INCREMENTAL LEARNING WITH ENSEMBLE BASED SVM CLASSIFIERS FOR NON-STATIONARY ENVIRONMENTS

by

Aycan Yalçın

B.S., Computer Engineering., İstanbul Technical University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2006

ACKNOWLEDGMENTS

I would like to thank to Prof. Fikret Gürgen for his supervision in this MS study and for his support, encouragement and kind tolerance. I also would like to thank to Dr. Zeki Erdem for his guidance, suggestions, contributions and time.

I am also grateful to all my friends for their motivation and encouragement throughout this study.

Special thanks to my family for their endless support in every phase of my life. Without their support, this thesis would have not been accomplished.

ABSTRACT

INCREMENTAL LEARNING WITH ENSEMBLE BASED SVM CLASSIFIERS FOR NON-STATIONARY ENVIRONMENTS

In this thesis, we evaluate the performance of support vector machines (SVM) ensemble, which is constructed by using Learn++ algorithm, on changing environment and propose incorporating forgetting mechanism to adapt this algorithm to changing environment.

In most of the real world applications, the data is collected over an extended period of time and the distribution underlying the data is likely to change by time. These changes make the model built on old data inconsistent with the new data, and regular updating of the model is necessary. For effective learning in a changing environment, the algorithm should be able to detect context change and quickly adjust the hypothesis to the current context. This can be achieved by revising the model by incorporating new examples and eliminating the effect of outdated concepts.

Learn++ is an ensemble based incremental learning algorithm that is able to learn new information. Therefore, it can be easily adapted to changing environments by using a forgetting mechanism to remove the redundant data from the ensemble. In this thesis, we propose using a forgetting strategy that is based on the performance of the base classifiers. Only the best K classifiers or the classifiers whose classification performance exceeds a threshold value are kept in the ensemble.

Our results indicate that incorporating forgetting mechanism improves the classification performance of the proposed algorithm on a changing environment. The proposed algorithm can effectively handle the gradual changes.

ÖZET

DİNAMİK ORTAMLARDA DVM SINIFLAYICI TOPLULUĞU İLE AŞAMALI ÖĞRENME

Bu tezde, Learn++ algoritması ile oluşturulmuş bir destek vektör makinesi (DVM) topluluğunun değişken (dinamik) bir ortamdaki sınıflandırma başarısı değerlendirilmiş ve bu algoritmanın dinamik ortama uyum sağlayabilmesi için eski geçersiz bilgiyi unutma mekanizmasının kullanılması önerilmiştir.

Birçok uygulamada, probleme ait veriler geniş bir zaman aralığında toplanır ve verinin temelini oluşturan dağılım zaman içerisinde değişebilir. Bu da eski veri üzerinde gerçeklenmiş olan modelin yeni veriler ile uyumsuz olmasına sebep olur ve bu durumda modelin düzenli olarak güncellenmesi gerekmektedir. Dinamik bir ortamda önerilen bir sınıflama modelinin başarılı olabilmesi için değişimin sınıflama modeli tarafından fark edilmesi ve modelin yeni ortama hızlı bir şekilde uyum sağlaması gerekir. Bu da oluşturulmuş olan modelin yeni örneklerle güncellenmesi ve eski geçersiz bilginin unutulmasıyla sağlanır.

Learn++, sınıflayıcı topluluğu oluşturarak aşamalı öğrenmeyi sağlayan bir algoritmadır ve yeni bilgiye uyum sağlama, yeni bilgiyi öğrenme becerisine sahiptir. Dolayısıyla, geçersiz bilgileri unutmasını sağlayacak bir yöntem kullanılarak kolayca dinamik ortamlar için uyumlu hale dönüştürülebilir. Bu tezde Learn++ kullanılarak oluşturulmuş bir DVM topluluğunun dinamik ortamlara uyumlu hale getirilebilmesi için topluluğu oluşturan sınıflayıcıların başarısını baz alan bir unutma yönteminin kullanılması öneriliyor. Tüm sınıflayıcılar içerisinde en başarılı K sınıflayıcı veya belli bir eşik değer üzerinde sınıflandırabilen sınıflayıcılar topluluk içerisinde tutularak bu sağlanmış oluyor.

Sonuçlarımız gösteriyor ki, önerilen algoritma yavaş değişen ortamlarda, oluşturulan modelin sınıflama başarısını artırmaktadır.

TABLE OF CONTENTS

ACKNOWLEDGMENTS III

ABSTRACT IV

ÖZET V

LIST OF FIGURES VIII

LIST OF TABLES X

LIST OF SYMBOLS XI

1. INTRODUCTION 1
   1.1. Background and Related Works 1
   1.2. Organization of the Thesis 4

2. SUPPORT VECTOR MACHINES 6
   2.1. Statistical Learning Theory 6
      2.1.1. Vapnik-Chervonenkis(VC) Dimension 8
      2.1.2. Structural Risk Minimization 9
   2.2. SVMClassifiers 10
      2.2.1. Linear Support Vector Machines 11
         2.2.1.1 The Separable Case 11
         2.2.1.2 The Non-Separable Case 15
      2.2.2. Non-Linear Classifiers 17
      2.2.3. Implementation Issues 20

3. LEARN++: AN INCREMENTAL LEARNING ALGORITHM 22
   3.1. IncrementalLearning 22
   3.2. Ensemble of Classifiers 23
   3.3. Learn++ 24

4. CHANGING ENVIRONMENT 30

5. RESULTS 36
   5.1. Evaluated Methods 36

5.2. Simulation Dataset 37
5.2.1. Rotated Chessboard Dataset 37
5.2.2. Circles Dataset 38
5.3. Simulation Results 39
5.3.1. Results on Rotated Chessboard Dataset 39
5.3.2. Results on Circles Dataset 44
6. CONCLUSIONS 48
REFERENCES 50

LIST OF FIGURES

FIGURE 2.1. Relation between empirical risk, VC dimension and expected risk 9

FIGURE 2.2. Nested subsets of functions, ordered by VC dimension 10

FIGURE 2.3. Linear separating hyperplanes for the separable case..... 11

FIGURE 2.4. Relation Two-out-of-many separating lines 12

FIGURE 2.5. The soft decision boundary for a dichotomization problem with data overlapping 15

FIGURE 2.6. A nonlinear SVM without data overlapping 18

FIGURE 2.7. Mapping the input space to the feature space 18

FIGURE 3.1. AdaBoost.M1 Algorithm 25

FIGURE 3.2. Algorithm of Learn++ 27

FIGURE 4.1. Illustration of the three data handling approaches for building online classifier ensembles..... 34

FIGURE 5.1. The rotated chess board dataset 37

FIGURE 5.2. Circles dataset 39

FIGURE 5.3. Performance graphic on chessboard data with 1^o rotation with RBF kernel (σ=40, C=20) without forgetting..... 40

FIGURE 5.4. Performance graphic on chessboard data with 1^{o} rotation with RBF kernel (σ=40, C=20) for top K classifiers 43

FIGURE 5.5. Performance graphic on chessboard data with 1^{o} rotation with RBF kernel (σ=40, C=20) with incorporating threshold approach 44

FIGURE 5.6. Performance graphic on circles data with RBF kernel (σ=5, C=10) 44

FIGURE 5.7. Performance graphic on circles data with RBF kernel with top K classifiers 45

FIGURE 5.8. Performance graphic on circles data with RBF kernel when incorporating threshold mechanism 46

LIST OF TABLES

TABLE 5.1. Contexts of Circles dataset 39

TABLE 5.2. Circles data distribution 39

TABLE 5.3. SVMLearn++ with RBF kernel (σ=40, C=20) on chessboard 40

TABLE 5.4. SVMLearn++ with Polynomial kernel (degree=5, C=15) on chessboard . 41

TABLE 5.5. SVMLearn++ with RBF kernel on chessboard with top K classifiers 41

TABLE 5.6. SVMLearn++ with Polynomial kernel on chessboard with top K classifiers 42

TABLE 5.7. SVMLearn++ with RBF kernel on chessboard for different threshold values..... 43

TABLE 5.8. SVMLearn++ with RBF kernel (σ=5, C=10) on circles 44

TABLE 5.9. SVMLearn++ with Polynomial kernel (degree=10, C=0.1) on circles 44

TABLE 5.10. SVMLearn++ with RBF kernel on circles with top K classifiers 45

TABLE 5.11. SVMLearn++ with polynomial kernel on circles with top K classifiers . 45

TABLE 5.12. SVMLearn++ with RBF kernel on circles for different threshold values 46

TABLE 5.13. SVMLearn++ with Polynomial kernel on circles for different threshold values 46

LIST OF SYMBOLS

h	VC dimension
x_i	attribute i of instance
x^t	instance t
d^t	desired output for instance t
y^t	real output for instance t
b	bias
w	weight vector
α	Lagrange multiplier
N_{sv}	the number of support vectors
C	regulation parameter
C_i	class i
ξ_i	slack variable
L_D	dual Lagrangian

1. INTRODUCTION

1.1. Background and Related Works

Pattern recognition is a research area that aims to determine a description of a given concept from a set of concept examples provided by a supervisor and from the background knowledge. The main goal of designing a pattern recognition system is to find the best model with the best generalization ability on the given task. This objective leads to development of several models to solve the pattern recognition problems; e.g. neural networks, statistical models like linear/quadratic discriminants, decision trees and genetic models, etc. These algorithms vary in their goals, learning strategies, the knowledge representation languages they employ and the type of training data they use.

Classification has been identified as an important problem in the emerging field of data mining. In classification problems, a set of example records, called a training set, where each record consists of several attributes are given. Attributes are either continuous or categorical. One of the attributes, called the classifying attribute, indicates the class to which each example belongs. The objective is to construct a model of the classifying attribute based on the other attributes to classify the future examples. Applications of classification arise in different areas, such as image analysis, character recognition, speech analysis, disease diagnostic, human identification, market prediction, etc.

The large margin methods such as Support Vector Machines and boosting have significantly advanced the state of the art by improving classification accuracy and by increasing the applicability of machine learning methods. One of the key benefits of these methods is their ability to efficiently learn in high dimensional feature spaces, either by the use of implicit data representations via kernels or by explicit feature induction.

Support Vector Machine (SVM) is a supervised learning method based on statistical learning theory presented by V. N. Vapnik (Vapnik, 1995). SVMs are applicable to both classification and regression, but this thesis only considers the use of SVMs for classification. SVMs focus on finding a linear discriminant function that maximizes the

separation or margin (the distance from the separating hyperplane to the nearest examples). This criterion provides a good upper bound of the generalization error, while minimizing the misclassification error. One of the most important characteristics of SVMs is that they can handle very large feature spaces. Their generalization ability and computational efficiency are both independent of the dimension of the input space. Therefore, they are efficient in very large feature space. The other benefit compared to conventional algorithms is that the SVM results in a globally optimal solution for the problem under study. SVMs have been successfully applied to many different areas such as face recognition, text categorization, etc.

While the support vector machine (SVM) can provide a good generalization performance, the classification result of the SVM is often far from the theoretically expected level in practical implementation because they are based on approximated algorithms due to the high complexity of time and space. Some studies have been presented to improve the classification performance and accuracy of the real SVM with ensemble based methods.

Ensemble based systems have attracted a great attention over the last decades due to their empirical success over single classifier systems on a variety of applications. Ensemble methods aim at improving the generalization performance of a given statistical learning technique. The general principle of ensemble methods is to construct a linear combination of some base classifiers, instead of using a single fit of the method. A rich collection of algorithms have been developed using multiple classifiers, such as boosting, bagging, binning, stacking, mixtures of experts, etc. Most of the widely studied ensembles use unstable classifiers or weak classifiers, such as decision trees, neural networks, etc. as base classifiers, but few are reported about the effectiveness of the ensembles of strong classifiers, like SVMs (Erdem *et al.* ,2005a)(Erdem *et al.* , 2005b) (Valentini and Dietterich,2002).

In most of the real-world applications the data are generated continuously and is available in small batches over a period of time. The acquisition of a representative training data is expensive and time consuming. In such settings, it is necessary to revise the existing system to incorporate the new examples. Also modifying a trained system may be cheaper

in time cost than building a new system. Due to these reasons, incremental learning ability is very important for machine learning approaches. (Kotsiantis and Pintelas, 2004), (Polikar *et al.*, 2001) and (Muhlbaier *et al.* ,2004) are some example works that the model is trained in an incremental fashion to accommodate new data without compromising classification performance on old data.

One of the recently proposed incremental algorithms is Learn++. Learn++ is an ensemble based algorithm proposed for incremental learning (Polikar *et al.*, 2001). This algorithm is inspired by the AdaBoost algorithm. Learn++ achieves incremental learning by using ensemble of classifiers where the base classifiers are trained using strategically updated distribution of the training data. The hypotheses are combined using weighted majority voting procedure. The strength of this algorithm is its ability to accommodate new data without forgetting previously acquired knowledge and without access to previously used data. Also a modified version of the Learn++, Learn++.MT, is proposed to reduce the number of generated classifiers and improve the performance of Learn++ (Muhlbaier e*t a.l*, 2004). Learn++.MT uses Dynamic Weighing Voting (DWM) algorithm to provide these improvements. In this thesis, a SVM ensemble will be constructed using Learn++ algorithm and we will try to adapt the proposed algorithm to changing environments.

