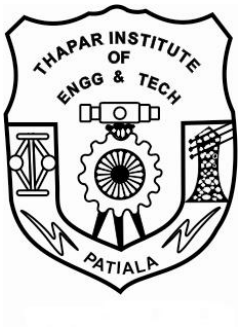
Analysis of E. coli Promoters Using Support Vector Machine

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering



By
Jasneet Singh Taneja
(8043111)

Under the supervision of
Mr. R S Salaria
Assistant Professor
Computer Science & Engineering Department

MAY 2006

COMPUTER SCIENCE & ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA – 147004

Declaration

I hereby certify that the work which is being presented in the thesis entitled **"Analysis of E. coli Promoters Using Support Vector Machine"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science & Engineering Department of Thapar Institute of Engineering & Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of Mr. R S Salaria. The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Jasneet Singh Taneja

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Mr. R S Salaria
Assistant Professor
Computer Science & Engineering Department
Thapar Institute of Engineering & Technology
Patiala

Dr. Seema Bawa
Head
Computer Science & Engineering. Department
Thapar Institute of Engg. & Tech.,
Patiala

Dr. T P Singh
Dean
Academic Affairs
Thapar Institute of Engg. & Tech.,
Patiala

Acknowledgement

I wish to express my deep gratitude to my guide Mr. R S Salaria, Assistant Professor, Computer Science & Engineering Department for his valuable guidance and support throughout the preparation of the thesis.

I am also thankful to Dr. Seema Bawa, Head, Computer Science & Engineering Department, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank all the staff members and my co-students who were always there at the need of the hour and provided me with all the help and facilities, which I required for the completion of the thesis.

Finally I am very grateful to my friend, Mr. Somitra Sanadhya, Research Scholar, ISI Calcutta, who spared his valuable time to help me at every moment I needed his expertise.

Jasneet Singh Taneja

(8043111)

Abstract

Support Vector Machines (SVM) is a family of learning algorithms which is currently considered as one of the most efficient methods in many real-world applications. The theory behind SVM was developed in the sixties and seventies by Vapnik and Chervonenkis, but the first practical implementation of SVM was only published in the early nineties. Since then the method gained more and more attention among the machine learning community, thanks to its ability to outperform most other learning algorithms (including neural networks) in many applications. As a result it has been successfully applied to all sorts of classification issues, ranging from handwritten character recognition to speaker identification or face detection in images.

SVM have been applied to many biological issues, including gene expression data analysis or protein classification. Some claim that biological data mining applications are one of the most promising uses of SVM, particularly for the high dimensionality of the data. As a result, research about SVM and computational biology is the object of much effort today, mainly due to researchers coming from the machine learning community. One can expect SVM to become a standard tool for bioinformaticians in the near future. Recently, the prediction of promoters has attracted many researchers' attention. Unfortunately most previous prediction algorithms did not provide high enough sensitivity and specificity. This is where SVM clearly stands out of the crowd.

Our main idea is to use computer power to calculate all possible patterns which are the possible features of promoters (training of SVM). Once this is done, it will be capable enough to determine whether a testing sequence is a promoter or not. In most of the practical applications, SVM consists of a kernel function that maps the data into a high dimensional feature space, which we will study in sufficient detail in this report. There are many types of kernels being used, viz. linear, polynomial,

additive, splines, gaussian, etc. In this thesis, the intention is to understand the basics and working of SVM, and to explore the performance of different kernels for promoter recognition problem. By the experimental results, radial basis function (Gaussian kernel) proves to be way ahead of others in promoter prediction problem.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

With increasing amounts of data being generated by businesses and researchers there is a need for fast, accurate and robust algorithms for data analysis. Improvements in databases technology, computing performance and artificial intelligence have contributed to the development of intelligent data analysis. The primary aim of data mining is to discover patterns in the data that lead to better understanding of the data generating process and to useful predictions. Examples of applications of data mining include detecting fraudulent credit card transactions, character recognition in automated zip code reading, and predicting compound activity in drug discovery. Real-world data sets are often characterized by having large numbers of examples, e.g. billions of credit card transactions and potential 'drug-like' compounds; being highly unbalanced, e.g. most transactions are not fraudulent, most compounds are not active against a given biological target; and, being corrupted by noise. The relationship between predictive variables, e.g. physical descriptors, and the target concept, e.g. compound activity, is often highly non-linear. One recent technique that has been developed to address these issues is the support vector machine. The support vector machine has been developed as robust tool for classification and regression in noisy, complex domains. The two key features of support vector machines are generalization theory, which leads to a principled way to choose a hypothesis; and, kernel functions, which introduce non-linearity in the hypothesis space without explicitly requiring a non-linear algorithm.

1.1 Motivation

The best way of solving a particular problem is to apply all available domain knowledge and spend a considerable amount of time, money and effort in building a rule system that will give the right answer. The second best way of doing anything is to learn from experience. Given the increasing quantity of data for analysis and the variety and complexity of data analysis problems being encountered in business, industry and

research, it is impractical to demand the best solution every time. The ultimate dream, of course is to have available some intelligent agent that can pre-process your data, apply the appropriate mathematical, statistical and artificial intelligence techniques, and then provide a solution and an explanation. In the meantime we must be content with the pieces of this automatic problem solver. It is the purpose of the data miner to use the available tools to analyze data and provide a partial solution to a business problem. The data mining process can be roughly separated into three activities: pre-processing, modeling and prediction, and explaining. Here, we are more concerned with the intermediate step, viz. prediction. The SVM for two-class classification is dealt with in sufficient detail and some practical issues discussed.

1.2 Machine Learning

The general problem of machine learning is to search a, usually very large, space of potential hypotheses to determine the one that will best fit the data and any prior knowledge. The data may be labeled or unlabelled. If labels are given then the problem is one of *supervised learning* in that the true answer is known for a given set of data. If the labels are categorical then the problem is one of *classification*, e.g. predicting the species of a flower given petal and sepal measurements. If the labels are real-valued the problem is one of *regression*, e.g. predicting property values from crime, pollution, etc. statistics. If labels are not given then the problem is one of *unsupervised learning* and the aim is characterize the structure of the data, e.g. by identifying groups of examples in the data that are collectively similar to each other and distinct from the other data.

1.2.1 Supervised Learning

Given some examples we wish to predict certain properties, in the case where there are available a set of examples whose properties have already been characterized the task is to learn the relationship between the two. One common early approach was to present the examples in turn to a learner. The learner makes a prediction of the property of interest, the correct answer is presented, and the learner adjusts its hypothesis accordingly. This is known as learning with a teacher, or supervised learning.

In supervised learning there is necessarily the assumption that the descriptors available are in some way related to a quantity of interest. For instance, suppose that a bank wishes to detect fraudulent credit card transactions. In order to do this some domain knowledge is required to identify factors that are likely to be indicative of fraudulent use. These may include frequency of usage, amount of transaction, spending patterns, type of business engaging in the transaction and so forth. These variables are the predictive, or independent, variables x. It would be hoped that these were in some way related to the target, or dependent, variable y. Deciding which variables to use in a model is a very difficult problem in general; this is known as the problem of feature selection. Many methods exist for choosing the predictive variables; if domain knowledge is available then this can be very useful in this context.

1.2.2 Unsupervised Learning

Unsupervised learning studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns. By contrast with *supervised learning* or *reinforcement learning*, there are no explicit target outputs or environmental evaluations associated with each input; rather the unsupervised learner brings to bear prior biases as to what aspects of the structure of the input should be captured in the output. The only things that unsupervised learning methods have to work with are the observed input patterns , which are often assumed to be independent samples from an underlying unknown probability distribution, and some explicit or implicit *a priori* information as to what is important.

1.3 Example SVM

SVMs are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are

RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).

