Yao. Diversity creation methods: A survey and categorisation. *Pattern Recognition Society*, 2004.

[18] Robert Bryll, Ricardo Gutierrez, and Francis Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition Society*, 2002.

[19] M. Burl, U. Fayyad, P. Perona, and P. Smyth. Automating the hunt for volcanoes on venus. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 302–309, Seattle, WA, 1994. IEEE Computer Society Press.

[20] P. R. Chang and J. T. Hu. Optimal nonlinear adaptive prediction and modeling of MPEG video in ATM networks using pipelined recurrent neural networks. *IEEE Journal on Selected Areas in Communications*, 15:1087–1100, August 1997.

[21] K. Chen, L. Wang, and H. Chi. Method of combining multiple classifiers with different features and their applications to text-independent speaker identification. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(3):417–445, 1997.

[22] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1999.

[23] Kevin J. Cherkauer. Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. In *Proceedings of the* $13^{th}$ *National Conference on Artificial Intelligence*, pages 15–21, Portland, OR, 1996. AAAI Press.

[24] A. Chodorek and R.R. Chodorek. An MPEG-2 video traffic prediction based on phase space analysis and its application to on-line dynamic bandwidth allocation. In *Proceedings* $2^{nd}$ *European Conference on Universal Multiservice Networks*, pages 44–55, April 2002.

[25] R. R. Coifman and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38:713–718, 1992.

[26] S. Cotter, K. Kreutz-Delgado, and B. Rao. Backward elimination for sparse vector subset selection. *Signal Processing*, 81:1849–1864, 2001.

[27] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error correcting output codes. *Journal of Artificial Intelligence Research*, 39:1–38, 1995.

[28] Thomas G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18:97–136, 1997.

[29] Thomas G. Dietterich. An experimental comparison of three methods for conscructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, pages 1–22, 1999.

[30] Thomas G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

[31] W. Dillon and M. Goldstein. *Multivariate Analysis Methods and Applications*. John Wiley and Sons, 1984.

[32] J. Doak. An evaluation of feature selection methods and their application to computer security. Technical report, University of California, Davis, Computer Science Dept., California, USA, 1992.

[33] A. D. Doulamis, N. D. Doulamis, and S. D. Kollias. Nonlinear traffic modeling of VBR MPEG-2 video sources. In *Proceedings IEEE International Conference on Multimedia and Expo, ICME*, pages 1318–1321, July-August 2000.

[34] A. D. Doulamis, N. D. Doulamis, and S. D. Kollias. Recursive nonlinear models for on line traffic prediction of VBR MPEG coded video sources. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks IJCNN*, pages 114–119, July 2000.

[35] A. D. Doulamis, N. D. Doulamis, and S. D. Kollias. An adaptable neural network model for recursive nonlinear traffic prediction and modeling of MPEG video sources. *IEEE Transcations on Neural Networks*, 14(1):150–166, January 2003.

[36] Harris Drucker. Improving regressors using boosting techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 107–115. Morgan Kaufmann, 1997.

[37] Harris Drucker and Corina Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems*, volume 8, pages 470–485. MIT Press, 1996.

[38] B. Efron and R. Tibshirani, editors. *An Introduction to the Bootstrap*. Chapman and hall, 1993.

[39] L. Eshelman. *The CHC Adaptive Search Algorithm. How to have safe search when engaging in nontraditional genetic recombination*, pages 265–283. Morgan Kaufmann, 1991.

[40] B. Everitt. *Cluster Analysis*. Heinemann Educational Books, 1974.

[41] Chong Feng, Dan He, and Zhili Sun. A survey on network traffic forecasting. In *??*, pages ?–?

[42] F. Fitzek and M. Reisselein. Mpeg–4 and h.263 video traces for network performance evaluation, Technical University of Berlin, http://www-tkn.ee.tu-berlin.de/research/trace/trace.html, 1998.

[43] Yoav Freund and Robert Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the* $2^{nd}$ *European Conference on Computational Learning Theory*, pages 23–37, San Francisco, CA, 1995. Springer-Verlag.

[44] Yoav Freund and Robert Shapire. Experiments with a new boosting algorithm. In *Proceedings of the* $13^{th}$ *International Conference on Machine Learning*, pages 148–156, San Francisco, CA, 1996. Morgan Kaufmann.

[45] V. S. Frost and B. Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, pages 70–81, March 1994.

[46] P. R. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. London Academic Press, 1981.