Currently, the majority of the research has been devoted to static environment, where the data is drawn from a stationary distribution. However, for many learning tasks the data is collected over an extended period of time and the underlying data generating mechanism or the target concept is likely to change over time. Different approaches have been proposed to cope with this concept drift. Three general approaches are; instance selection, instance weighting and ensemble learning. In instance selection the goal is to select the data which are most relevant to the current concept. The most common technique is using a sliding window. FLORA (Widmer and Kubat,1996) and FRANN (Kubat and Widmer,1994) are some examples of instance selection algorithms using fixed size window. On the other hand, FLORA2 (Widmer & Kubat,1996) and (Klinkenberg,and Joachims, 2000) use a window of adaptive size. In instance weighting approach, examples are weighted according to their age and relevance with the current concept, as in (Klinkenberg,2004). Ensemble based learners combine the decisions of a set of base classifiers and also use a mechanism to handle new instances and forget redundant data. A number of ensemble based algorithms

have been proposed to cope with changing environment. These algorithms vary depending on their data selection methods, combination rules and forgetting mechanisms. AddExp (Kolter and Maloof,2005), (Scholz and Klinkenberg, 2005) uses ensemble based methods to handle the concept drift. Also there are some methods using ensemble of decision trees, such as CVFDT (Domingos *et al.*, 2001) and CDC (Stanley, 2003), for handling of concept drift. (Klinkenberg,2001) proposed a SVM based method for the cases that the labeled data are not enough for an effective learning and handles also unlabeled data for the learning.

The recent innovations motivated this study on non-stationary environments. In incremental learning approaches, the classification model is updated to accommodate the new training data. Therefore, these algorithms can be easily adapted to the changing environment by adding a forgetting mechanism. In this thesis, we will evaluate the performance of an incremental learning approach, SVM ensemble using Learn++, on changing environment. Also we incorporate a forgetting mechanism into Learn++ algorithm to handle the concept drift and evaluate the results.

1.2. Organization of the Thesis

The rest of this thesis is organized as follows:

In chapter 2, we give the main concepts behind SVMs approach. Additionally, details of linear and non-linear SVMs have been given in this chapter.

In chapter 3, an ensemble based incremental learning algorithm Learn++ is described. First the main concepts to understand the algorithm are explained. Finally, Learn++ algorithm is described in details.

Chapter 4 provides an overview of concept drift and related works. Initially a basic description of concept drift is given. Following that, the types of concepts drift and the approaches to handle the concept drift are presented.

In chapter 5, the experimental results are presented.

Finally, chapter 6 summarizes the findings and gives directions for possible future works.

2. SUPPORT VECTOR MACHINES

In this section, the theoretical foundations and general concepts of SVM will be explained. This chapter will start by explaining *statistical learning theory* upon which SVMs are based. Secondly, basics of *VC (Vapnik-Chervonenkis) theory* and *Structural Risk Minimization* principle, which is the essential idea under the SVM, will be outlined. Afterwards, the base SVM classifiers and the advancement of SVMs from being a linear classifier to a non-linear classifier will be described. Finally, implementation methods of SVMs will be explained.

2.1. Statistical Learning Theory

The statistical learning theory was first introduced by Vladimir Vapnik (Vapnik, 1995). The statistical learning theory concerns the problem of choosing desired functions on the basis of empirical data. The main goal of the statistical learning theory is to find the function which best describes the relation between inputs and the outputs. This is studied in a statistical framework that is there are assumptions of statistical nature about the underlying phenomena.

In a typical statistical learning problem, given a finite sample of data ($S = \{(x_i, y_i) \in X \text{ x } Y \mid i = 1,...,m\}$) generated by an unknown source, the model yielding best predictions on future data generated by the same source is aimed to be found. It is assumed that there is a fixed but unknown probability distribution $P(X,Y)$ in X x Y space. A function is defined to learn the $x_i \rightarrow y_i$ mapping. Considering a learning machine with a set of adjustable parameters α, the learning machine seeks to find α to find the mapping. The choice of the parameter and so the selection of the model is very important for generalization performance. There are two basic constructive approaches to design a model with a good generalization performance (Kecman, 2004) (Vapnik 1998):

- choosing an appropriate structure of the model and keeping the estimation error fixed in this way, minimizing the training error
- keeping the value of the training error fixed and minimizing the confidence interval

The main difference between these approaches is the minimization of different cost (error, lost) functionals. Classic NNs implements the first approach, but SVMs use the second approach.

In order to choose the best approximation, the loss function, which is defined as the discrepancy between the response of the supervisor and the response of the learning machine, should be measured (Vapnik, 1995). The expected value of the test error for a trained machine can be defined as:

$$R(\alpha) = \int \frac{1}{2} |y - f(x, \alpha)| dP(x, y) \tag{2.1}$$

The R(α) is also called expected risk. When $p(x,y)$ exists, $dP(x,y)$ can be written as $p(x,y)dxdy$. Ideally we would like to minimize the expected value of the loss which is given by the risk functional. However, in a realistic scenario, the information provided by the sample is incomplete, therefore the probability distribution function is unknown and the training set includes the only available information. Since the estimation of the probability distribution is difficult especially in high-dimensional space, the classification function is tried to be found without using $dP(x,y)$. This can be achieved by choosing the function, within a set of functions, which minimizes the expected risk R(α). Since we only have the data distribution of the training data samples, not the full probability distribution, we can only calculate the risk over the training data, also known as empirical risk. The most common method is to minimize the empirical risk. As shown below, no probability distribution appears in the empirical risk formulation:

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(x_i, \alpha)| \tag{2.2}$$

Minimizing the empirical risk is a straightforward method and is used by many learning machines including neural networks and most instance based methods. This risk minimization principle is called as *Empirical Risk Minimization (ERM).* ERM guarantees a small value of the expected risk provided that sufficient training data is available. However, minimizing the training error does not necessarily minimize the error on test set when the

hypothesis space is too large. Since the capacity of the learning machine is not considered in *ERM*, minimizing the empirical risk functional instead of true risk functional might lead to over-fitting problem. Over-fitting occurs when the classification performance of the best model relative to the training data tends to be significantly worse when applied to new data. This means that the average loss significantly underestimates the expected loss. We can avoid the over-fitting problem by controlling the model complexity. The simplest model that explains the data should be preferred (Ocam's razor).

The value $\frac{1}{2}|y-f(x_i,\alpha)|$ is called as the loss. Vapnik shows that choosing some η such that $0 \leq \eta \leq 1$, the upper bound of the error is (Burges, 1998):

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\left(\frac{(h(\log(2l/h)+1) - \log(\eta/4)}{l}\right)} \tag{2.3}$$

where h is a nonnegative integer called as "Vapnik Chervonenkis(VC) dimension" (explained in the next section), which is a measure of capacity. The second term in the right side of the equation is called "VC confidence". Hence, the risk can be estimated by calculating the upper bound.

2.1.1 Vapnik-Chervonenkis (VC) Dimension

The VC (Vapnik-Chervonenkis) dimension is a property of a set of approximating functions of a learning machine that is a measure of the complexity (or capacity) of that class of functions. The VC dimension for a set of classifying functions is defined as the size of the largest dataset that can be shattered by this set of functions. A learning machine (a class of functions f(α)) is said to shatter the dataset X, if this learning machine (this class of functions) can correctly assign all possible labels to the instances in X.

If a set of functions has a high capacity that can explain every possible dataset, these functions are expected to have low empirical risk, and so this function set is not expected to generalize well. On the other hand, functions with small capacity but able to explain our particular data can work well on the unseen data. Also based on the above formula given for expected risk, VC confidence interval increases with an increase in VC dimension h for a

fixed number of the training data pairs *l*. The relation between empirical risk, VC dimension and expected risk is shown in figure 2.1.

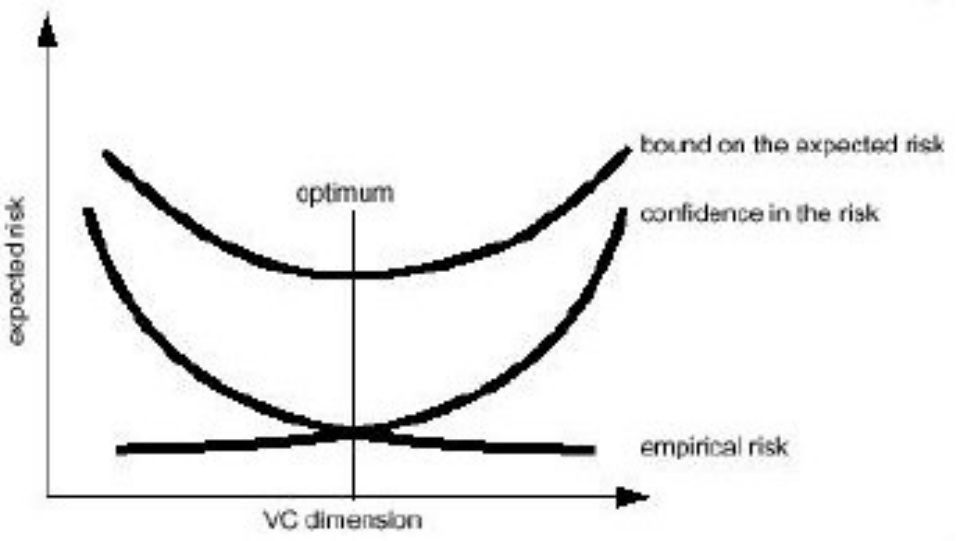


Figure 2.1. Relation between empirical risk, VC dimension and expected risk

2.1.2 Structural Risk Minimization (SRM)

In contrast to ERM, which aims to minimize the empirical risk, *Structural Risk Minimization (SRM)* principle suggests a tradeoff between the empirical risk and the complexity of the approximation function. The goal of the SRM is to find a model with the right capacity to describe the given data, therefore, aims to minimize the expected risk:

$$R_{emp}(\alpha) = \int error(f(x_i, \alpha)y_i)\, dP(x,y) \tag{2.4}$$

Since the underlying data distribution cannot be calculated in most of the classification problems, an estimated risk is used instead.

$$R_{exp}(\alpha) \leq R_{emp}(\alpha) + VC(H) \tag{2.5}$$

The VC confidence term depends on the chosen class of functions, whereas the empirical risk and actual risk depend on the particular function chosen by the training procedure. The goal is to find the subset of the chosen set of functions that minimize the

risk bound. A nested structure is introduced by dividing the entire class of functions into nested subsets ordered by VC dimension (figure 2.2)

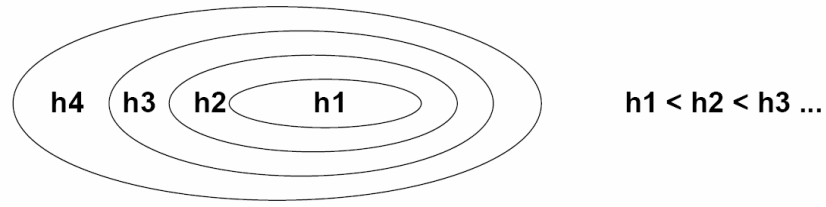


Figure 2.2. Nested subsets of functions, ordered by VC dimension.

In such a nested set of functions, every function always contains a previous, less complex, function. For each subset, a h or a bound on h is computed and the subset minimizing the bound on the actual risk, whose sum of empirical risk and VC confidence is minimal, is selected.

2.2. SVM Classifiers

SVM is a supervised learning technique proposed for both classification and regression problems by Vapnik. Support vector machines (SVMs) have been successfully applied in many different areas such as handwritten digit recognition, object recognition, speaker identification, face detection, text categorization, etc. The SVM is developed based on the SRM principle and the basic idea is that for a given learning task with a given finite amount of training data, the best generalization performance will be achieved if the right balance is struck between the accuracy attained on that particular training set and the capacity of the machine, that is the ability of the machine to learn any training set without error (Burges 1998). SVM method achieves this by constructing an optimal separating hyperplane that maximizes the margin, where the margin means the minimal distance from the separating hyperplane to the closest data points.

In this section linear and nonlinear SVMs will be described.

2.2.1 Linear Support Vector Machines

2.2.1.1. The Separable Case

Consider a binary classification problem, for a linearly separable dataset $\boldsymbol{S}=\{(\boldsymbol{x}_i,y_i) \mid i=1,\ldots,m\}$. The separating hyperplane can be described by the equation (2.6).

$$f(x) = w^T x + b = \sum_{i=1}^{m} w_i x_i + b \tag{2.6}$$

where $\mathbf{w}$ is a weight vector and the scalar b is bias, and these values determine the position of the separating hyperplane.

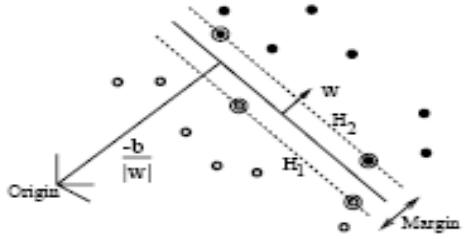


Figure 2.3. Linear separating hyperplanes for the separable case. The support vectors are circled (from Burges 1998)

In the case of the classification of linearly separable data, there are many hyperplanes that satisfy the equality given with the equation 2.6 and solve this classification task. Intuitively, a hyperplane that passes too close to the training examples is expected to be sensitive to noise and, therefore, less likely to generalize well for data outside the training set. Instead, a hyperplane that is farthest from all training examples is expected to have better generalization capabilities. Also as shown in figure 2.4, decision boundary with large margin seems to probably give a better generalization performance than that having a smaller margin. Therefore, SVM classifier seeks to find the hyperplane with the largest margin among all the hyperplanes that minimize the empirical risk.