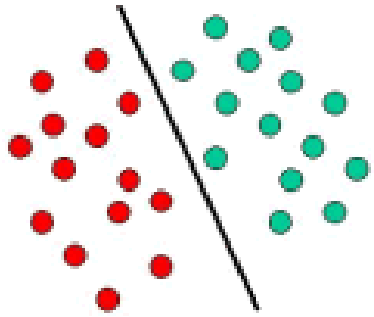


Figure 1.1 Linear separator

The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is depicted in the illustration below. Compared to the previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

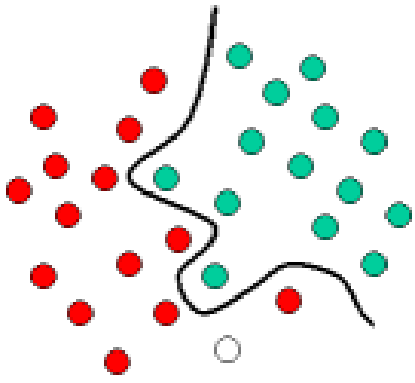


Figure 1.2 Non linear separator

The illustration below shows the basic idea behind Support Vector Machines. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.

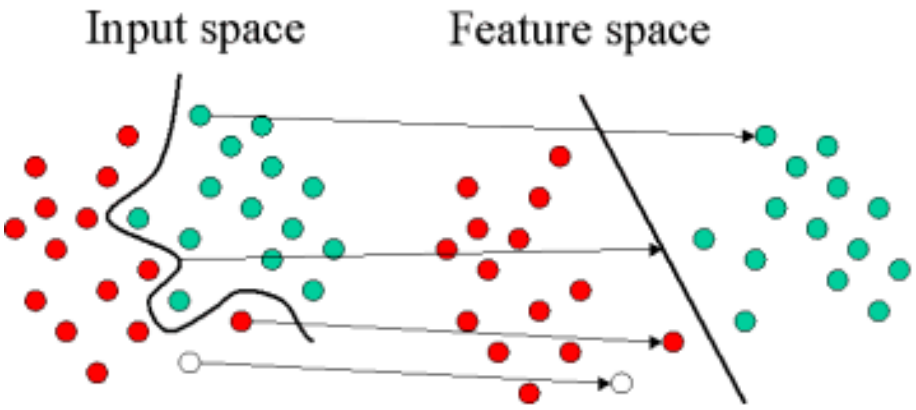


Figure 1.3 Mapping from input space to feature space

1.4 Brief Working of SVM

Support vector machine is a method of obtaining the optimal boundary of two sets in a vector space independently on the probabilistic distributions of training vectors in the sets. Its fundamental idea is very simple; locating the boundary that is most distant from the vectors nearest to the boundary in both of the sets. This simple idea is a traditional one, however, recently has attracted much attention again. This is because of the introduction of *kernel method,* which is equivalent to a transformation of the vector space for locating a nonlinear boundary.

We assume at first a linearly separable problem, as shown in *Figure 1.1*. Our aim is finding the "optimal" boundary hyperplane which exactly separates one set from the other. Note that our "optimal" boundary hyperplane should classify not only the training vectors, but also unknown vectors in each set.

However, an accurate estimation is difficult since the dimension of vectors is often much higher than the number of training vectors. It is referred as “curse of dimensionality”.

Now we try another simple approach without any estimation of distribution. In this approach, the “optimal” boundary is defined as the most distant hyperplane from both sets. In other words, this boundary passes the “midpoint” between these sets. Although the distribution of each set is unknown, this boundary is expected to be the optimal classification of the sets, since this boundary is the most isolated one from both of the sets. The training vectors closest to the boundary are called *support vectors*.

The above discussion is applicable to the case of linearly separable sets only. If the sets are not linearly separable, a hyperplane exactly classifying the sets does not exist, as explained earlier. The *kernel method* explained here is a method of finding truly nonlinear boundaries.

The fundamental concept of kernel method is a deformation of the vector space itself to a higher dimensional space. Consider the linearly non-separable example as shown in *Figure 1.1*. If the two-dimensional space is transformed to the three-dimensional one as shown in *Figure 1.3(right)*, “green” vectors and “red” vectors are linearly separable.

1.5 Goal

Currently, most approaches to the computational analysis of gene expression data attempt to learn functionally significant classifications of genes in an *unsupervised* fashion. A learning method is considered unsupervised if it learns in the absence of a teacher signal that provides prior knowledge of the correct answer. Existing gene expression analysis methods begin with a definition of similarity (or a measure of distance) between expression patterns, but with no prior knowledge of the true functional classes of the genes. Genes are then grouped using a clustering algorithm.

Support vector machines (SVMs) [15, 4, 12] and other *supervised* learning techniques adopt the opposite approach. SVMs have been successfully applied to a wide range of

pattern recognition problems, including handwriting recognition, object recognition, speaker identification, face detection and text categorization [4]. SVMs are attractive because they boast an extremely well developed theory. A support vector machine finds an optimal separating hyperplane between members and non-members of a given class in an abstract space. SVMs, as applied to gene expression data, begin with a collection of known classifications of genes. These collections, such as genes coding for ribosomal proteins contain genes known to encode proteins that function together and hence exhibit similar expression profiles. One could build a classifier capable of discriminating between members and non-members of a given class, such that, given expression data for a particular gene, one would be able to answer such questions as, "Does this gene code for a ribosomal protein?" Such a classifier would be useful in recognizing new members of the class among genes of unknown function. Furthermore, the classifier could be applied to the original set of training data to identify outliers that may have been previously unrecognized. Whereas unsupervised methods determine how a set of genes clusters into functional groups, SVMs determine what expression characteristics of a given gene make it a part of a given functional group. Because the question asked by supervised methods is much more focused than the corresponding question asked by unsupervised methods, supervised methods can use complex models that exploit the specific characteristics of the given functional group. And our motive will be to determine exactly how well can a SVM achieve this afore mentioned goal, and with which computational models can it perform the best.

The present work is organized as follows. In Chapter 2 we talk about the definition and some significant features of the promoter, then we discuss a little about the Support Vector Machines. Chapter 3 is completely devoted to explanation of the concepts working behind SVM and the present state of the art. Our work intended and what we want to achieve is discussed in chapter 4. In chapter 5 we will go through the actual solution of the problem in hand, where we will present our methods in the promoter prediction. In the next chapter we can find the summarization of the results and the conclusion of our work. Following it is the list of references to all the great studies and material that helped us in achieving our intended goal. Relevant help topics are available

in appendices A & B regarding terminology used in the field of machine learning and the promoter data used.

Chapter 2
A Brief Review of SVM and Promoters

Promoter is a fragment of DNA sequence that is responsible for the transcription from DNA to RNA. Through the study on promoters, we can find out which DNA sequence will be transcribed into RNA. In this thesis we do not search for the features of promoters by observation. There may exist, some implicit features in promoter regions. The more features we know, easier it is to predict promoters. We are here trying to take advantage of computers to do some operations in sequences and help us predict promoters. The dataset for our promoter prediction in this thesis contains only one species, Escherichia coli (E. coli).

It is believed that promoters in the homologous gene are highly similar in DNA sequences, and sometimes they even have only a little position offset across different species.

2.1 Preliminaries

In this section, we shall first explain what the promoter is and biological function role of the promoter in brief, and then present some distinct features of promoter sequences that have been known and some methods for the promoter predictions.

2.1.1 Definition of the Promoter

A gene can be roughly divided into five parts: promoter, 5' UTR (untranslated region), exons, introns, 3'UTR, and polyadenylation site. Exons can be translated into proteins after they are transcribed into RNA. The *promoter* plays an important role in DNA transcription. The promoter is defined as the sequence in the region of the upstream of the *Transcriptional Start Site* (TSS) [2]. The related position of the promoter in a DNA sequence is illustrated in *Figure 2.1*.