[47] César Guerra-Salcedo and Darrell Whitley. Genetic search for feature subset selection: A comparison between CHC and GENESIS. In *Proceedings of the third annual Genetic Programming Conference*. Morgan Kaufmann, 1998.

[48] César Guerra-Salcedo and Darrell Whitley. Genetic approach to feature selection for ensemble creation. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 236–243, 1999.

[49] Simon Günter and Horst Bunke. Creation of classifier ensembles for handwritten word recognition using feature selection algorithms. In *Proceedings of the* $8^{th}$ *IWFHR*, pages 183–188, Niagara-on-the-lake, Canada, 2002.

[50] L. Hall, K. Bowyer, R. Banfield, D. Bhadoria, W. Kegelmeyer, and S. Eschrich. Comparing pure parallel ensemble creation techniques against bagging. In *The Third IEEE International Conference on Data Minin*, pages 533–536, 2003.

[51] L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transaction On Pattern Analysis And Machine Intelligence*, 12:993–1001, 1990.

[52] G. Harikumar, C. Couvreur, , and Y. Bresler. Fast optimal and suboptimal algorithms for sparse solutions to linear inverse problems.

[53] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

[54] D. P. Heyman and T. V. Lakshman. Source models for VBR broadcast-video traffic. *IEEE/ACM Transactions on Networking*, 4:40–48, February 1996.

[55] T. K. Ho. Random decision forests. In *Proc. of the 3rd Int'l Conference on Document Analysis and Recognition*, pages 278–282, Montreal, Canada, August 1995.

[56] Tin Kam Ho. The random subspace method for cosntructing decision forests. *IEEE Transaction On Pattern Analysis And Machine Intelligence*, 20(8):832–844, August 1998.

[57] R. Horn and C. Johnson, editors. *Matrix Analysis*. Camridge University Press, Cambridge, MA, 1985.

[58] Anil Jain and Douglas Zongker. Feature selection: Evaluation, applicaiton, and small sample performance. *IEEE Transaction On Pattern Analysis And Machine Intelligence*, 19(2):153–158, 1997.

[59] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the* $11^{th}$ *International Conference on Machine Learning*, pages 121–129, San Francisco, CA, 1994. Morgan Kaufmann.

[60] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[61] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[62] K. Kreutz-Delgado, J. Murray, B. Rao, K. Engan, T.-W. Lee, and T. Sejnowski. Dictionary learning algorithms for sparse representation. *Neural Computation*, 15(2):349–396, 2003.

[63] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 231–238, Denver, CO, 1995. MIT Press.

[64] M. Krunz and H. Hughes. A traffic model for MPEG-coded VBR streams. In *Proceedings of Joint International Conference on Measurement and*

*Modeling of Computer Systems, ACM SIGMETRICS*, pages 47–55, May 1995.

[65] Yuansong Liao and John Moody. Constructing heterogeneuos committees using input feature grouping: Application to economic forecasting. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[66] Huan Liu and lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transaction On Knowledge and Data Engineering*, 17(4):491–502, April 2005.

[67] Richard Maclin and David Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the* $14^{th}$ *National Conference on Artificial Intelligence*, pages 546–551, Providence, RI, 1997. AAAI/MIT Press.

[68] S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, December 1993.

[69] U. Marti and H. Bunke. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. *International Journal of Pattern Recognition and Artificial Intelligence*, 15:65–90, 2001.

[70] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2004.

[71] J. Moody and J. Utans. Principled architecture selection for neural networks: Application to corporate bond rating prediction. In *Advances in Neural Information Processing Systems*, volume 4. Morgan Kaufmann, 1991.

[72] S. K. Murthy, K. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[73] P. Nair, A. Choudhury, and A. Keane. Some greedy learning algorithms for sparse regression and classification with mercer kernels. *Journal of Machine Learning Research*, 3:781–801, 2002.

[74] P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transaction On Computers*, 26(9):917–922, September 1977.

[75] B. K. Natarajan. Sparse approximate solutions to linear system. *SIAM Journal on Computing*, 24(2):227–234, April 1995.

[76] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Feature selection using multi-objective genetic algorithms for hand written digit recognition. In *Proceedings of the* $16^{th}$ *ICPR*, pages 568–571, 2002.

[77] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Feature selection for ensembles: A hierarchical multi-objective genetic algorith approach. In *Proceedings of the* $7^{th}$ *ICDAR*, pages 676–680, 2003.

[78] David Opitz. Feature selection for ensembles. In *Proceedings of the* $16^{th}$ *International Conference on Artificial Intelligence*, pages 379–384, 1999.