This can be formulated as follows assuming that all the training data satisfy the following constraints:

$$w^T.x_i + b \geq +1 \qquad for\ y = +1 \tag{2.7}$$

$$w^T.x_i + b \leq -1 \qquad for\ y = -1 \tag{2.8}$$

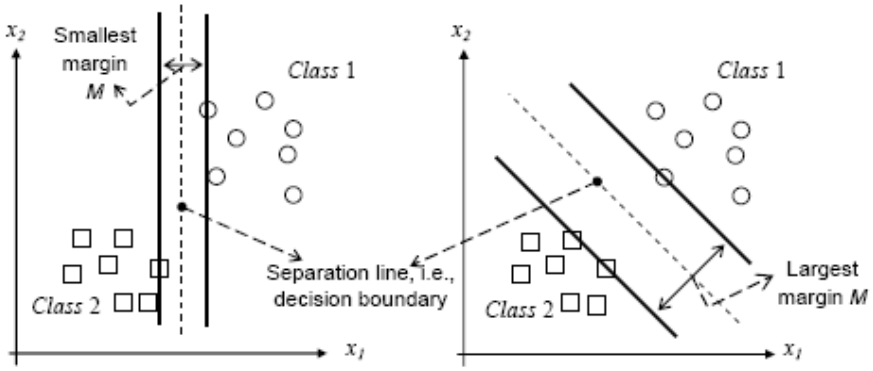


Figure 2.4. Relation Two-out-of-many separating lines: a good one with a large margin (right) and a less acceptable separating line with a small margin (left) (from Kecman 2004)

Or this can be combined as:

$$y_i\,(w^T.x_i + b) - 1 \geq 0, \quad i = 1\ldots m \tag{2.9}$$

The pair (w,b) can be rescaled as follows without loss of generality:

$$\underbrace{\min}_{x \in X} \left| w^T x_i + b \right| = 1 \tag{2.10}$$

This constraint defines a set of canonical hyperplanes. In order to restrict the expressiveness of the hypothesis space, the SVM searches for the simplest solution that classifies the data correctly, by maximizing the margin.

Derivation of the margin between the hyperplanes can be found in (Kecman,2004) or other references. The margin between two hyperplanes can be derived by both geometric and algebraic methods and is formulated as:

$$M = \frac{2}{\|w\|} \tag{2.11}$$

The maximum margin can be found by minimizing the Euclidean norm of the weight vector w. Therefore, the problem of maximizing the margin is transformed to minimizing $\|w\|^2$ or ($\frac{1}{2}$ $w^T w$), subject to constraints in 2.9. This optimization problem is now a convex quadratic programming (QP) problem and the Lagrangian optimization technique is an efficient way to solve this optimization problem (Burges, 1998). The Lagrangian function for this optimization problem is:

$$L(w,b,\alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{m} \alpha_i \left[y_i \left(w^T x_i + b \right) - 1 \right] \tag{2.12}$$

where $\alpha = (\alpha_1, \alpha_i, \ldots . \alpha_m)^T$ is the Lagrange multiplier. The Lagrangian L has to be minimized with respect to the primal variables **w** and b and maximized with respect to the dual variables α_i to find an optimal saddle point. This problem can be solved either in primal space (the space of the parameters w and b) or in dual space (the space of Lagrange multipliers) (Kecman, 2004).Since handling of the inequality constraints in a direct way is a difficult task, a solution in a dual space will be considered. The basic idea under dual methods is that the fundamental unknowns of the problem are the dual variables (Cristianini and John, 2000). The primal can be transformed to dual by using Karush-Kuhn-Tucker (KKT) conditions. For the convex optimization problems, satisfying KKT conditions is necessary and sufficient for the solution of the problem. These conditions are:

- The optimality condition ($\partial J/\partial z=0$) is satisfied by setting to zero the derivative of the Lagrangian with respect to the primal variables at the saddle point:

$$\frac{\partial L}{\partial w_o} = 0 \quad \Rightarrow \quad w_o = \sum_{i=1}^{m} \alpha_i y_i x_i , \tag{2.13}$$

$$\frac{\partial L}{\partial b_o} = 0 \quad \Rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0 , \tag{2.14}$$

- Also the complementary conditions, which states that the products between dual variables and constraints equals zero at the solution point, must be satisfied:

$$y_i \left(w^T x_i + b \right) - 1 \geq 0 , \qquad i = 1 \ldots m \tag{2.15}$$

$$\alpha_i \geq 0 , \qquad \forall i \tag{2.16}$$

$$\alpha_i \left[y_i \left(w^T x_i + b \right) - 1 \right] = 0 , \qquad \forall i. \tag{2.17}$$

The resulting dual formulation by substituting (2.13) and (2.14) into the primal variables in Lagrangian is:

$$L_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} y_i y_j \alpha_i \alpha_j x_i^T \cdot x_j . \tag{2.18}$$

Therefore, the problem of finding a saddle point for $L_P(w,b)$ is transformed into maximizing $LD(\alpha)$. In order to find the optimal separating hyperplane, the dual Lagrangian L_D has to be maximized with respect to α_i, subject to the following constraints:

$$\alpha_i \geq 0 \qquad \forall i \tag{2.19}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \qquad \forall i \tag{2.20}$$

In the dual Lagrangian $L_D(\alpha)$ the training data appears only as dot products $x_i^T x_j$ and this property will be very handy perform the classification in a higher (e.g., infinite) dimensional space. Furthermore, the dual problem scales with the amount of training data (there is one Lagrange multiplier per example). Moreover, the KKT condition allows us to identify the training examples that define the largest margin hyperplane. These examples will be known as Support Vectors. The support vectors are the only points having non-zero Langrange multipliers. For the linearly separable case, the support vectors lie on one of the two hyperplanes that define the largest margin and the number of the support vectors is generally much less than the number of total training samples.

The optimal hyperplane (vector w_0) is a linear combination of the vectors of the training set:

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{2.21}$$

and the bias term b is found from the KKT complementary condition on the support vectors:

$$b_o = \frac{1}{N_{sv}} \left[\sum_{s=1}^{N_{sv}} \left(y_s - x_s^T w_o \right) \right], \qquad s = 1,\ldots,N_{sv} \tag{2.22}$$

where N_{svv} is the number of the support vectors.

The decision function is then given by:

$$f(x) = \sum_{i=1}^{m} y_i \alpha_i x_i^T x + b_o \tag{2.23}$$

2.2.1.2. The Non-Separable Case

The approach explained so far is valid for linearly separable data where there is a perfect mapping x->f(x,α) without overlapping. However, in practice this is rare and in most of the real world problems the data does not satisfy this condition. In case of overlapping, the above solution cannot be used since the constraint given by the equation 2.9 cannot be satisfied. Therefore, the algorithm presented in the previous section must be extended in order to handle the non-separable data.

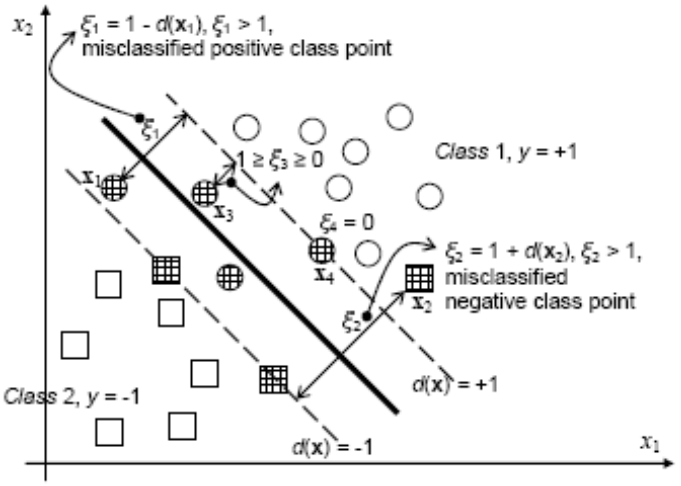


Figure 2.5. The soft decision boundary for a dichotomization problem with data overlapping.

The main problem with the above approach is that it produces a hypothesis with zero training error and, in the case of noisy data; a hypothesis with no training error will result in poor generalization performance. So the given formulation must be changed to allow some data to be unclassified (Figure 2.5). In practice a soft margin is allowed and the training data points inside this margin are neglected. By doing so, an optimal separating hyperplane can be constructed for the remaining training data points. A regulation/penalty parameter C is introduced to control the width of the soft margin. The classifier with the maximal margin can be found by striking a balance (by searching a tradeoff) between the misclassification and VC dimension of the model (Vapnik, 1995). The misclassification is considered by introducing non-negative slack variables $\xi_i \geq 0$ for each training example requiring that:

$$w^T.x_i + b \geq +1\text{-}\xi_i, \qquad \xi_i \geq 0 \quad \text{for} \quad y_i = +1 \tag{2.24}$$

$$w^T.x_i + b \leq -1+\xi_i, \qquad \xi_i \geq 0 \qquad \text{for } y_i = -1 \tag{2.25}$$

and 2.24 and 2.25 can be merged as follows:

$$(w^T.x_i + b) \geq 1-\xi_i, \quad \xi_i \geq 0 \tag{2.26}$$

The value of ξ_i gives the distance between the training example to the decision boundary and the sum of ξ_i values gives an upper bound on the number of training errors.

The new problem can be formulated as :

$$\min \frac{1}{2} w^T w + C\sum_{i=1}^{m} \xi_i^k \ . \tag{2.27}$$

As previously explained C is defined as a regulation parameter and suggests a trade-off between training error and the margin. Choosing a large C leads to small number of misclassification, means over-fıtting on the training data, hence for C = ∞, misclassification error will be zero which is impossible for overlapping case. Therefore, a feasible solution can be found only for C < ∞. In practice, C is selected by using cross-validation technique.

The parameter 'k' in the given formulation is an integer value which is typically 1 or 2. In the case that k is chosen as 1, neither the slack variables nor their Lagrange multiplier will appear in the dual Lagrangian L_D. The solution to the quadratic programming problem (2.27) is given by the saddle point of the primal Lagrangian Lp (w, b, ξ, α, β) shown below:

$$L_P(w,b,\xi,\alpha,\beta) = \frac{1}{2} w^T w + C\left(\sum_{i=1}^{m} \xi_i\right) - \sum_{i=1}^{m} \alpha_i \left[y_i \left(w^T x_i + b\right) - 1 + \xi_i\right] - \sum_{i=1}^{m} \beta_i \xi_i \ . \tag{2.28}$$

where α_i, β_i are the Lagrange multipliers. The optimal saddle point can be found by minimizing L_P with respect to w, b, ξ and maximizing L_P with respect to α_i, β_i. Considering a solution as in the separable case, following constraints must be satisfied:

$$\frac{\partial L_P}{\partial w_o} = 0 \qquad \Rightarrow \qquad w_o = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{2.29}$$

$$\frac{\partial L_P}{\partial b_o} = 0 \qquad \Rightarrow \qquad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.30}$$

$$\frac{\partial L_P}{\partial \xi_{i_o}} = 0 \qquad \Rightarrow \qquad \alpha_i + \beta_i = 0 \tag{2.31}$$

Also the KKT complementary conditions must be satisfied:

$$\alpha_i \left[y_i \left(w^T x_i + b \right) - 1 + \xi_i \right] = 0, \qquad i = 1 \ldots m \tag{2.32}$$

$$\beta_i \xi_i = (C - \alpha_i)\xi_i = 0, \qquad i = 1 \ldots m \tag{2.33}$$

Due to the KKT complementary conditions, the last two terms will disappear from the dual Lagrangian problem. The classifier for the soft margin is given below (same as the hard margin classifier):

$$L_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} y_i y_j \alpha_i \alpha_j x_i^T \cdot x_j \,. \tag{2.34}$$

The optimal hyperplane can be found by maximizing the dual Lagrangian L_D with respect to α_i, subject to the constraints:

$$C \geq \alpha_i \geq 0, \qquad i = 1 \ldots m \tag{2.35}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.36}$$

Comparing the solutions for the linearly separable and non-separable cases, the only difference is the modified bounds of the Lagrange multipliers α_i (the constraint 2.34). In the non-separable case, C is the upper bound of the Lagrange multipliers. Finally, expression for the decision function for a soft margin classifier is same as for linearly separable case as given by 2.34.

2.2.2. Non-Linear Classifier

In the most of the real-world problems, the data is overlapped and can only be modeled by non-linear decision surfaces. The methods presented in the previous section can

only handle the data that can be modeled by linear hyperplanes, therefore, it should be extended to handle nonlinearly separable data.

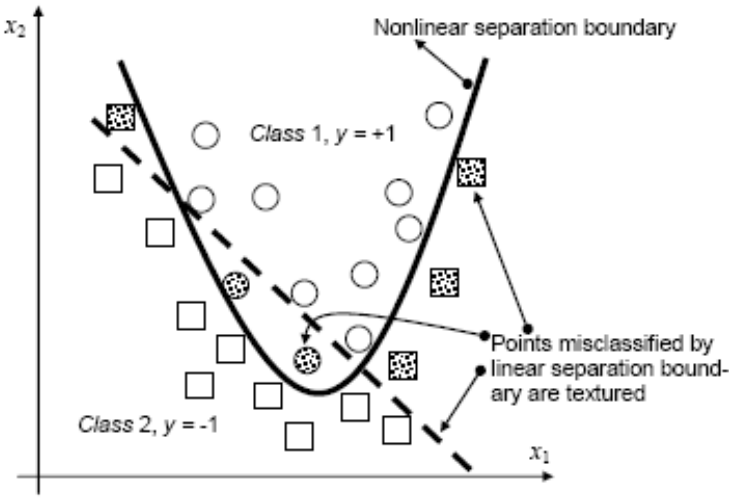


Figure 2.6. A nonlinear SVM without data overlapping (from Kecman, 2004)

The basic idea when applying SVM to nonlinear problems is to map the original nonlinearly separable data into a high-dimensional feature space (through some nonlinear mapping), where the data is linearly separable (Figure 2.7).

$$x \in \mathrm{R}^{\mathrm{N}} \rightarrow \Phi(x) = \left[\phi_1(x) \;\; \phi_1(x), \ldots, \phi_1(x)\right]^T \tag{2.37}$$

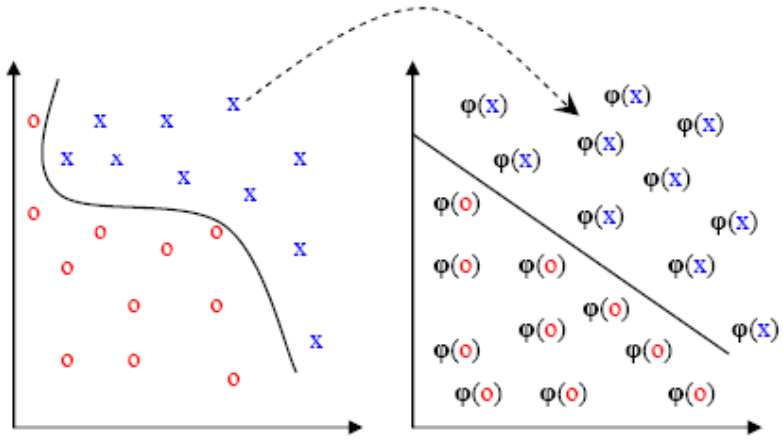


Figure 2.7. Mapping the input space to the feature space, where linear classification is possible.