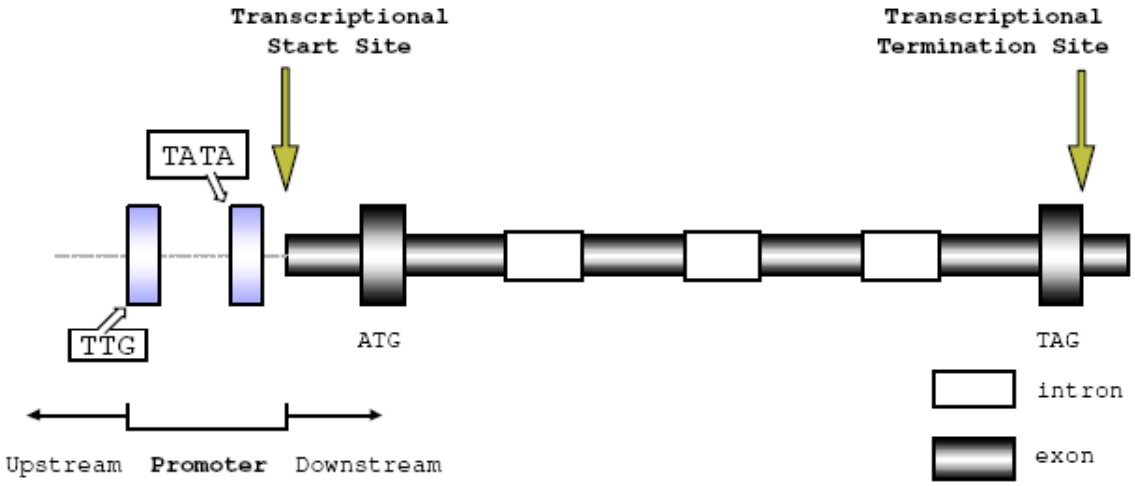


Figure 2.1 The promoter region in a DNA sequence

A promoter is required for a DNA sequence to be transcribed. In a DNA sequence transcription, there must be a promoter in the sequence. Binding with different kinds of RNA polymerases in the promoter regions, the DNA sequence will be transcribed into various RNA sequences, such as tRNA, rRNA, and mRNA. mRNA is important for protein production. After translation, an mRNA sequence will be translated into a protein (an amino acid sequence). Thus, if we know which DNA sequence can be transcribed and translated into an amino acid sequence, it will be useful for biologists to understand the gene regulation and expression [14]. When the promoter sequence is bound with the RNA Polymerase II enzyme, which we want to predict, the DNA sequence can be transcribed into mRNA sequence. The central dogma of molecular biology is shown in *Figure 2.2*.

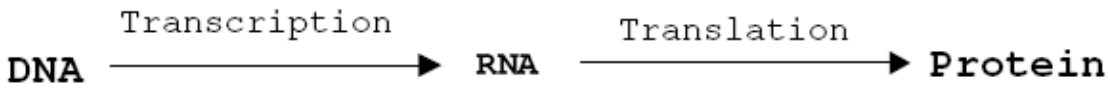


Figure 2.2 The central dogma of molecular biology

2.1.2 Significance of Promoter Prediction

Because the gene sequence data are growing fast recently, it is important to maintain and annotate such data. However, traditional biological experiments are not enough. How to design good computer algorithms and software to analyze and annotate gene sequences becomes one of the most important issues today.

If we know the position of the promoter, we will know the position of the first exon. With knowing the position of the first exon, we get the starting position of the coding region, which will be translated into the protein sequence. How to find the promoter of a DNA sequence is one of the essential points in the work of gene sequence analysis.

If we know which segment of a DNA sequence is the promoter sequence, we can use the promoter sequence to regulate the speed of translation from DNA into a protein. Furthermore, the promoter is also useful in genetically modified foods. With the similar method, we can also have the protein which causes disease grow more slowly, even destroy it.

Since the promoter is located around the upstream of TSS in a DNA sequence, and the RNA Polymerase II is always binding in that region. The transcription starts from the end of 5' of the DNA sequence, the 5' UTR (upstream of TSS) contains promoter sites (such as TATA-box), and the 3' UTR (downstream of TSS) contains stop codon. The translation stops when the stop codon is met.

However, sometimes even the upstream of TSS of a DNA sequence contains some transcriptional features, the promoter may not exist. Whether a DNA sequence transcribed or not can be verified by biological experiments, but experiments are usually time consuming and take high cost. By the promoter prediction method, we may be able to narrow down the promoter regions among massive DNA sequences. A further experiment then can be designed and tested. Therefore, much more time and cost will be saved.

2.1.3 Features of Promoter Sequences

Here we will show some significant features of promoter sequences which have been reported in some literatures. Some of these features are valid only in either prokaryotic or eukaryotic promoter sequences.

2.1.3.1 TATA Box and TTG Box

The *transcriptional elements*, which have high appearance frequency in the promoter region, are guessed to play an important role in transcriptions to find transcriptional elements from the promoter sequences is the basic concept of finding the features of promoters. Experimentally, two identified transcriptional elements in promoter sequences are the -10 box and -35 box. -10 and -35 means that these elements always appear around the positions of -10 and -35 (The position of TSS is +1). The -10 box is TATA-box and -35 box is the pattern of TTG.

2.1.3.2 CpG Islands

CpG islands is another feature of promoter sequences. CpG islands means that in a sequence, the G nucleotide usually appears following the C nucleotide. The 'p' in CpG denotes the phosphodiester linkage of the DNA sequence. In fact, a DNA sequence is always methylated around the TSS, so CpG islands have high appearance frequency in the promoter in all DNA sequences. This feature is found in eukaryotic promoter sequences. No significant CpG islands have been observed in prokaryote. So this feature can not help us in the promoter prediction with *E.coli*.

2.2 Facts at Hand and Background of SVM

Each vector in the gene expression matrix may be thought of as a point in an m-dimensional space. A simple way to build a binary classifier is to construct a hyperplane separating class members from non-members in this space. This is the approach taken by perceptrons, also known as singlelayer neural networks.

Unfortunately, most real-world problems involve non-separable data for which there does not exist a hyperplane that successfully separates the class members from non-class members in the training set. One of the solutions to the inseparability problem is to map the data into a higher-dimensional space and define a separating hyperplane there. This higher-dimensional space is called the *feature space*, as opposed to the *input space* occupied by the training examples. With an appropriately chosen feature space of sufficient dimensionality, any consistent training set can be made separable.

However, translating the training set into a higher-dimensional space incurs both computational and learning-theoretic costs. Representing the feature vectors corresponding to the training set can be extremely expensive in terms of memory and time. Furthermore, artificially separating the data in this way exposes the learning system to the risk of finding trivial solutions that overfit the data.

Support vector machines elegantly sidestep both difficulties [15]. SVMs avoid overfitting by choosing a specific hyperplane among the many that can separate the data in the feature space. SVMs find the *maximum margin hyperplane*, the hyperplane that maximixes the minimum distance from the hyperplane to the closest training point (see *Figure 2.2*). The maximum margin hyperplane can be represented as a linear combination of training points. Consequently, the decision function for classifying points with respect to the hyperplane only involves dot products between points. Furthermore, the algorithm that finds a separating hyperplane in the feature space can be stated entirely in terms of vectors in the input space and dot products in the feature space. Thus, a support vector machine can locate a separating hyperplane in the feature space and classify points in that space without ever representing the space explicitly, simply by defining a function, called a *kernel function* that plays the role of the dot product in the feature space. This technique avoids the computational burden of explicitly representing the feature vectors.