[79] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[80] Nikunj C. Oza and Kagan Tumer. Input decimation ensembles: Decorrelation through dimensionality reduction. In *Proceedings of the* $2^{nd}$ *International Workshop on Multiple Classifier Systems*, pages 210–217, Cambridge, UK, 2001.

[81] A. G. Parlos, K. T. Chong, and A. F. Atiya. Application of the recurrent multilayer perceptron in modelling complex process dynamics. *IEEE Transactions on Neural Networks: Special Issue on Dynamic Recurrent Neural Networks: Theory and Applications*, 5:255–266, March 1994.

[82] Bambang Parmanto, Paul W. Munro, and Howard R. Doyle. Improving committee diagnosis with resampling techniques. In *Advances in Neural Information Processing Systems*, volume 8, pages 882–888. MIT Press, 1995.

[83] T. Poggio and F. Girosi. A sparse representation for function approximation. *Neural Computation*, 10(6):1445–1454, August 1998.

[84] P. Pudil, F. J. Ferri, J. Novovičová, and J. Kittler. Floating search methods for feature selection with nonmonotonic criterion functions. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, pages 279–283, 1994.

[85] J. R. Quinlan. *C4.5: Programs for Mahine Learning*. Morgan Kaufmann, 1993.

[86] J. R. Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the* $13^{th}$ *National Conference on Artificial Intelligence*, pages 725–730, Denver, CO, 1996. AAAI/MIT Press.

[87] S. J. Reeves. An efficient implementation of the backward greedy algorithm for sparse signal reconstruction. *IEEE Signal Processing Letters*, 6(10):266–268, October 1999.

[88] Amanda J.C. Sharkey, editor. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, 1999.

[89] W. Siedlecki and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10(11):335–347, November 1989.

[90] P. Smyth, U. Fayyad, M. Burl, P. Perona, and P. Baldi. Inferring ground truth from subjective labelling of venus images. In *Advances in Neural Information Processing Systems*, volume 7, pages 1085–1092, Denver, CO, 1995. MIT Press.

[91] S. D. Stearns. On selecting features for pattern classifiers. In *Third International Conference on Pattern Recognition*, pages 71–75, Coronado, CA, 1976.

[92] J. A. K. Suykens, L. Lukas, and J. Vandewalle. Sparse approximation using least squares support vector machines. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, pages 11757–11760, May 2000.

[93] Kagan Tumer and Nikunj C. Oza. Decimated input ensembles for improved generalization. In *Proceedings of the International Joint Conference on Nerual Networks*, Washington, DC, 1999.

[94] K. Wang, C.-H. Lee, and B. Juang. Selective feature extraction via signal decomposition. *IEEE Signal Processing Letters*, 4(1):8–11, January 1997.

[95] Ian H. Witten and Eibe Frank. Data Mining: Practical machine learning tools with Java implementations, Morgan Kaufmann, San Francisco, http://www.cs.waikato.ac.nz/∼ml/weka/index.html, 2000.

[96] S. J. Yoo. Efficient traffic prediction scheme for real-time VBR MPEG video transmission over high-speed networks. *IEEE Transactions on Broadcasting*, 48:10–18, March 2002.

[97] Meide Zhao, F. Queck, and Xindong Wu. RIEVL: recursive induction learning in hand gesture recognition. *IEEE Transaction On Pattern Analysis And Machine Intelligence*, 20:1174–1185, 1998.

[98] Zijian Zheng. Generating classifier committees by stochastically selecting both attributes and training examples. In *Proceedings of PRICAI*, pages 12–23, Berlin, 1998. Springer-Verlag.

[99] Zijian Zheng and Geoffrey I. Webb. Stochastic attribute selection committees. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, pages 321–332, Berlin, 1998. Springer.

[100] Zijian Zheng and Geoffrey I. Webb. Stochastic attribute selection committees with multiple boosting: Learning more accurate and more stable classifier committees. Technical Report TR C98/13, Deakin University, Geelong, Victoria, Australia, May 1998.

[101] Zijian Zheng, Geoffrey I. Webb, and Kai Ming Ting. Integrating boosting and stochastic attribute selection committees for further improving the performance of decision tree learning. In *Proceedings of the* $10^{th}$ *IEEE ICTTAI*, pages 216–223, Los Almitos, CA, 1998. IEEE Computer Society Press.

[102] Douglas Zongker and Anil Jain. Algorithms for feature selection: An evaluation. In *Proceedings of the International Conference on Pattern REcognition*, pages 18–22. IEEE, 1996.