The optimal separating hyperplane can be found in this feature space by using the above technique, and once the hyperplane is found, it can be mapped back to the original input space. In the feature space, the linear decision function in dual form is given by:

$$f(x) = \text{sign}(w^{\mathrm{T}}\Phi(x) + b) = \text{sign}\left(\sum_{i=1}^{m} y_i \alpha_i \Phi^T(x_i)\Phi(x) + b\right) \quad (2.38)$$

Because of the complexity of finding the optimal hyperplane in a higher dimensional space, 'kernel trick' is used in the solution of constructing non-linear decision surfaces. The mapping is performed implicitly by using kernels for estimating the inner product in feature space. In the case of the use of kernel function in the training algorithm, the explicit calculation of $\Phi(x)$ is not required anymore. Instead of calculating the scalar product $\Phi^T(x_i)\Phi(x)$ in the feature space, the Kernel function is directly calculated for given training data vectors in an input space:

$$K(x_i, x_j) = \Phi^T(x_i)\Phi(x_j) \quad (2.39)$$

Thus, the decision function can be rewritten as follows:

$$f(x) = \text{sign}\left(\sum_{i=1}^{m} y_i \alpha_i K(x_i, x) + b\right) \quad (2.40)$$

The optimal decision surface can be calculated without any substantial increase in computational expense with a suitable choice of kernel. A function must satisfy the following conditions in order to be used as a feasible kernel:

1. The kernel functions must be symmetric
2. The kernel functions must satisfy the Mercer's teorem

Below is given some popular kernels:

- The homogenous polynomial kernel :

$$K(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_\mathrm{i} \cdot \mathrm{x}_\mathrm{j})^{\mathrm{d}} \quad (2.41)$$

- The Gaussian kernel (also called Radial Basis Function (RBF) kernel):

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2} \quad (2.42)$$

- The linear kernel (also called polynomial kernel):

$$K(\mathrm{x}_i, \mathrm{x}_j) = \mathrm{x}_\mathrm{i} \cdot \mathrm{x}_\mathrm{j} \quad (2.43)$$

Having reviewed the basic concepts, we can proceed with the method for the designing nonlinear separating hyperplanes. The procedure of designing the optimal separating hyperplane in a feature space is same as the design of hard and soft margins in an input space. The dual Lagrangian in the feature is given by:

$$L_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} y_i y_j \alpha_i \alpha_j \Phi_i^T \cdot \Phi_j \,. \tag{2.44}$$

by using an appropriate kernel, the below dual Lagrangian should be maximized:

$$L_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} y_i y_j \alpha_i \alpha_j K\left(x_i \cdot x_j\right) \tag{2.45}$$

subject to the constraints:

$$\alpha_i \geq 0, \qquad i = 1 \ldots m \tag{2.46}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.47}$$

Considering an overlapping in the training data, the constraints for α_i should be changed. Thus the nonlinear soft margin classifier will be a solution for the problem given by the equation 2.45 subject to the following constraints:

$$C \geq \alpha_i \geq 0, \qquad i = 1 \ldots m \tag{2.48}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.49}$$

Finally, after the calculation of the dual variables, the decision hyper-surface is determined by:

$$f(x) = \sum_{i=1}^{m} y_i \alpha_i K(x, x_i) + b \tag{2.50}$$

2.2.3. Implementation Issues

Even though the implementation of SVM theory is not straightforward, it is applicable. In this section, we will present the major steps of SVM classifier design.

When designing a SVM classifier, first step is to choose the kernel function and the kernel parameters. After that, the regulation parameter C is selected. Cross validation technique can be used to choose appropriate kernel parameters and regulation parameter. Finally, the QP problem is solved using an appropriate QP method. MINOS, LOQO, TOMLAB or Matlab Optimization Toolbox can be used to solve the optimization problem. Once the model is constructed, the model is tested on the test dataset.

Many different tools and libraries have been proposed for the implementation of SVM theory. The most popular ones are LIBSVM, SVMLight, mySVM, SVM Torch. In this thesis, OSU SVM library is used. OSU SVM is a Support Vector Machine toolbox for the MATLAB numerical environment based on LIBSVM (Chang and Lin, 2001).

3. LEARN++: AN INCREMENTAL LEARNING ALGORITHM

Learn++ is an incremental learning algorithm which is inspired by the AdaBoost (adaptive boosting) algorithm. This algorithm is proposed as an ensemble based classifier.

This section includes review of basic concepts to understand Learn++: incremental learning and ensemble of classifiers. Finally, details of Learn++ algorithm are explained.

3.1. Incremental Learning

In many real-world applications, the data is collected in small batches over a period of time. Furthermore, new classes may be introduced in the subsequent batches. In such cases, an existing classifier should be able to acquire newly introduced knowledge without forgetting previously acquired knowledge. This ability of the classifier is called as *incremental learning*. Incremental learning is primarily focused on the sequential processing of the data, so that the resulting model is no worse than the classifier trained on the batch data. Desired qualities of an incremental learning algorithm can be summarized as (Kuncheva, 2004):

- *Single pass through data*: To learn from each example without revisiting it
- *Limited memory and processing time*: Each sample should be processed in a constant time regardless of the number of the past data
- *Any time learning*: If stopped at time t the algorithm should provide the best answer

Many algorithms have been suggested for incremental learning. The main approaches used for incremental learning are growing or pruning of classifier architecture, selection of most informative training samples or modification of classifier weights. The proposed algorithms can learn from the new data, however, they require access to old data or suffer from catastrophic forgetting or unable to accommodate new classes. The proposed algorithm Learn++ is an incremental learning algorithm that satisfies all the following criteria:

1) It should be able to learn additional information from new data.
2) It should not require access to the original data, used to train the existing classifier.

3) It should preserve previously acquired knowledge (that is, it should not suffer from catastrophic forgetting).
4) It should be able to accommodate new classes that may be introduced with new data. (Polikar *et al.*, 2001)

3.2. Ensemble of Classifiers

Satisfying the above criteria, Learn++ also follows a different approach to the incremental learning problem. Learn++ achieves incremental learning by using ensemble of classifiers approach. Ensembles of classifiers are widely used to improve the accuracy of a classification system. An ensemble of classifiers combines the decisions of a set of classifiers to classify the new examples. It has been discovered that, in most cases ensembles produce more accurate predictions than the individual classifiers that make them up (Dietterich, 1997).

Many methods have been developed for construction of ensembles. A review of ensemble methods can be found in (Dietterich, 1997) and (Dietterich, 2000). Various combining methods may differ in their architectures, combination methods and the selection of individual classifiers. One general technique for constructing ensembles is subsampling of training examples. The learning algorithm is run several times, each time with a different subset of the training data. Some of these methods, such as Bagging and Boosting are *meta-learners* i.e. they can be applied to *any* base learner. Other methods are specific to particular learners.

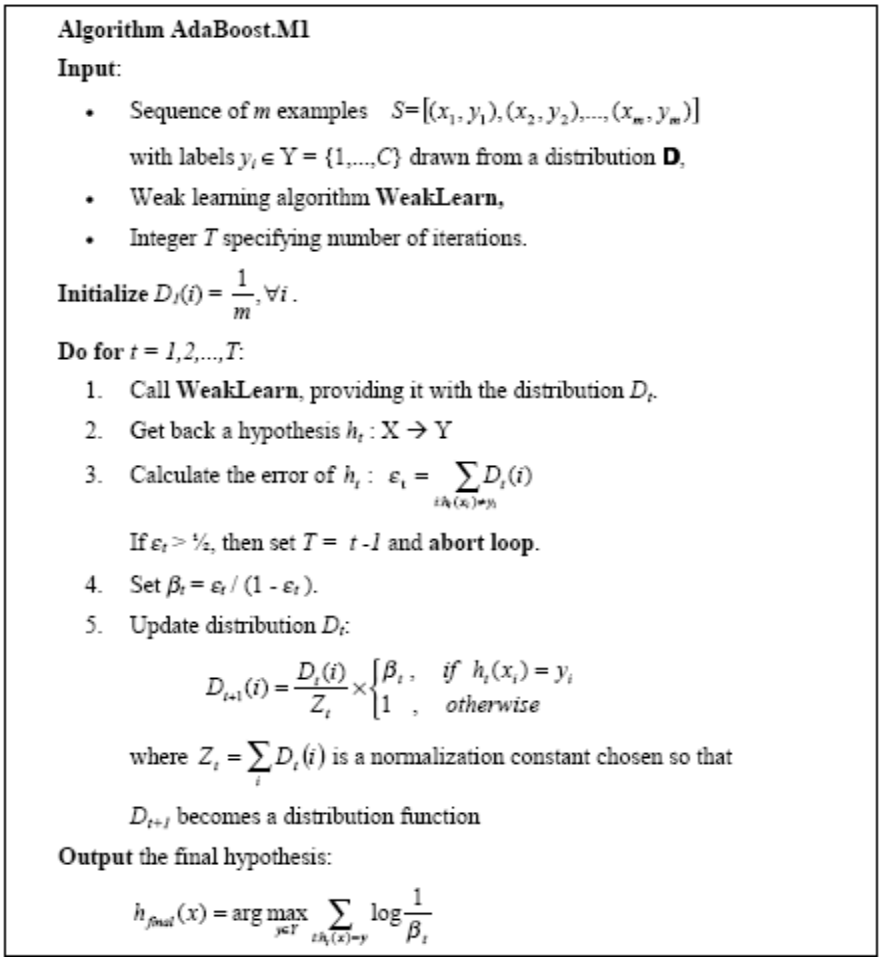
In a Bagging ensemble, each classifier is trained on a set of m training examples, drawn randomly with replacement from the original training set of size m. Such training set is called a *bootstrap replicate* of the original set. Predictions on new examples are made by taking the majority vote of the ensemble. Bagging is typically applied to learning algorithms that are *unstable*, i.e., a small change in the training set leads to a noticeable change in the model produced. Since each ensemble member is not exposed to the same set of examples, they are different from each other. By voting the predictions of each of these classifiers, Bagging seeks to reduce the error due to variance of the base classifier. Bagging of *stable* learners, such as Naive Bayes, does not reduce error.

Another method used to construct ensembles by manipulating the training examples is boosting. There are many variations of boosting. Learn++ was inspired from the AdaBoost algorithm which is first proposed by Freund and Schapire to improve the classification performance of weak classifiers (see algorithm 1). Boosting works by repeatedly running a given weak algorithm on various distributions over the training data and then combining the hypothesis produced by the weak learner into a single hypothesis.

The boosting algorithm AdaBoost assumes that the base learner can handle weighted examples. If the learner cannot directly handle weighted examples, then the training set can be sampled according to a weight distribution to produce a new training set to be used by the learner. ADABOOST maintains a set of weights over the training examples; and in each iteration i, the classifier C_i is trained to minimize the weighted error on the training set. The weighted error of C_i is computed and used to update the distribution of weights on the training examples. The weights of misclassified examples are increased and the weights on correctly classified examples are decreased. The next classifier is trained on the examples with this updated distribution and the process is repeated. After training, the ensemble's predictions are made using a weighted vote of the individual classifiers: $\sum w_i C_i (x)$. The weight of each classifier, w_i, is computed according to its accuracy on the weighted example set it was trained on.

3.3. Learn++

Learn++ was inspired from the AdaBoost algorithm and was first introduced in (Polikar, 2001) as an incremental learning algorithm for supervised neural networks. In a similar fashion as AdaBoost, Learn++ generates weak hypotheses and then combines the output hypothesis through weighted majority voting. Classifiers are trained on a set of examples that is drawn from a dynamically updated distribution of the training data. The algorithm mainly focuses on hard instances; instances that are repeatedly misclassified are called as hard instances, and forces additional classifiers to be trained with them. Thus Learn++ attempts to produce new classifiers that are better able to predict examples for which the current ensemble's performance is poor. The algorithm of Learn++ is given in Figure 3.2.



Algorithm AdaBoost.M1

Input:

- Sequence of m examples $S=[(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)]$ with labels $y_i \in Y = \{1,...,C\}$ drawn from a distribution **D**,
- Weak learning algorithm **WeakLearn,**
- Integer T specifying number of iterations.

Initialize $D_1(i) = \frac{1}{m}, \forall i$.