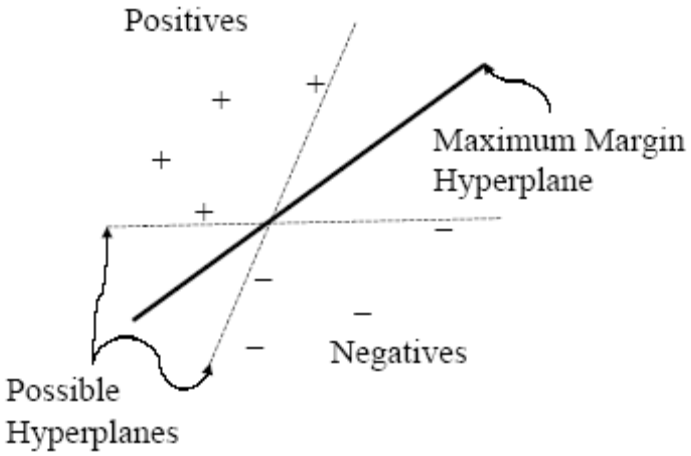


Figure 2.3 Maximum margin hyperplane. The figure shows four positive and four negative examples in a two-dimensional input space. Three separating hyperplanes are shown, including the maximum margin hyperplane**.**

The selection of an appropriate kernel function is important, since the kernel function defines the feature space in which the training set examples will be classified. As long as the kernel function is legitimate, an SVM will operate correctly even if the designer does not know exactly what features of the training data are being used in the kernel-induced feature space. Human experts often find it easier to specify a kernel function than to specify explicitly the training set features that should be used by the classifier. The kernel expresses prior knowledge about the phenomenon being modeled, encoded as a similarity measure between two vectors.

In addition to counteracting overfitting, SVM's use of the maximum margin hyperplane leads to a straightforward learning algorithm that always deterministically converges to the same solution for a given data set, regardless of the initial conditions, unlike the back propagation learning algorithm for artificial neural networks.

Another appealing feature of SVM classification is the sparseness of its representation of the decision boundary. The location of the separating hyperplane in the feature space is

specified via real-valued weights on the training set examples. Those training examples that lie far away from the hyperplane do not participate in its specification and therefore receive weights of zero. Only the training examples that lie close to the decision boundary between the two classes receive nonzero weights. These training examples are called the *support vectors*, since removing them would change the location of the separating hyperplane. The support vectors in a two-dimensional feature space are illustrated in *Figure 2.3*.

The SVM learning algorithm is defined so that, in a typical case, the number of support vectors is small compared to the total number of training examples. This property allows the SVM to classify new examples efficiently, since the majority of the training examples can be safely ignored. In essence, the SVM focuses upon the small subset of examples that are critical to differentiating between class members and non-class members, throwing out the remaining examples. This is a crucial property when analyzing large data sets containing many uninformative patterns, as is the case in many data mining problems. SVMs effectively remove the uninformative patterns from the data set by assigning them weights of zero.

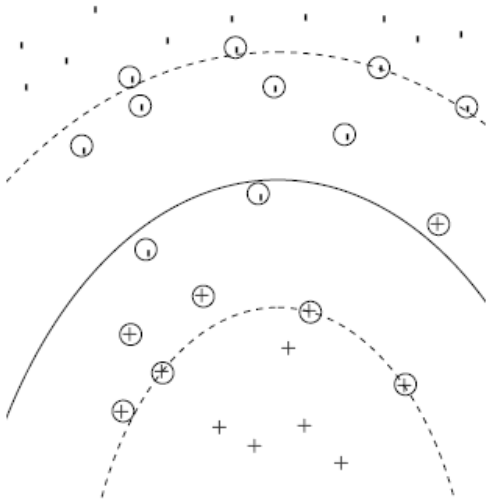


Figure 2.4 **A separating hyperplane in the feature space corresponding to non-linear boundary in the input space.**

The maximum margin allows the SVM to select among multiple candidate hyperplanes; however, for many datasets, the SVM may not be able to find any separating hyperplane at all, either because the kernel function is in appropriate for the training data or because the data consists of mislabeled examples. The latter problem can be addressed by using a *soft margin* that accepts some misclassifications of the training examples. A soft margin can be obtained in two different ways. The first is to add a constant function to the kernel function output whenever the given input vectors are identical [13]. The second is to define *a priori* an upper bound on the size of training set weights [5]. In either case, the magnitude of the constant factor to be added to the kernel or to bound the size of the weights controls the number of training points that the system misclassifies. The setting of this parameter depends on the specific data at hand. Completely specifying a support vector machine therefore requires specifying two parameters: the kernel function and the magnitude of the penalty for violating the soft margin.

To summarize, a support vector machine finds a non-linear decision function in the input space by mapping the data into a higher dimensional feature space and separating it there by means of a maximum margin hyperplane. The computational complexity of the classification operation does not depend on the dimensionality of the feature space, which can even be infinite. Overfitting is avoided by controlling the margin. The separating hyperplane is represented sparsely as a linear combination of points. The system automatically identifies a subset of informative points and uses them to represent the solution. Finally, the training algorithm solves a simple convex optimization problem. All these features make SVMs an attractive classification system.

Chapter 3
Concepts and State of The Art in SVM Classification

In many fields of science, computer science in particular, automatic learning from examples is a long-standing goal. Also, in recent years the amount of information that has to be processed has exploded and there is a growing need to extract structure from the data instead of just storing it. Moreover, if one is able to capture some dependence in the data, this knowledge can be used to predict future situations [11]. To model the structure in the data it is usually important to know the causality of the data generating process. Usually a factor causing a change in some other factor is called a variable and the factor which changes due to the change of the variable is called a response variable. In many cases there are several variables affecting the response variable and in this work we use symbol $\vec{x}$ to denote a vector that consists of variables $\vec{x}_1, \ldots, \vec{x}_N$ and the symbol y is used to denote a response variable.

There are two main types of learning from examples. In regression the response variable *y* can have infinitely many values; while in classification it can have only a finite number of values. Another interpretation is that, in the regression case one is learning a function and in the classification case one is learning sets and the number of sets is infinite. In this work we deal with classification, which is usually simpler type of learning. In particular, we examine a classification algorithm called the support vector machine, which has many good practical and theoretical properties. One of the best practical properties is its simplicity, which can be very important when dealing with large datasets. History of the support vector machines is ambiguous: it is difficult to name one single paper that introduced the concept. The fundamental theory of linear classifiers, which also includes support vector machines, dates back to the 1930's while Rosenblatt in 1956 introduced the perceptron of which the support vector machines are one very special case. Linear large margin classifiers, which are the simplest form of the support vector machines have also been invented by several people in various fields of research and similar ideas are

presented in many articles. Generalization of linear large margin classifiers to the nonlinear case was introduced in [3], which is the first paper that presents the current support vector machine methodology, but due to earlier work it cannot be said that it is the essential paper that constitutes the principles of the support vector machines. Theoretical properties of the support vector machines have been studied in conjunction with general research on the theoretical properties of machine learning. Theory on which the support vector machines are based has changed over time. The first explanation was given by the Vapnik-Chervonenkis theory, followed by the theory of large margin classifiers, but these explanations have still left something to hope for.

Let us review what is known till date about Support Vector Machines. We will go through the concepts working behind SVM in classification with a set of reasonable examples.

3.1 What Exactly is a Support Vector Machine?

Support vector machines are a family of learning algorithms. The SVM we will study here learns how to classify objects into two classes, based on a series of observations (this task is also called *pattern recognition)* [8].

Let us call $\mathcal{X}$ the set of objects one might want to classify. A particular object $\vec{x}$ is an element of this set, which we denote by $\vec{x} \in \mathcal{X}$. The class or category of an object can take two values, which can be for instance –l or +1. We will use the notation y to represent such a class, hence $y \in \{-1, +1\}$.