Do for $t = 1,2,...,T$:

1. Call **WeakLearn**, providing it with the distribution D_t.
2. Get back a hypothesis $h_t : X \rightarrow Y$
3. Calculate the error of h_t : $\varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$

   If $\varepsilon_t > \frac{1}{2}$, then set $T = t-1$ and **abort loop**.
4. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.
5. Update distribution D_t:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t, & \text{if } h_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$

   where $Z_t = \sum_i D_t(i)$ is a normalization constant chosen so that D_{t+1} becomes a distribution function

Output the final hypothesis:

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

Figure 3.1. AdaBoost.M1 Algorithm (from (Freund and Schapire, 1997))

Learn++ makes some modification in the basic ideas of the AdaBoost to achieve the incremental learning from additional data, without forgetting the previously acquired knowledge, even when the new classes are introduced with the new. Learn++ optimizes the distribution update rule of the AdaBoost to accomplish this task. Also they differ in definition of training error and evaluation of individual hypotheses.

The inputs of the Learn++ algorithm for each dataset D_k, k=1,…,K are

1 – Sequence of m training examples selected from the dataset D_k data with their correct labels; $S_k = [(x_1,y_1),(x_2,y_2),.....(x_m, y_m)]$

2- A classification algorithm base classifier (any supervised classifier)

3- An integer T_k specifying the number of classifiers to be generated

As mentioned before, each classifier is trained over a different subset of the training dataset, so each classifier learns a different part of the data. In the first step, at each iteration, the weights are adjusted and the distribution is obtained by normalizing the weights that are assigned in the previous iteration based on the performance of the classifiers on that instance. In general, the instances that are correctly classified by many of the classifiers get lower weights and the instances which tend to be misclassified get higher weight to increase the probability of being selected in the next iteration. Thus, the algorithm focuses on the examples that seem to be hard to classify. In the first iteration, unless there is a prior knowledge to select otherwise, all the examples are given equal probability to be selected into the training set and the weights are equally initialized to *1/m*. In the next step (step2), training subset (TR_t) and test subset (TE_t) are chosen according to the current distribution. D_t. Then, the base classifier is trained over the training subset (TR_t) to generate a hypothesis (h_t) (step 3).

In the step 4, the error of the generated hypothesis is calculated on the entire dataset $S_k = TE_t + TR_t$:

$$\varepsilon_t = \sum_{i:\mathrm{ht(xi)} \neq yt} \mathrm{D}_t(\mathrm{i}) \tag{3.1}$$

The error is calculated as the sum of the weights of the misclassified instances. Since the base classifier is only expected to have a 50 percent (or better) empirical classification performance to ensure a reasonable performance, the hypothesis h_t is discarded if $\varepsilon_t > 1/2$ and TE_t and TR_t are reselected by returning step2. In case of $\varepsilon_t > 1/2$ constraint is satisfied, the normalized error is calculated as given below (step 4): i :$h_t(x_i)$

$$\beta_t = \varepsilon_t / 1 - \varepsilon_t \quad (0 < \beta_t < 1) \tag{3.2}$$

In the next step, all the hypotheses generated so far are combined using the weighted majority voting in order to obtain the composite hypothesis H_t (Step 5):

$$H_t = \arg\max_{y \in Y} \sum_{t:\, h_t(x)=y} \log \frac{1}{\beta_t}. \tag{3.3}$$

Algorithm Learn++

Input: For each database drawn from $\mathcal{D}_k$ $k=1,2,...,K$

- Sequence of m training examples $S=[(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)]$.
- Weak learning algorithm WeakLearn.
- Integer T_k, specifying the number of iterations.

Do for $k=1,2,...,K$:

Initialize $w_1(i) = D(i) = 1/m$, $\forall i$, unless there is prior knowledge to select otherwise.

Do for $t = 1,2,...,T_k$:

1. Set $\boldsymbol{D}_t = \mathbf{w}_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that $\boldsymbol{D}_t$ is a distribution.
2. Randomly choose training TR_t and testing TE_t subsets according to $\boldsymbol{D}_t$.
3. Call WeakLearn, providing it with TR_t.
4. Get back a hypothesis $h_t : X \rightarrow Y$, and calculate the error of h_t : $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$ on $S_t = TR_t + TE_t$. If $\varepsilon_t > 1/2$, set $t = t-1$, discard h_t and go to step 2. Otherwise, compute normalized error as $\beta_t = \varepsilon_t/(1-\varepsilon_t)$.
5. Call weighted majority, obtain the composite hypothesis $H_t = \arg\max_{y\in Y} \sum_{t:h_t(x)=y} \log(1/\beta_t)$, and compute the composite error $E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i) = \sum_{i=1}^{m} D_t(i)[\![H_t(x_i) \neq y_i]\!]$. If $E_t > 1/2$, set $t = t-1$, discard H_t and go to step 2.
6. Set $B_t = E_t/(1-E_t)$ (normalized composite error), and update the weights of the instances:

$$w_{t+1}(i) = w_t(i) \times \begin{cases} B_t, & \text{if } H_t(x_i) = y_i \\ 1, & \text{otherwise} \end{cases}$$
$$= w_t(i) \times B_t^{1-[\![H_t(x_i)\neq y_i]\!]}$$

Call weighted majority on combined hypotheses H_t and **Output** the final hypothesis:

$$H_{final} = \arg\max_{y\in Y} \sum_{k=1}^{K} \sum_{t:H_t(x)=y} \log\frac{1}{B_t}$$

Figure. 3.2. Algorithm of Learn++ (from Polikar *et al.*, 2001)

The classification decision is made by using this composite hypothesis. H_t chooses the class that receives the highest vote from all hypotheses. The weights of the hypotheses are computed based on their performances on their training dataset. Since the weight of

each hypothesis is calculated as inversely proportional to its normalized error, the hypotheses performing well are given higher weights. The composite error of the H_t is calculated in a similar way given in step-4 as the sum of the weights of the misclassified instances (step 6):

$$E_t = \sum_{i:\, H_t(x_i) \neq y_i} D_t(i) = \sum_{i=1}^{m} D_t(i)[|H_t(x_i) \neq y_i|] \tag{3.4}$$

If the composite error is more than 0.5 ($E_t > 1/2$), the H_t is discarded and new hypothesis is constructed by returning step2. This constraint can only be ignored during the immediate iteration after a new dataset is introduced. If $E_t < 1/2$, the composite error is computed:

$$B_t = E_t / (1-E_t) \tag{3.5}$$

The weights of the instances are then updated to determine the probability of being selected in the next distribution:

$$\begin{aligned} w_{t+1}(i) &= w_t(i) \times \begin{cases} B_t, & if \;\; H_t(x_i) = y_i \\ 1, & otherwise \end{cases} \\ &= w_t(i) \times B_t^{1-[|H_t(x_i) \neq y_i|]} \end{aligned} \tag{3.6}$$

Using this rule, the weights of the instances which are correctly classified by the composite hypothesis are reduces, since the current weight is multiplied by B_t which is less than 1 by its definition. The weights of the misclassified samples remain unchanged since multiplied by 1. Therefore, for the correctly classified instances the probability of being selected in the next training set is reduced. Thus, the algorithm focuses more on the hard instances that are repeatedly misclassified and additional classifiers are trained with these instances. This distribution update rule makes possible to learn incrementally, therefore, it constitutes the heart of the algorithm.

In the last step, the final hypothesis is constructed after T_k hypotheses are generated for each dataset. The final hypothesis is computed by using the weighted majority voting algorithm:

$$H_{final} = \arg\max_{y \in Y} \sum_{k=1}^{K} \sum_{t:H_t(x)=y} \log \frac{1}{B_t} \tag{3.7}$$

Learn++ achieves incremental learning by generating additional classifiers without losing the prior knowledge. Another important property of the algorithm is that it is independent of the base classifiers

In this thesis, we use an incremental learning approach of SVM ensemble by using Learn++. The ensemble is obtained by retraining the SVM base classifiers on the dynamically updated distribution of the training dataset as explained above.

4. CHANGING ENVIRONMENT

Most of the machine learning researches applied to the problems with the assumption that the data is drawn from a stationary distribution. However, in most of real world applications, the data is collected over an extended period of time and the distribution underlying the data is likely to change by time. These changes make the model built on old data inconsistent with the new data, and regular updating of the model is necessary. This problem is known as *concept drift*. A typical example is information filtering, i.e. the adaptive classification of documents with respect to a particular user interest. The interest of the user and the document content may change over time. Another example is online shopping behaviours. The companies can collect the data like sales figures and customer data to find patterns in the customer behaviour to predict future sales and can develop a predictive model. As the customer behaviour tends to change over time, the model should be adapted accordingly. There are many other examples such as detecting and filtering spam email, weather prediction, etc. In order to make time-critical predictions, the model must be able to capture the up-to-date trends. This can be achieved by revising the model by incorporating new examples and eliminating the effect of outdated concepts.

In real world applications the concept drift may occur (1) suddenly or (2) gradually. In natural systems, usually gradual drifts (e.g. seasonal, demographic, habitual, etc) are expected, but sometimes the class description for the problem may change rapidly due to the hidden context (a dependency not given explicitly in the form of predictive features). For example, monetary concerns of someone graduating from college might suddenly change, whereas a slowly wearing piece of factory equipment might cause a gradual change in the quality of output parts.

Hidden changes in context may not only be a cause of a change of target concept, but may also cause a change of the underlying data distribution. Even if only the data distribution changes without any change in the target concept, this may often lead to the necessity of revising the current model, since the model will not perform well on the new data. The necessity in the change of current model due to the change of data distribution is called *virtual concept drift* (Widmer and Kubat, 1993). Virtual concept drift and real

concept drift often occur together. Virtual concept drift alone may occur, e.g. in the case of spam categorization. From the practical point of the current models needs to be changed regardless of the type of the concept drift.

Different strategies should be used to handle the concept drift depending on the type of change. For example, when the underlying hidden context is likely to recur due to cyclic phenomena (e.g. seasonal), to keep the past successful classifiers and reuse them may be an appropriate approach. In the case of gradual changes, one appropriate approach is to use a moving window on the training data. When an abrupt change is detected, retraining of the classifier may be appropriate. Also one problem in handling of the concept drift is to distinguish the noise and the concept drift. Some algorithms may interpret the noise as concept drift, and adjust the model to the noise. An ideal learner should combine robustness to noise and sensitivity to concept drift (Widmer and Kubat, 1996). An ideal concept drift handling system should be able to: (1) quickly adapt to the concept drift;(2) be robust to noise and able to distinguish it from the concept drift, and (3) recognize and reacts to reoccurring contexts, such as seasonal differences (Tsymbal, 2004) . For effective learning in a changing environment, the algorithm should be able to detect context change without explicitly inform about the concept drift and quickly adjust the hypothesis to the current context and can use the previous experience when the old context reappears (Widmer and Kubat, 96).

Three main approaches currently used for concept drift handling are (Tsymbal, 2004):

1) Instance Selection
2) Instance Weighting
3) Ensemble Learning (or learning with multiple concept descriptions)

In *instance selection* approach, the main goal is to select the data that is most relevant to the current concept. The most common instance selection method is to use a sliding window that moves over recently arrived instances. The classifier is updated (retrained) with the most recent training data and the learnt concept is used for classification in the immediate future. Recently arrived instances are added to the window and the old ones are removed (forgotten) from the window. In the simplest case fixed-sized window is used (also called as “time-based forgetting”). If the window size is chosen too small, the model will

quickly react to the changes but the accuracy of the model will decrease due to the insufficient data in the window. On the other hand, choosing a large window may lead to a slow but stable and well trained classifier. The choice of the best window size is a compromise between fast adaptivity and good generalization. So the forgetting parameters should be adjusted. Furthermore, in some algorithms the window size is dynamically adjusted to the current extent of concept drift ("adaptive size"). When a change is detected the window size is reduced, means that old data is forgotten, and in case of a static period the window size is expanded to a predefined size. Examples of window-based algorithms include the FLORA family of algorithms (Widmer and Kubat, 1996), FRANN (Kubat and Widmer, 1994). Some algorithms use a window of fixed size, while others use heuristics to adjust the window size to the current extent of concept drift, e.g. "Adaptive Size" (Klinkenberg, 2004), and FLORA2 (Widmer and Kubat, 1996). Batch Selection of Klinkenberg (2004) may be considered as instance selection as well. Groups of instances ("batches") are considered to be relevant to the target concept if they are well classified by the current model.

Sometimes the importance of the examples does not directly depend on their age. In some cases, the older samples may be more useful than the more recent data points. In the *instance weighting* approaches, a weight is attached to each data point. Instances can be weighted according to their age, and their relevance to the current concept. Therefore a weighting scheme is taken into account to adapt the weights with respect to actual performance of the learner. Klinkenberg (2004) shows that instance weighting techniques handle concept drift worse than analogous instance selection techniques, which is probably due to overfitting the data (Tysmbal, 2004).

Ensemble approach can be appropriate when high accuracy is required and the time is not the primary concern. An ensemble combines the decisions of a set of classifiers to where the each classifier is trained with a subset of the available training dataset. When using ensemble to handle the concept drift, first it must be determined how to choose the subsets which with to train the base classifiers. Then a combination rule must be determined to combine the classifier responses. Finally, a mechanism must be established to cope with the concept drift and forget the less relevant instances. Many different ensemble methods

have been proposed to handle the concept drift. These approaches can be grouped as follows (Kuncheva, 2004):

- *Dynamic combiners(or "horce racing" algorithms)* where the base classifiers are trained in advance and the concept drift is tracked by changing the combination rule
- *Incremental approaches* where the fresh data is used to update the ensemble and forgetting mechanism is used to remove the redundant data from the ensemble

These approaches are not mutually exclusive and combinations of these methods are possible.

In *dynamic combiner* approach, generally, variants of the Weighted Majority algorithm (Littlestone and Warmuth, 1994) are used. The weights of the classifiers are updated base on their performance as compared with the overall ensemble. Hedge β works in the same way as Weighted Majority, but instead of taking majority voting, one classifier is selected from the ensemble and its decision is taken as the decision of ensemble (Freud and Schapire, 1997). Also Winnow is another algorithm that follows the weighted majority algorithm with a different update rule. In the Winnow algorithm, the classifiers predicting correct labels are rewarded by incrementing their weights and the classifiers giving incorrect decisions are demoted by decreasing the weight of the corresponding classifier. Even though the weighted majority voting has been preferred in most of the dynamic combiners, it is also possible to apply other combination rules. In dynamic combiners approach, the base classifiers are not re-trained with new instances, therefore, this approach is not appropriate for the problems where new types/classes are introduced with new data.

In *incremental ensembles*, the ensemble is updated using new data and the forgetting mechanism is incorporated to remove the old/redundant data from the ensemble. Approaches to handle the data to determine how the fresh data is added into the ensemble can be categorized into three groups: (1) reusing data points; (2) filtering and; (3) using data blocks and chunks. The idea for the first approach is that the training samples for the classifiers in the ensemble can be created incrementally. The data points are reused by random sampling with replacement (Breiman, 1999). In filtering approach, the data is selected with a similar approach as in Boosting. The first data points are used to train the first classifier of the ensemble. The data points which the next classifier to be trained on is

filtered so that the samples that are misclassified by the ensemble built so far constitute the half of the training dataset of the corresponding classifier (Breiman, 1999). The most common approach for data selection is the thirds approach; using data blocks and chunks. In this method, it is assumed that the data arrives as blocks of data points at a time. The base classifiers are trained on a chunk of data. The ensemble can be trained on the most recent data chunk, on a set of past blocks or on the whole set of blocks. The data blocks can be fixed size as in (Wang, 2003) or variable size (Stanley, 2003). The illustration of these approaches can be found in Figure 4.1.

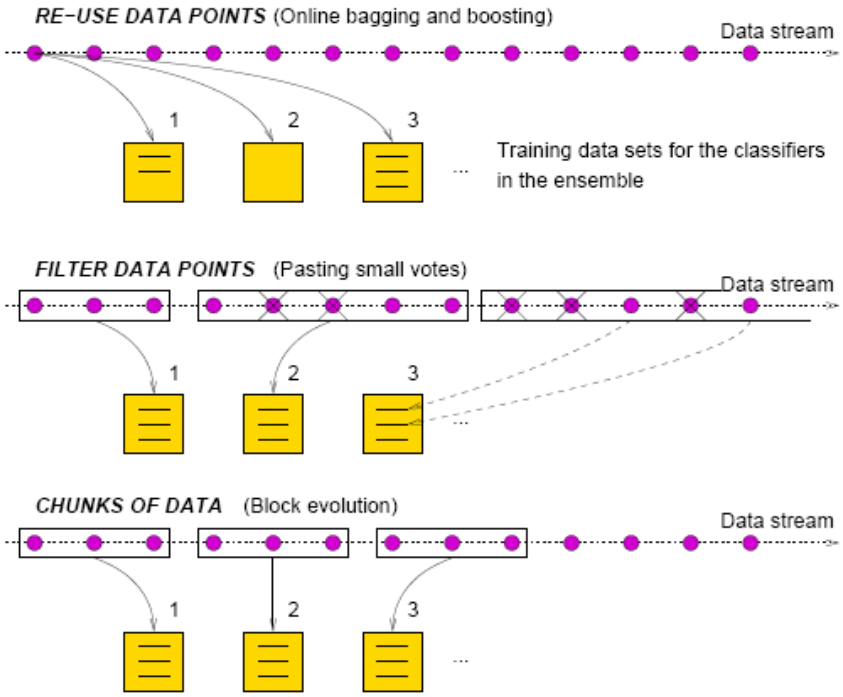


Figure 4.1. Illustration of the three data handling approaches for building online classifier ensembles (from Kuncheva, 2004)

Any *incremental ensemble* approach requires forgetting mechanism when faced with changing environment. The forgetting mechanism is used to determine the classifiers to be removed from the ensemble as the new members are added to adapt to the changing concept. The simplest strategy is to drop the oldest classifier from the ensemble and train a new classifier on the new data to take its place. This strategy is named "*replace the oldest*".

A more complex forgetting strategy is based on the performance of the base classifiers. The classifiers whose error exceeds a certain threshold are dropped from the ensemble. This strategy is called as "*replace the looser*". In (Wang, 2003) only the top K classifiers with the highest performance on the current training data chunk are kept in the ensemble. Also (Stanley, 2003) uses a model where the performance of each member against all seen instances are recorded and the classifiers whose performance falls below a particular threshold are periodically removed. Also (Kolter and Maloof, 2005) proposed a model that a new classifier is added when the ensemble produces an incorrect prediction, and the classifier with the lowest weight, and so with the worst performance, is removed from adding the new member when the number of classifiers reaches a predetermined constant value.

5. RESULTS

In this thesis, we evaluated the ensemble based SVM classifiers, which is constructed based on Learn++ to gain incremental learning capability, on a non-stationary environment. Also, we incorporated forgetting mechanism to see if this improves the performance of the algorithm.

5.1. Evaluated Methods

In order to evaluate the learning approaches for drifting concepts, following methods have been tested on a dataset with a concept drift:

- Learn++ without pruning
- Learn++ with top K classifiers with the highest performance
- Learn++ with replace the looser mechanism

First experiments were done using a SVM ensemble, which is constructed based on Learn++, on a non-stationary environment. Secondly, a forgetting mechanism was added that is only the top K classifiers with the highest performance are kept in the ensemble. Finally, we tried to adapt the approach to changing environment by pruning the classifiers under a predefined threshold.

SVMs were used as base classifiers. In the experiments, we used Gaussian (RBF) and Polynomial SVM kernel functions:

Gaussian Kernel : $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}$

Polynomial Kernel: $K(x_i, x_j) = (x_i \cdot x_j)^d$

The SVM classifier parameters are the regulation parameter C and the RBF width σ (for RBF kernel) and the polynomial degree d (for polynomial kernel). The choice of the classifier parameters is a form of model selection. Since the optimal parameters are domain specific, kernel and regularization parameters were selected jointly to determine the best

model for each dataset. We used 5-fold cross-validation technique to calculate the optimal SVM parameters for each dataset.

5.2. Simulation Datasets

We have performed the tests on two artificial datasets. All the used datasets have two classes. The concept drift was simulated by changing the underlying data generation mechanism. All the test results given in the following sections are the average of ten runs.

5.2.1. Rotated Chessboard Dataset

This data set is a synthetic dataset with two classes and two continuous attributes. The dataset is artificially generated for testing the performance of Learn++ on incremental learning for non-stationary environment. The experiments on this synthetic data used a changing concept based on a rotating chess board. Figure 5.1 illustrated this dataset.

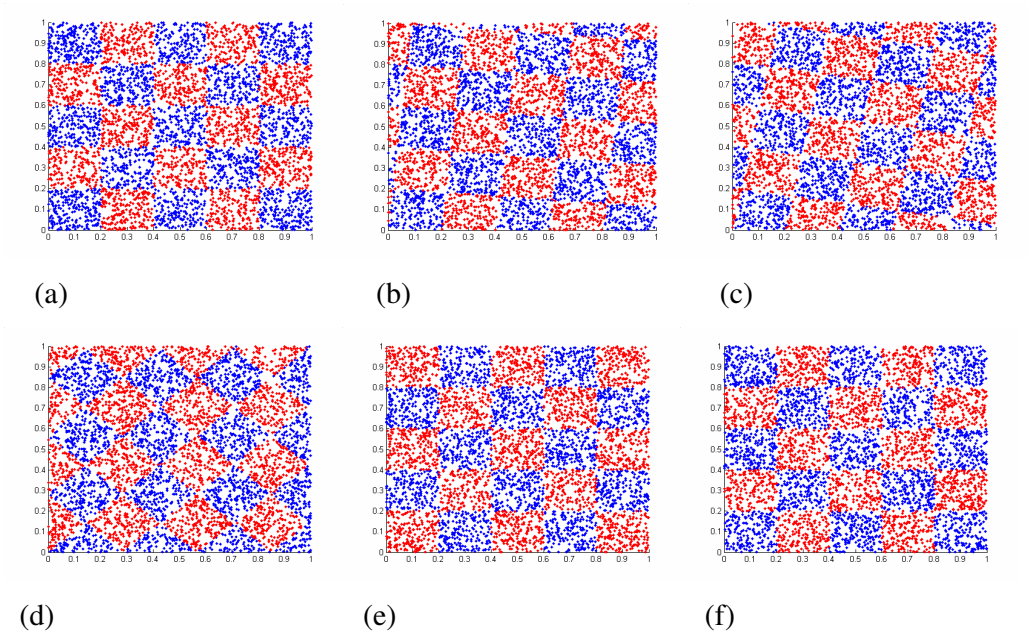


Figure 5.1. The rotated chess board dataset with no rotation (a), 5^{o} rotation (b), 10^{o} rotation (c), 45^{o} rotation (d), 90^{o} rotation (e) and 180^{o} rotation (f).

Each attribute has values uniformly distributed in [0, 1]. The attributes are the coordinates of the examples in a two dimensional space. The dataset was generated by using the below formula:

```
d  = [x.cosα-y.sinα,x.sinα+y.cosα];
s = ceil(d (1)/a)+floor(d (2)/a);
label = 2-mod(s,2);
```

Here *x* and *y* gives the coordinates of the data point, α is the inclination of the chess board data to the origin and *a* is the length of the cells in the chess board.

This dataset has 15 contexts. The dataset is composed of 250 random generated examples in each context and each class is represented by 50% of the examples in each context. We simulated the concept drift by changing the underlying data generation function. First a chessboard dataset with "0" rotation (α) was constructed. The time-changing concepts are simulated by changing the orientation of the chess board in a smooth manner. It is assumed that the data is collected as small batches at a time and the classifiers are trained on a chunk of data. The tests have been performed on a fixed chunk size (250 is the chunk size). To simulate the gradual concept drift, the rotation value is selected as 1^{o}. We also tested the performance on abrupt change by rotating the dataset with 5^{o} degree.

It should be noted that in all simulations, no old data were used in subsequent stages of learning, strictly complying with the notion of incremental learning.

Results on this dataset are presented and discussed in the section 5.3

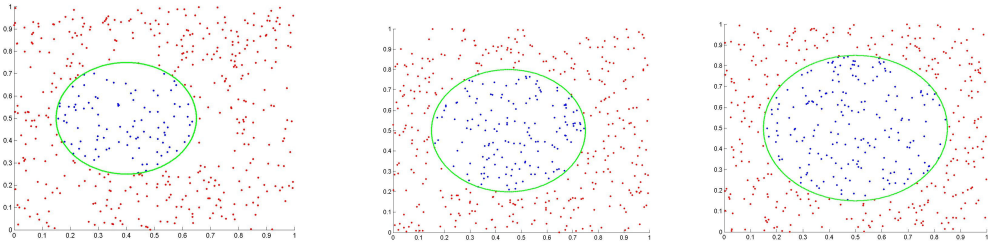
5.2.2. Circles Dataset

This dataset is an artificial dataset as the chess board dataset with two continuous attributes and two classes. Each context defines the strategy to classify the examples by six new classification functions. This dataset has six contexts defined by the following six circles:

Table 5.1. Contexts of Circles dataset

Center	[0.4,0.5]	[0.45,0.5]	[0.5,0.5]	[0.55,0.5]	[06,0.5]	[0.65,0.5]
Radius	0.25	0.3	0.35	0.4	0.45	0.5

Figure 5.2 illustrated this dataset:



DS1 DS2 DS3

Figure 5.2. Circles dataset

Each context is composed of 500 random generated examples. The class distribution in each context is given in the following table:

Table 5.2 Circles data distribution

Dataset	**DS1**	**DS2**	**DS3**	**DS4**	**DS5**	***Test***
Class1	388	363	297	243	190	*150*
Class2	*112*	*137*	*203*	*257*	*310*	*350*

5.3. Simulation Results

5.3.1. Results on Rotated Chessboard Dataset

For each training session only one data chunk was used. This means that only chunk1 was used in the first training session, only chunk2 was used in the second training session, and so on. For each raining session $t = 1, 2, 3, \ldots k$ SVM classifiers were generated by Learn++. Each hypothesis of the k^{th} training session was generated using a training subset TR_t and testing subset TE_t drawn from chunk k.

First experiments were done for Learn++ with SVM base classifiers without forgetting.

Figure 5.3 shows the classification performance of the Learn++ on the test data. According to Figure 5.3, the performance on the test set steadily improved. The gradual increase in the performance of the algorithm on the test data demonstrates the incremental learning capability of the Learn++ algorithm.