An observation is simply an object $\vec{x}$ together with its class y, which we denote by $(\vec{x}, y)$. With these notations a series of *N* observations can be written as follows:

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}, \qquad \textit{(Eqn 3.1)}$$

where for any i = 1,…,N, $(\vec{x}_i, y_i)$ is an *observation*. The goal of a learning algorithm is to use a set of observations *S,* which we call the *training set,* to learn a

rule which can be used in the future to classify any new object $\vec{x} \in \mathcal{X}$ into a class $y \in \{-1, +1\}$.

SVM is one particular algorithm that performs this pattern recognition task, i.e. learning from *S* a *classifier* for future objects.

Example 1 *Consider the problem of guessing whether a protein* is *an enzyme or not based on its amino-acid sequence (primary structure). In that case the set X* is *the set of all possible finite-length amino-acid sequences, and the class y would be* $+1$ *if the protein* is *an enzyme and* -1 *if it* is *not. A training set* $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ *would be a database of N protein sequences* $\vec{x}_1, \ldots, \vec{x}_N$ *together with their classes* $y_1, \ldots, y_N$. *A SVM would then learn from this database a rule which could be used to classify any new protein as enzyme or non-enzyme from its primary structure.*

3.2 What Makes it so Special?

The problem of learning a classification rule from a set of examples is very classical, and is the main focus of the discipline called *machine learning.* Many learning algorithms existed before the invention of SVM: famous methods include classification and regression trees (CART) or neural networks. However, since its introduction in the mid-nineties the SVM method has quickly become one of the most popular learning algorithms in the machine learning community, and is currently applied to more and more real-life classification problems (including biological issues).

The main reason for its popularity is that it is very efficient compared to other methods in many real world applications [1]. Benchmark experiments have been conducted to compare different learning methods in various applications, and SVM are usually among the best algorithms, or even outperform other methods (for instance in text classification, written character recognition or protein family identification from the primary structure).

Being efficient means several things, which we can quickly sum up:

Good generalization performance: once the SVM is presented with a training set, it is able to learn a rule, which can *correctly classify any new object* quite often.

Computational efficiency*:* the algorithm is efficient in terms of speed and complexity.

Robust in high dimensions*:* dealing with large dimensional objects (like images of gene expression data) is usually difficult for learning algorithms, because of the overfitting issue (see next section). SVM seem to be more robust than other methods in such cases.

Finally, also SVM became popular in the machine learning and mathematics community because it is based on a beautiful theory, which is the subject of much theoretical research.

3.3 How Come it's so Efficient?

This is a difficult question, and today nobody knows for sure why it works so well. However theoretical results suggest that its efficiency is mainly due to its capacity to find rules that classify objects with high confidence, to prevent them form *overfitting.*

Overfitting is a central issue for learning algorithm. When a training set is presented to a learning algorithm, the algorithm usually tries to find a rule which explains well the observations, i.e., which correctly classifies most of the objects in the training set. Sometimes the algorithm can find a very complicated rule which perfectly classifies the objects in the training set, but this rule could be useless to classify new observations because it is too related to the training set: we say that such a rule does not *generalize* well, and this phenomenon is called overfitting.

Example 2 *Suppose the goal* is *to learn whether a protein* is *an enzyme or not based on the primary sequence, and the training set* is *very small, e.g. the following:*

$$S = \{(MKSRAAVA \ldots, +1), (MNVMGFAA \ldots, +1), (MKTRDSQS \ldots, -1), (MKNEKRKT \ldots, -1)\}$$

Hence this training set contains two enzymes and two non-enzyme proteins.
From this training set one can imagine many rules, which correctly classify all four sequences, e.g.:

If the sequence starts with MKSR or MNVM then the protein is *an enzyme, otherwise it* is *not. Obviously this rule* is *not very good, because it won't be able to correctly classify many enzymes which are not in the training set, even though it correctly classifies every sequence in the training set: hence this rule will not generalize well, and over fitting occurs.*

Example 3 *Neural networks are known to be very powerful to correctly classify almost any training set. However they often overfit and many people consider them as "black box" which can learn almost anything but don't always generalize well. To prevent a neural network from over fitting, several methods are used to prevent it from "learning too well".*

Most learning algorithms try to find a rule which correctly classify the objects in the training set: this is the most natural way to learn. However it is usually not clear whether a learned rule will generalize well, or whether it will overfit. To cope with this overfitting issue, Vapnik and Chervonenkis studied the link between the ability of a learning algorithm to learn a good rule for the training set, and its ability to generalize well, i.e. not to overfit. Their work resulted in a theory called *statistical learning theory,* and their theoretical results then gave birth to SVM. As a result SVM were designed from the beginning with a single goal: *generalize well,* to

the contrary of other learning algorithms which are designed with a different goal: *correctly classify the training set.*

3.4 SVM and Vapnik's Statistical Learning Theory

In this section we explain Vapnik's Statistical Learning Theory in a little detail and try to establish the link between the theory and Support Vector Machines. We also go through some basic definitions and try o understand the relevance of the theory to machine learning concept. The topics covered here are:

- Vapnik's Statistical Learning Theory
- How does Statistical Learning Theory link to SVM

3.4.1 Vapnik's Statistical Learning Theory

Statistical learning theory makes a link between two important features of a learning algorithm:

- Its ability to learn a rule that correctly classifies most examples in the training set.
- The ability of the resulting rule to correctly classify new objects (i.e., the ability to generalize well)

Let us use the symbol f to denote a classification rule (or *classifier),* i.e. a mapping from the space of objects $\mathcal{X}$ to the space of classes {-1, + 1}. In other words, the classifier f classifies any object $\vec{x} \in \mathcal{X}$ into the class $f(\vec{x})$, which is either -1 or +1.

With these notations we can reformulate the learning task as follows: a learning algorithm uses a training set *S* to learn a particular classifier $f \in \mathcal{F}$. Here *F* denotes the set of all possible classifiers the algorithm can choose among. For example, in the case of decision trees, the set *F* can be seen as a set of trees with questions attached to each node.

In order to choose a classifier f from the set *F* based on the training set *S,* it is natural to judge the quality of every $f \in \mathcal{F}$ by their ability to correctly classify the objects in the training set. Hence we define the *empirical risk* of the classifier f as the percentage of good classification it makes on the training set. Let us denote by $R_{emp}(f)$ the empirical risk, which is therefore a number between 0 and 1: for a classifier f which correctly classifies all examples in the training set, we have $R_{emp}(f) = 0$, while for a "bad" classifier f which makes a mistake for each example, we have $R_{emp}(f) = 1$.

Hence $R_{emp}(f)$ characterizes the capacity of a rule f to correctly classify the examples in the training set. Classical learning algorithms like neural networks usually search for classification rules $f \in \mathcal{F}$ with $R_{emp}(f)$*)* as small as possible.

What about the generalization performance of a rule f*?* A convenient way to define it is to suppose that the training examples (as well as the future examples to be classified) are generated one by one (and independently) by a random machine according to a fixed probability distribution. Under this hypothesis an object to be classified is a random object $\vec{x}$, its class is random variable which can take only two values (for instance +1 and -1), and an observation *(X, Y)* is governed by a probability *P*. In this probabilistic framework, the probability *P* is of course unknown a priori. The only thing we assume is that the training set is made of *N* random variables $(X_1, Y_1), \ldots, (X_N, Y_N)$ which are generated according to *P*.

It is now possible to define precisely what "*generalization performance*" of a rule f means: it is the probability that the rule f makes a mistake on a new sample randomly generated according to the distribution *P,* which we can write as:

$$R(f) = P\left(f(X) \neq Y\right). \qquad \textit{(Eqn 3.2)}$$

R(f) is called the *risk* of the classifier f, and quantifies the ability of a rule to generalize well: if *R(f)* $= 0$ then f will never make any error of any new observation, so it generalizes perfectly.