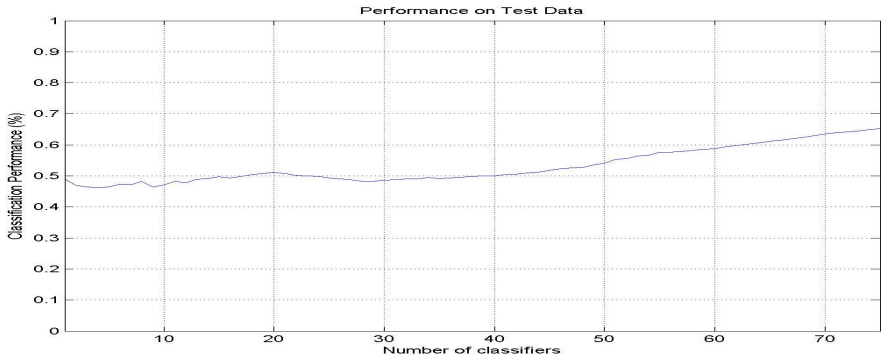


Figure 5.3. Performance graphic on chessboard data with 1^o rotation with RBF kernel (σ=40, C=20) without forgetting

Also table 5.3 and 5.4 shows the performance of the algorithm for different number of classifiers. Each column shows the number of the base classifiers trained on one training session. In the first column, two classifiers are constructed for each chunk, and since we have 15 chunks, 30 classifiers constructed for that experiment. There are two parts in the table. The upper part shows the results for 1^o rotation value. The below part of the table gives the results for 1^o rotation. The first row shows the generalization performance of the algorithm on rotating chessboard dataset with 1^o rotation. The second row gives the elapsed time for training.

Table 5.3 SVMLearn++ with RBF kernel (σ=40, C=20) on chessboard

#classifiers	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
(%)Gen.	64,76	64,86	65,56	65,34	65,80	65,40	65,20	66,28	65,72
Time (sn)	4,06	7,65	12,25	17,96	24,9	32,6	41,6	51,79	62,98
(%)Gen.	51,52	52,00	51,92	51,52	51,88	51,76	51,44	51,88	51,88
Time (sn)	4	7,46	12,05	17,6	24,35	32,4	43,4	37	71,6

Table 5.4. SVMLearn++ with Polynomial kernel (degree=5, C=15) on chessboard

# classifiers	2	3	4	5	6	7	8	9	10
(%)Gen.	50,64	52,08	54,24	57,20	54,96	52,32	50,32	53,60	50,20
Time (sn)	7,2	12,6	15,2	18,4	27,2	41,2	50,4	63,5	72,18
(%)Gen.	50,40	54,00	47,58	52,00	50,20	49,89	50,40	50,20	51,08
Time (sn)	7	14,6	16,8	21,2	29,8	43,8	48,5	55,6	73,6

The results provided in table 5.3 and table 5.4 show that there is no direct relation between the number of classifiers and the performance of the algorithm. The performances are all around ~65% (for RBF) and ~52% (for polynomial) regardless of the classifier number. Also it is clearly seen that the time complexity is directly affected from the classifier numbers. The time complexity increases when increasing, as expected, even though there is no improvement in the generalization performance of the model.

When we compare the results for 5^o rotation and 1^o rotation, it is seen that the performance clearly decreases especially in RBF kernel. For RBF kernel the performance decreases from 65% to 51%. For polynomial kernel the performance is just below 50% for both rotation values. RBF kernel performs better than polynomial kernel on this dataset.

Secondly, we tested the algorithm by incorporating a forgetting approach. In this approach, only the top K classifiers with best performance are kept and others are discarded. K is a predetermined constant value. We implemented this approach by replacing the worst classifier with the new classifier when the number of classifiers reaches this predetermined constant value. This approach is used to discard the redundant old knowledge to adapt to the new concept. The results for different K values are given in following tables.

Table 5.5. SVMLearn++ with RBF kernel on chessboard with top K classifiers

#classifiers	5	10	15	20	25	30	35	40	45	50
(%)Gen.	82,08	82,40	81,92	79,44	78,68	77,24	75,65	74,24	72,40	71,92
Time (sn)	12,8	15,43	17,84	19,98	23,7	23,5	24,93	26,07	33	32,23
(%)Gen.	58,76	59,88	57,52	55,04	53,00	52,44	53,92	53,92	54,48	54,68
Time (sn)	12,68	15,27	18	19,71	21,67	23,16	24,48	25,47	47,16	33,25

Table 5.6. SVMLearn++ with Polynomial kernel (degree=5, C=15) on chessboard with top K classifiers

# classifiers	5	10	15	20	25	30	35	40	45	50
(%)Gen.	57,60	54,72	55,04	56,16	54,96	53,60	50,20	54,60	53,10	52,10
Time (sn)	13,8	18,6	21,6	25,3	28,7	32,5	48,6	59,52	64,35	78,8
(%)Gen.	50,64	52,08	54,24	57,20	54,96	52,32	50,32	53,60		50,20
Time (sn)	7,2	12,6	15,2	18,4	27,2	41,2	50,4	63,5		72,18

As shown in table 5.5, the number of classifiers has an important role in this approach. From the above table, it can be clearly seen that the performance of the model decreased for very large and very small classifier numbers. When the number of classifiers is too small, this means that also the relevant knowledge is discarded, the performance decreases probably due to the insufficient learning. On the other hand, when the number of classifiers is too large, that is forgetting is not enough so the algorithm is similar as the fist approach that no forgetting is incorporated, the performance decreased due to the irrelevant knowledge. Therefore, the choice of the classifier number is important, and there is a trade-off between forgetting and learning from the data.

Also, when comparing the performances with the pervious approach, Learn++ without forgetting, performance of the model improved from ~65% to ~ 80% (for RBF kernel) for an appropriate selection of the K value (classifier number). Furthermore, when we investigate the performance graphics of the proposed approaches, on the test data, (given with Figure 5.3 and Figure 5.4.), even though the graphics have similar characteristics, the inclination of the graphics shows that the second approach could adapt to the concept drift more quickly than the first approach.

Finally, another forgetting approach is used to adapt the Learn++ algorithm to the changing environment. In this approach, the classifiers whose performance falls below a certain threshold are dropped from the ensemble. The results for different threshold values are given in table 5.7, and the performance graphic is given in figure 5.5.

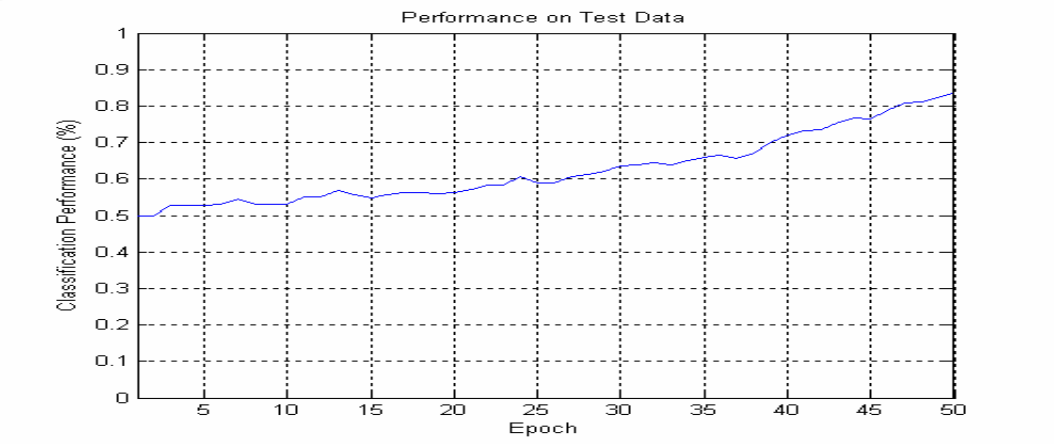


Figure 5.4. Performance graphic on chessboard data with 1^{o} rotation with RBF kernel (σ=40, C=20) for top K classifiers

Table 5.7. SVMLearn++ with RBF kernel (σ=40, C=20) on chessboard for different threshold values

Threshold	**60**	**65**	**70**	**75**	**80**	**85**	**90**	**95**
(%)Gen.	78,20	79,64	81,16	82,44	82,96	81,96	78,24	59,80
#classifiers	30,2	27,5	20	13,4	6,1	4,8	2	2
Time (sn)	22,39	20,95	19,05	16,89	14,37	13,40	12,78	11,92
(%)Gen.	60,04	60,60	67,00	61,44	59,96	59,76	59,48	50,84
#classifiers	10	9,4	6,7	5	5	5	3,9	2
Time (sn)	15,11	14,05	13,41	13,31	13,23	13,19	12,71	11,93

Provided results in the above tables show that the selection of the threshold value directly affects the classification performance of the algorithm. If a high threshold is selected, the number of classifiers decreased since there are many classifiers that cannot reach this performance so the valid knowledge can be discarded. On the other hand, when selected threshold value is too low, the number of classifiers increases, which causes time complexity and the classification performance decreases due to the redundant data. When comparing the performance results, it is clear that incorporating this forgetting mechanism improved the classification performance. By using this method, a better performance was obtained with less classifier. However, as similar with the previous two approaches, this method can handle only gradual changes. When the chess board is rotated by 5^{o} instead of 1^{o}, the performance results are not promising.

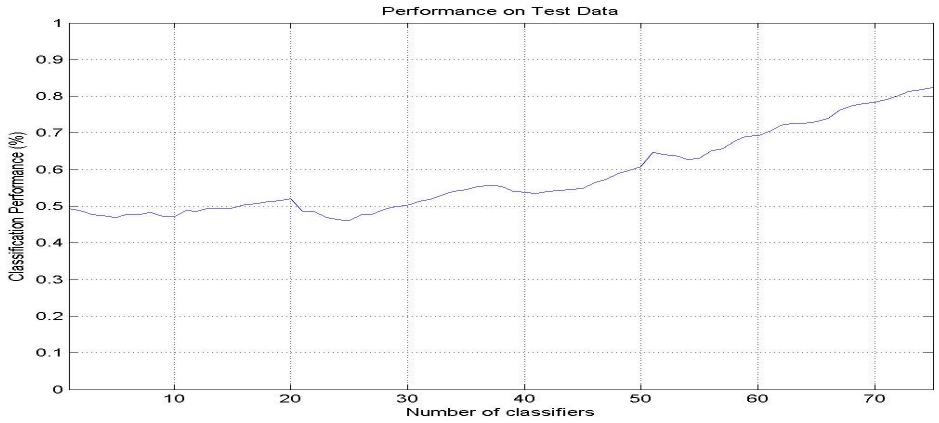


Figure 5.5. Performance graphic on chessboard data with 1^{o} rotation with RBF kernel (σ=40, C=20) with incorporating threshold approach

5.3.2. Results on Circles Dataset

Same approaches described in the previous section are also tested on Circles dataset. In circles dataset there are 5 different contexts with different radius and center coordinates. Each dataset contains 500 data points and the chunk size used in the tests is 250. The first tests were performed without incorporating forgetting mechanism.

Table 5.8. SVMLearn++ with RBF kernel (σ=5, C=10) on circles

#classifiers	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
(%)Gen.	70,48	70,50	70,16	71,08	71,42	70,26	71,80	73,52	71,28
Time (sn)	2,23	3,94	6,44	9,1	12,84	20,76	21,67	26,33	32,35

Table 5.9. SVMLearn++ with Polynomial kernel (degree=10, C=0.1) on circles

# classifiers	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
(%)Gen.	65,72	65,92	67,64	66,96	67,20	66,70	67,10	67,00	66,72
Time (sn)	4,57	13,6	14,2	23	29,1	38	47,8	61,62	78,2

We obtained similar results as on the chessboard dataset. The performances are all around ~71% (for RBF) and ~66% (for polynomial) regardless of the classifier number, but the time complexity increased proportional with the classifier number. As shown in the

below graphic, the performance of the model steadily increases. This means that the SVM with Learn++ can handle new data.

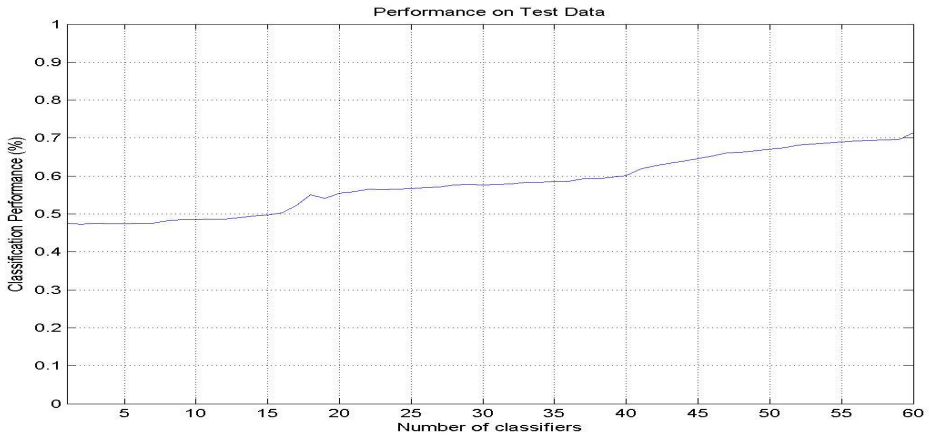


Figure 5.6. Performance graphic on circles data with RBF kernel (σ=5, C=10)

Table 5.10. SVMLearn++ with RBF kernel on circles with top K classifiers

#classifiers	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
(%)Gen.	90,56	91,60	91,40	82,96	82,00	81,24	80,40	77,16	73,56	70,00
Time (sn)	9,5	13,21	14,46	15,45	16,68	16,99	17,55	17,6	17,64	20,48

Table 5.11. SVMLearn++ with polynomial kernel on circles with top K classifiers

#classifiers	**5**	**10**	**15**	**20**	**25**	**30**	**35**	**40**	**45**	**50**
(%)Gen.	90,48	91,24	90,64	83,84	82,20	82,12	80,68	67,60	68,60	67,80
Time (sn)	14,4	18,2	21,8	24,6	26	28,16	29,4	30,1	30,375	29,43

Table 5.10 and 5.11 includes the results when only the best top K classifiers are selected. When we compare the results for SVMLearn++ without forgetting and with selecting best K classifiers, it is clearly seen that incorporating the forgetting mechanism extremely improves the performance of the model both RBF and polynomial kernels. This provides ~25-30% improvement in generalization performance without increasing the time complexity. Furthermore, since the number of classifiers is limited by the algorithm, it also decreases the time complexity. Also we can see in figure 5.7, forgetting provides more quick adaptation to the concept drift.