For a given rule f and a given training set *S,* the empirical risk $R_{emp}(f)$ can be observed but the risk *R(*f*)* is not observed. Intuitively however, it is natural to think that in many cases, if $R_{emp}(f)$*)* is small (i.e., if the rule f makes few errors on the training set), then *R(f)* is small too. This is why many learning algorithms try to find a rule f with a small $R_{emp}(f)$*),* in the hope that *R(f)* will be small too.

This intuition is partially true: for a given rule f*,* by the law of large numbers, the empirical risk $R_{emp}(f)$*)* converges to the risk *R(f)* when the size of the training set increases, so $R_{emp}(f)$*)* is a good indicator of *R(f).*

The situation is a bit more complex when one analyses a learning algorithm.
Indeed a learning algorithm has to choose a rule f from a set *F* based on the observation of the training set. As we said before, a natural choice for learning algorithm is to choose the rule with smallest empirical risk on the training data, i.e. to choose the rule $\hat{f}$ defined by:

$$R_{emp}(\hat{f}) = \inf_{f \in \mathcal{F}} R_{emp}(f).$$

(*Eqn 3.3)*

This method is usually called *empirical risk minimization (ERM)*. On the other hand the real goal of the learning algorithm is to find the rule which generalizes best, i.e. to find the rule f* which satisfies:

$$R(f^*) = \inf_{f \in \mathcal{F}} R(f).$$

(*Eqn 3.4)*

The central question now becomes: is it true that the risk $R(\hat{f})$ of the rule selected by empirical risk minimization is not far from $R(f^*)$? The answer to this question appears to be negative when overfitting occurs: in that case, one can find rules very good at explaining the data (i.e., $R_{emp}(\hat{f})$ is very small), but with a poor generalization capacity (i.e., $R(\hat{f})$*)* is not small), and better rules could be found.

The main results in statistical learning theory relate $R(\hat{f})$ to $R(f^*)$. A typical result is the following:

$$\mathbf{E}R(\hat{f}) \leq R(f^*) + c\sqrt{\frac{V(\mathcal{F})}{N}},$$

(*Eqn 3.5)*

where c is a universal constant, *V(F)* is a quantity characteristic of the set of rules *F* called the *Vapnik-Chervonenkis dimension (or simply VC dimension)* of the class, and *N* is the number of examples in the training set. One should remember that the selected rule $\hat{f}$ depends on the training set, so it is in fact a random rule, and its risk $R(\hat{f})$ is therefore a random variable: this is why we use the symbol E in *Eqn3.5* to denote its average with respect to the random choice of the training sample.

Eqn3.5 shows that on average, the risk of the selected rule $\hat{f}$ is not far from the best possible risk $R(f^*)$ if two conditions hold:

- The number of observations *N* in the training set should be large enough.
- The VC dimension of the set of possible rules *F* should be small enough.

Hence the VC dimension plays a central role: the smaller the VC dimension, the better the generalization performance of the rule selected by empirical risk minimization. In other words, a learning algorithm will find a rule which generalizes well if it can chose the rules from a set *F* with small VC dimension.
It would be too long to study in detail what the VC dimension of a set is: let us just say that the "larger" a class *F,* the larger its VC dimension. For example the set of rules a neural network can chose is very large, because it can almost always find a rule which perfectly explains any training set: not surprisingly its VC dimension tends to infinity when the number of neurons increases, which means that *Eqn3.5* makes no sense for a neural network.

In conclusion, the main contribution of statistical learning theory to the design of learning algorithm is to point out the importance of controlling the "size" of the set of rules *F* the algorithm can chose among, as measured by its VC dimension.

3.4.2 How does Statistical Learning Theory Link to SVM?

We saw in the previous section that the main results of the so-called statistical learning theory relate the risk of rule $\hat{f}$ selected by a learning algorithm from a set of rules *F* by empirical risk minimization to the risk of the best rule in the set *F* by *Eqn3.5*. Using this result, the fundamental idea behind SVM is the following: *in order to obtain good generalization performance, the VC dimension must be controlled.*

Controlling the VC dimension ensures good generalization performance. That means that the set *F* has to be "small". However, if it is too "small", then the rules is contains might be too simple to correctly classify the objects in the training set, simply because the classification task is not that easy. In other words, if *F* is small then there is a risk that $R(f^*)$ might be large. In that case, even if $R(\hat{f})$ is a good approximation of $R(f^*)$ (because the VC dimension is small), the resulting rule will not be good because F itself is so small.

Hence there are two opposite goals when designing a learning algorithm:

- Chose *F* as large as possible to ensure that at least one element in *F* has a small risk;
- Chose *F* as small as possible to ensure that the risk of the selected rule is almost as small as the risk of the best rule.

In order to solve this apparent paradox Vapnik proposed to consider a family of rules *F* which is the increasing union of families with increasing VC dimension,

i.e.:

$$\mathcal{F}_0 \subset \mathcal{F}_1 \subset \ldots \subset \mathcal{F}_n \subset \ldots \subset \mathcal{F},$$

with:

$$V(\mathcal{F}_1) \leq V(\mathcal{F}_2) \leq \ldots \leq V(\mathcal{F}_2) \leq \ldots \leq V(\mathcal{F}).$$

In each family *k* let $\hat{f}_k \in \mathcal{F}_k$ denote the rule with the smallest empirical risk in the family of rules *Fk,* and let $f_k^* \in \mathcal{F}_k$ denote the rule in *Fk* with smallest risk. When *k* increases, $R_{emp}(\hat{f}_k)$decreases because $\hat{f}_k$ minimizes the empirical risk of a set of increasing size$(\mathcal{F}_k \subset \mathcal{F}_{k+1})$. On the other hand, the term:

$$c\sqrt{\frac{V(\mathcal{F}_k)}{l}}$$

in *Eqn3.5*, which we call the confidence interval, increases with *k,* because

$$V(\mathcal{F}_k) \leq V(\mathcal{F}_{k+1})$$

As a result, the bound on the risk $R(f_k^*)$ given by *Eqn3.5* typically first decreases with *k,* and then increases as shown in *Figure 3.1*. As a result there might exist an optimal *k** which ensures the lowest upper bound for the generalization error of $\hat{f}_{k^*}$*:* choosing this $\hat{f}_{k^*}$ is called *structural risk minimization.* A SVM is an implementation of the structural risk minimization principle.

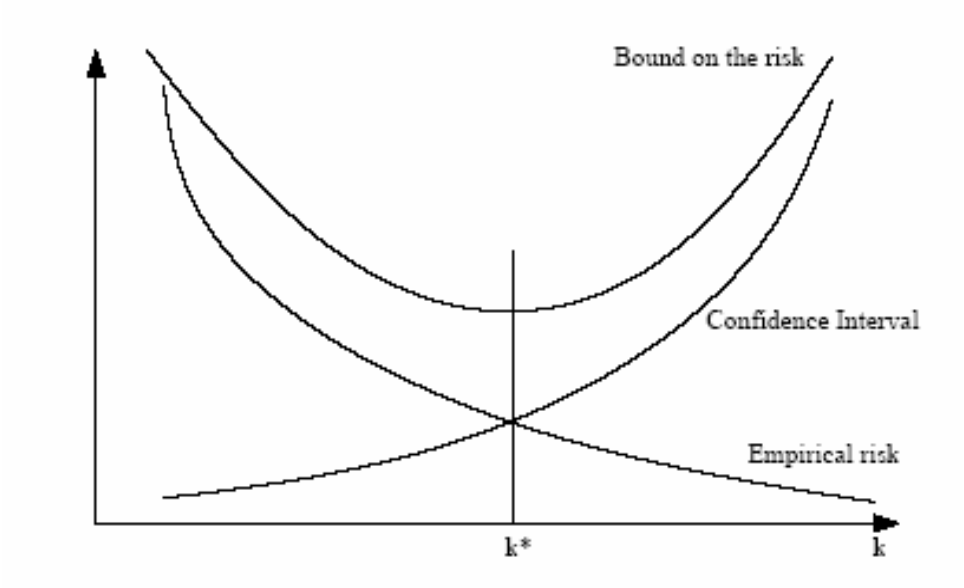


Figure 3.1 Structural risk minimization

3.5 About SVM Algorithm

We saw in the preceding part of this chapter that a support vector machine is a learning algorithm whose purpose is to classify objects into classes. Let us now study in more detail the algorithm itself.

The objects to be classified can be almost anything, like pictures, sound, molecules, graphs etc ... However, we will only consider a particular kind of objects, namely finite dimensional real vectors. A m-dimensional real vector $\vec{x}$ has m coordinates which we can write $\vec{x} = (x_1, \ldots, x_m)$, where each x_i is a real number $(\text{i.e., } x_i \in \mathbb{R} \text{ for } i = 1, \ldots, m)$. In that case the set of possible objects is denoted by $\mathcal{X} = \mathbb{R}^m$. Observe that such objects are often derived from more complex objects (like images, molecules...) by computing the values of different features to characterize the object [9].

Example 4 *Suppose one wants to represent proteins for classification. One possibility* is *to create a 20-dimensional real-valued vector for each protein, where each coordinate represents the percentage of a particular amino acid in the protein composition.*

The advantage of studying such objects is that they represent points in a Euclidean space.

It is possible to design simple linear classifiers on such spaces.

3.5.1 Linear Classifiers

Suppose m $=$ 2, i.e. each object has only two coordinates: $\vec{x} = (x_1, x_2)$. Such an object can be represented by a point in a plane, see *Figure 3.2*.

In that space a line can be characterized by a linear equation of the form:

$$\vec{w}.\vec{x} + b = 0,$$

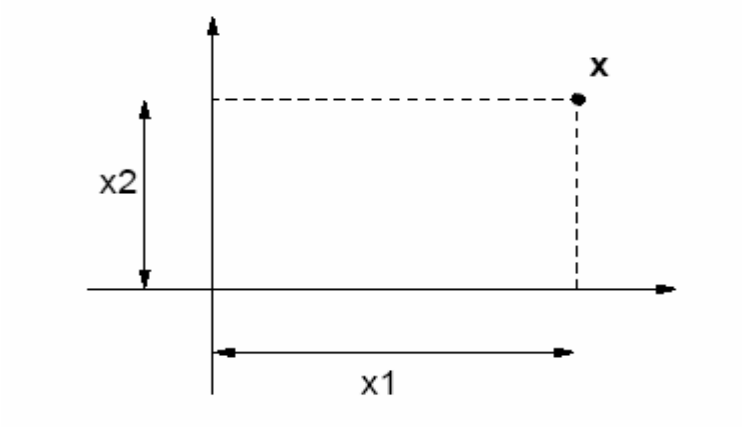


Figure 3.2 Vector Representation in $\mathbb{R}^2$

where $\vec{w}$ and b are two vectors and $\vec{w}.\vec{x}$ is the dot product between $\vec{w}$ and $\vec{x}$, i.e.:

$$\vec{w}.\vec{x} = w_1 \times x_1 + w_2 \times x_2.$$

In other words the set of points $\vec{x}$ that satisfy *Eqn3.5* form a straight line in the plane, see *Figure 3.2*.

It is well known that two vectors $\vec{x}_1$ and $\vec{x}_2$ are orthogonal (or perpendicular) if and only if their dot product is equal to zero, i.e. $\vec{x}_1.\vec{x}_2 = 0$. This property can be used to check that the vector $\vec{w}$ is orthogonal to the line defined by *Eqn3.5*. Indeed, if $\vec{x}_1$ and $\vec{x}_2$ are two points on the line, they satisfy $\vec{w}.\vec{x}_1 + b = \vec{w}.\vec{x}_2 + b = 0$. By subtracting the two inequalities one get $\vec{w}.(\vec{x}_1 - \vec{x}_2) = 0$, which shows that the vector $\vec{w}$ is orthogonal to the vector $\vec{x}_1 - \vec{x}_2$, which is precisely parallel to the line (see *Figure 3.2*).

Such a line divides the whole space into two areas, or *half-spaces*. One is defined by the set of X such that:

$$\vec{w}.\vec{x} + b > 0,$$

and the other is defined by the set of X such that:

$$\vec{w}.\vec{x} + b < 0.$$

The line itself is the frontier between the two areas (see *Figure 3.3*).

Such a line defines a classifier, which we call a *linear classifier* as follows. It classifies any point $\vec{x} \in \mathbb{R}^2$ depending on its position with respect to the line. It is classified as + 1 if $\vec{w}.\vec{x}+b>0$, or as -1 if $\vec{w}.\vec{x}+b<0$. If $\vec{x}$ falls on the line ($\vec{w}.\vec{x}+b=0$) then there is indeterminacy.

This construction can be generalized to any dimension greater than 2 by keeping the same notations. In $\mathbb{R}^m$ the dot product is defined by:

$$\vec{w}.\vec{x} = w_1 \times x_1 + w_2 \times x_2 + \ldots + w_m \times x_m.$$

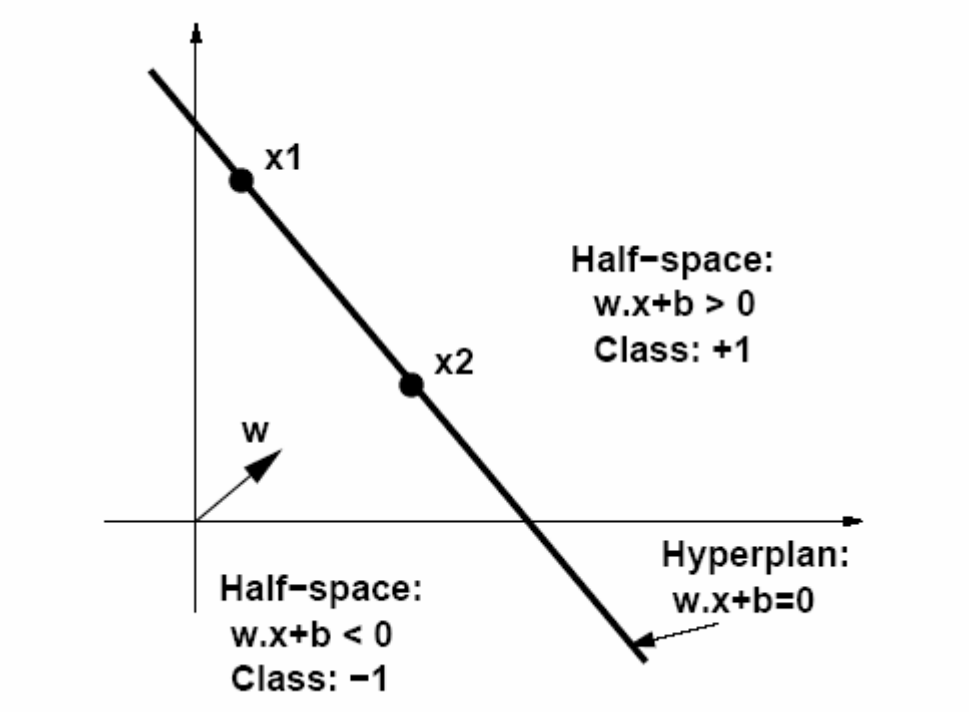


Figure 3.3 Linear Separator in $\mathbb{R}^2$

For any vectors $\vec{w}$ and b in $\mathbb{R}^m$, the set of points $\vec{x}$ which satisfies :

$$\vec{w}.\vec{x}+b=0$$

is called an hyperplane (the equivalent to a line when m = 2) which divides the whole space into two half-spaces. It therefore defines a linear classifier which classifies any vector $\vec{x}$ into one class or the other depending on the sign of $\vec{w}.\vec{x}+b$.