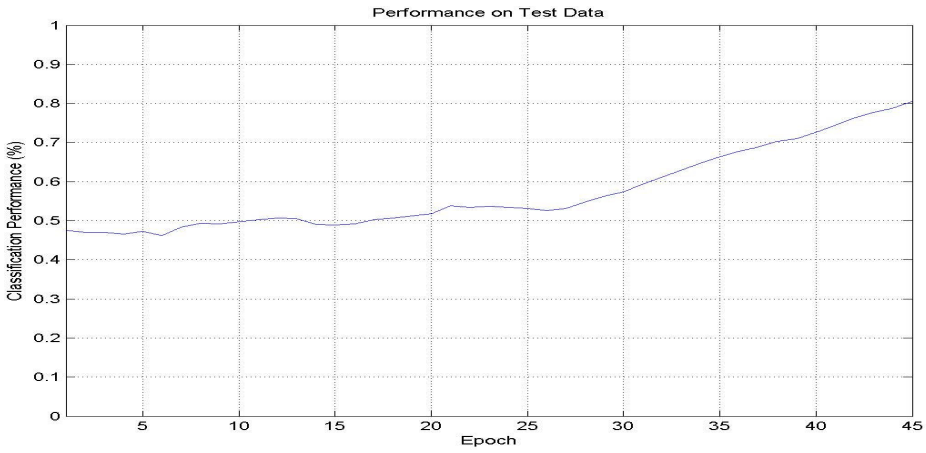


Figure 5.7. Performance graphic on circles data with RBF kernel (σ=5, C=10) with top K classifiers

Table 5.12. SVMLearn++ with RBF kernel on circles for different threshold values

Threshold	**60**	**65**	**70**	**75**	**80**	**85**	**90**	**95**
(%)Gen.	77,16	80,28	81,40	81,24	90,24	88,28	90,64	90,44
#classifiers	40	18,3	15	14,9	10	9,8	5	4,7
Time (sn)	17,65	17,5	16,99	16,76	15,69	14,61	13,39	12,79

Table 5.13. SVMLearn++ with Polynomial kernel on circles for different threshold values

Threshold	**60**	**65**	**70**	**75**	**80**	**85**	**90**	**95**
(%)Gen.	74,00	79,76	81,12	88,24	90,44	89,60	90,92	90,52
#classifiers	19,9	17,5	15	12,9	10	9,7	5	4,6
Time (sn)	31	27,8	31,2	26,8	25,6	22,4	18,35	20,2

We obtain similar results both for best K classifiers and classifiers perform better than a predetermined threshold. In both cases, the classification performance is improved in a changing environment. Forgetting irrelevant knowledge provides quick adaptation to changing concept and improves the classification performance. The performance graphic for the last method is given in figure 5.8.

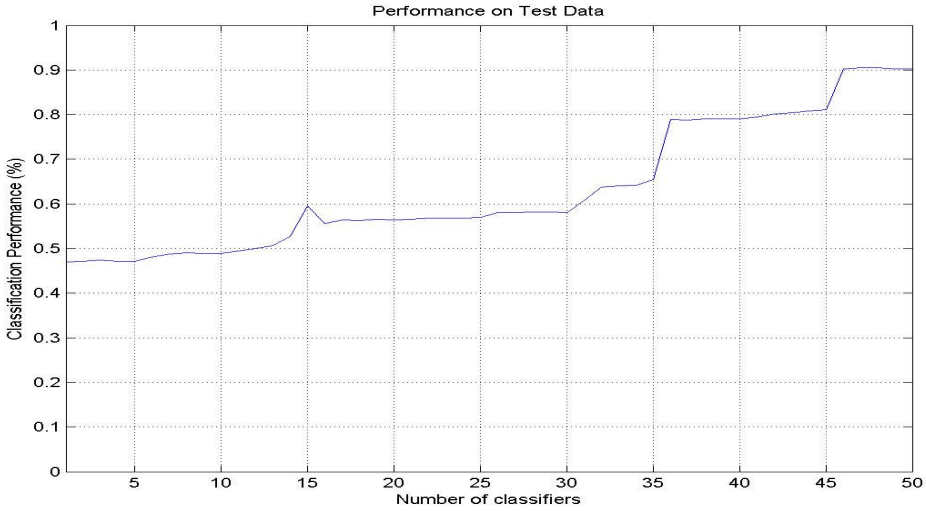


Figure 5.8. Performance graphic on circles data with RBF kernel when incorporating threshold mechanism

6. CONCLUSIONS

In this thesis we have proposed incorporating a forgetting mechanism into the incremental learning algorithm with SVM ensemble, which is constructed with Learn++ algorithm, to adapt the algorithm to the changing environment. Simulations have been performed on two synthetic datasets and the concept drift has been simulated by changing the underlying data generation mechanism by updating some parameters.

We first examined the performance of the Learn++ with SVM base classifiers on changing environment. Experimental results demonstrate that even though the performance of the algorithm gradually increases on the test data during the training, the overall results are not promising.

In the next stage, we examined the effect of pruning the ensemble such that only top K classifiers with highest performance are kept in the ensemble. The results show that the overall performance of the model is improved by choosing an appropriate K value. Choosing a too low K value may result in elimination of relevant data, so the performance of the model may decrease. On the other hand, choosing a too high K value may cause a decrease in the classification performance of the model due to an irrelevant old data. Therefore, the K value should be selected considering these issues.

Finally, we used another forgetting strategy that is removing the classifiers whose classification errors exceed a predetermined threshold value. As similar in the pervious method, using this forgetting strategy improves the classification performance on changing environment. The value of the threshold also affects the performance of the model. The threshold value should be selected considering the trade-off between forgetting and learning from the new data. Also when comparing two forgetting approaches used in this thesis, there is not a significant difference between the resulting performances, both provide similar performances.

Simulations have been performed for different rotation values of the data. For all of the mentioned three approaches, the performance of the model in case for the abrupt change

is too low comparing with the performance for a gradual change. Even though it is obvious that applying a forgetting strategy improves the performance, it does not provide promising results for abrupt changes, therefore, another approach should be applied in case of abrupt change.

Various additional improvements to the algorithm can be proposed as future work. As described before, different approaches can be used to handle different types of concept drift. The performance of the algorithm can be examined for different types of concept drift (e.g. abrupt change, recurring contexts, etc.) and different forgetting approaches, different distribution update rules or different combining algorithms can be used to cope with specific types of concept drift.

7. REFERENCES

Bousquet,O.,S. Boucheron and G. Lugosi , 2004, "Introduction to Statistical Learning Theory", In *Advanced Lectures in Machine Learning*, LNAI 3176, Springer-Verlag.

Breiman, L, 1999, "Pasting small votes for classification in large datbaes and online", *Machine Learning,* 36(1-2):85-103.

Burbidge, R. and B. Buxton ,2001, An Introduction to Support Vector Machines for Data Mining. *Keynote Papers, Young OR12.*

Burges , C.J.C., 1998, "A tutorial on support vector machines for pattern recognition", *DataMining and Knowledge Discovery*, vol. 2, no. 2, pp. 1-47.

Campbell, C., 2002, "Kernel methods: a survey of current techniques", *Neurocomputing*, *48*, 63-84.

Chang, C. –C and Ling C. –J., 2001, "LIBSVM: A library for support vector machines", http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Cristianini, N. and S. John, 2000, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press.

Delany, S.J., P Cunningham & A Tysmbal, 2006, "A comparison of Ensemble and Case-base Maintenance Techniques for Handling Concept Drift in Spam Filtering", *Proceedings of FLAIRS.*

Dietterich, T.G., 1997, "Machine Learning research: Four current directions", *AI Magazine*, 18(4): 97-136.

Dietterich, T. G., 2000, "Ensemble methods in machine learning", In *Proceedings of the First International Workshop on Multiple Classifier Systems* (2000), Springer Verlag, pp. 1-15.

Domingos, P., G. Hulten, L. Spencer, 2001, "Mining Time-Changing Data Streams", Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining (pp. 97-106). San Francisco, CA: ACM Press.

Erdem, Z., R. Polikar, F. Gürgen, N. Yumuşak, 2005,"Reducing the Effect of Out-voting Problem in Ensemble Based Incremental Support Vector Machines", International Conference on Artificial Neural Networks (ICANN 2005) 11-15 September, Warsaw, Poland.

Erdem, Z., R. Polikar., F. Gürgen, N. Yumusak, 2005,"Ensemble of SVMs classifier for incremental learning", *Multiple Classifier Systems Lecture Notes in Computer Science,*3541, 246-256, (SCIE).

Evgeniou, T., M. Pontil and T. Poggio, 2000, "Statistical Learning Theory : A Primer"*, International Journal of Computer Vision*, vol. 38, no. 1, pp. 9-13.

Freund, Y. and R. Schapire, 1996,"Experiments with a new boosting algorithm," *Proc. of the13th International Conference on Machine Learning,* pp.148-156.

Freund, Y. and R. Schapire, 1997, "A decision theoretic generalization of on-line learning and an application to boosting," *Computer and System Sciences,* vol. 57, no. 1, pp. 119-139.

Hall, O., K.B Bowyer, Bhadoria, D., 2004, "A Comparison of Ensemble Creation Techiques", The Fifth International Conference on Multiple Classifier Systems, June.

Kecman, V., 2004,Support Vector Machine Basics, School of Engineering Report.

Kelly, M. G., D. J. Hand & N.M Adams, 1999, "The impact of changing populations on classifier performance", In Proc 5th ACM SIGDD International Conference on Knowledge Discovery and Data Mining.

Klinkenberg, R. and T. Joachims, 2000, "Detecting Concept Drift with Support Vector Machines", ICML 2000: 487-494.

Klinkenberg, R. ,2001, "Using Labeled and Unlabeled Data to Learn Drifting Concepts", in 'Workshop notes of the IJCAI-01 Workshop on Learning from Temporal and Spatial Data', AAAI Press, Menlo Park, CA, USA, pp. 16–24.

Klinkenberg, R. and Rüping, S., 2003, "Concept Drift and the Importance of Examples", in: Text Mining - Theoretical Aspects and Applications, J. Franke, G. Nakhaeizadeh, I. Renz (Hrsg.), Physica-Verlag, Berlin, 55-77.

Klinkenberg, R., 2004, "Learning Drifting Concepts: Example selection vs. example weighting", *Intelligent Data Analysis,*8(3).

Kolter, J. Z.and M. A. Maloof, 2005, "Using additive expert ensembles to cope with concept drift", *ICML*: 449-456.

Kotsiantis, S. B. and P.E. Pintelas, 2004, "An online ensemble of classifiers", *The Fourth International Workshop on Pattern Recognition in Information Systems -PRIS, In conjunction with 6th International Conference on EnterpriseInformation Systems, Porto – Portuga*l.

Kubat, M. and G. Widmer, 1994, "Adapting to drift in continuous domains", *Technical Report OFAI-TR-94-27, Austrian research Institude for Artificial Intelligence, Vienna.*

Kuncheva , L. I., 2004, "Classifier Ensembles for Changing Environment", 5th Int. Workshop on Multiple Classifier Systems.

Littlestone, N. and M.K. Warmuth, 1994, "The weighted majority algorithm", *Information and Computation,* 108(2): 212-261.

Muhlbaier, M., A. Topalis, R. Polikar, 2004, "Learn++.MT: A New Approach to Incremental Learning", 5th Int. Workshop on Multiple Classifier Systems, Springer LNCS Vol. 3077, 52-61.

Polikar, R., L.Udpa, S. Udpa, V. Honavar, 2001, "Learn++: An incremental learning algorithm for supervised neural networks", IEEE transactions on Systems, Man and cybernetics,Part C: Applications and Reviews, Vol. 31, No. 4, pp. 497-508.

Roli, F., 2005, "Semi-supervised Multiple Classifier Systems: Background and Research Directions", MCS 2005, LNCS 3541, pp.1-11.

Scholtz, M. and R. Klinkenberg, 2005, "An Ensemble Classifier for Drifting Concepts", in: *proceedings of the 2nd InternatioanlWorkshop on Knowledge Discoverty from Data Streams*.

Stanley, K., 2003, "Learning concept drift with a committee of decision trees", Technical Report UTAI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA.

Tsymbal, A., 2004, "The problem of concept drift: definitions and related work", Technical Report TCD-CS-2004-15, Computer Science Department, Trinity College Dublin, Ireland.

Valentini, G. and T.G. Dietterich, 2002, "Bias-Variance Analysis and Ensembles of SVM", Multiple Classifier Systems : 222-231.

Vapnik, V., 1995, The Nature of Statistical Learning Theory, Springer-Verlag, New York.

Vapnik, V.,1998, *Statistical Learning Theory*, John Wiley, New York.

Wang, H., 2003, "Mining Concept-Drifting Data Streams using Ensemble Classifiers*", 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Aug., Washington DC, USA.

Widmer G., Kubat M., 1993, “Effective learning in dynamic environments by explicit context tracking”, *Proc. 6th European Conf. on Machine Learning ECML*, Springer-Verlag, Lecture Notes in Computer Science 667, 227-243.

Widmer, G. and M. Kubat, 1996, “Learning in the Presence of Concept Drift and Hidden Context”, Machine Learning, (23), pp. 69-101.