3.5.2 Linearly Separable Training Set

Using the notations introduced above let us consider a training set *S* which is a set of points $\vec{x}_i \in \mathbb{R}^m$ together with their classes $y_i \in \{-1, +1\}$ for i = 1 ... , N, i.e.:

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}.$$

Our goal in this chapter is to learn a classifier from such a training set. Obviously linear classifiers are very simple, and they might not be very efficient to classify a given training set, e.g. if positive and negative examples are spread everywhere in the plane. Therefore, for introductory purposes, we will only consider a very limited class of training set here, namely the training set which can be perfectly classified by at least one linear classifier. This restriction will enable us to present the main features of SVM, and will be discarded further.

We say that the training set *S* is *linearly separable* if there exists at least one linear classifier defined by two vectors w and *b* which correctly classifies all objects in *S,* i.e.:

$$\begin{cases} \vec{w}.\vec{x}_i + b > 0 & \text{if } y_i = +1 \\ \vec{w}.\vec{x}_i + b < 0 & \text{if } y_i = -1 \end{cases}$$

for all i = 1, ... , *N*. This situation is represented in *Figure 3.4*.

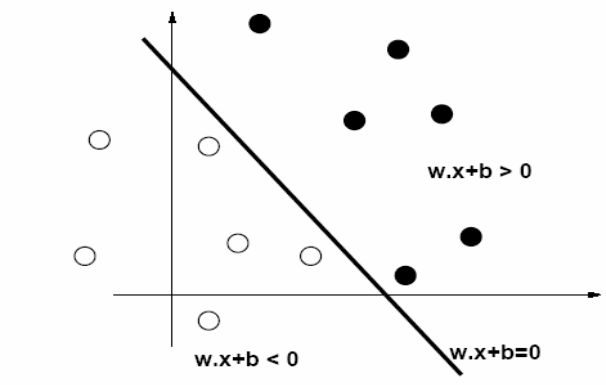


Figure 3.4 Separable training set: black circles are positive examples, white circles are negative examples

3.5.3 Linear SVM for Separable Training Set

If the training set is linearly separable (see *Figure 3.4*) then there are usually many linear classifiers, which correctly classify it (see *Figure 3.5*).
In order to apply the results of the statistical learning theory, remember that in order to choose a "good" classifier one should take care of two factors:

- Choose a classifier with an empirical risk as small as possible.
- Choose a classifier from a family with VC dimension as small as possible.

In case of linear classifiers for a separable training set the first condition is fulfilled by choosing any classifier, which correctly classifies all objects in the training set, i.e. by choosing any of the classifiers depicted in *Figure 3.5*.

In order to fulfill the second condition one needs to know what VC dimension means for families of linear classifiers. A classical result in learning theory states that the VC dimension is related to the smallest distance between a point in the training set and the separating hyperplane. This distance is also called *margin* and denoted by γ. *Figure 3.6* shows the margin for a particular classifier.

More precisely the VC dimension of the set of linear classifiers, which correctly classify the data with a margin at least equal to γ decreases with γ. As a result an application of the structural risk minimization principle leads to the following rule:
Chose a linear classifier which correctly classifies all training examples (to have the smallest possible empirical risk)

Among those classifiers, chose the one with the largest margin (to have the smallest possible VC dimension).

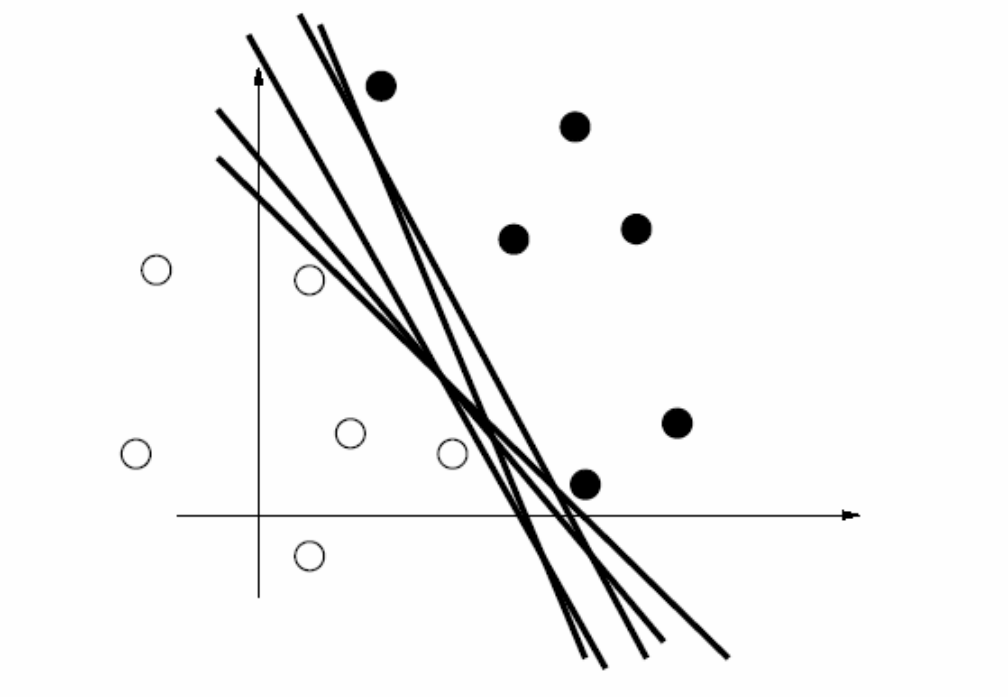


Figure 3.5 Linear classifier that correctly classifies the training set

The hyperplane with the largest margin for a given training set is called the *optimal hyperplane.* The learning algorithm, which chooses the liner classifier with largest margin is called a *linear support vector machine for separable training set.*

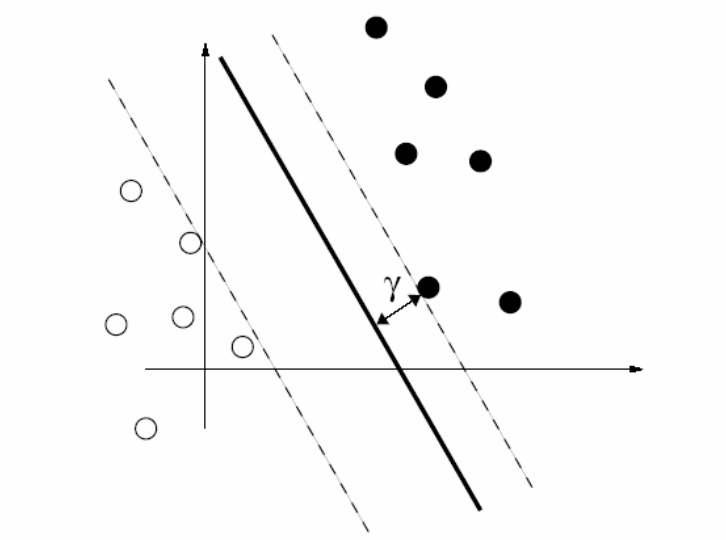


Figure 3.6 The margin γ of a linear classifier

3.6 Kernels and Non-Linear Support Vector Machines

In previous section we studied linear SVM. The resulting classifier is very simple, because it simply classifies a point as positive or negative depending on whether the point is on one side or on the other side of an hyperplane, e.g., a line in 2 dimensions.

Such classifiers are clearly too simple to reflect the complexity of some tasks. As an example, consider the training set represented in *Figure 3.7*. Any linear classifier will do a bad job on this training set, which however looks very simple to separate if one could use circles instead of lines. In this section we show how SVM can be very easily generalized to handle such cases.

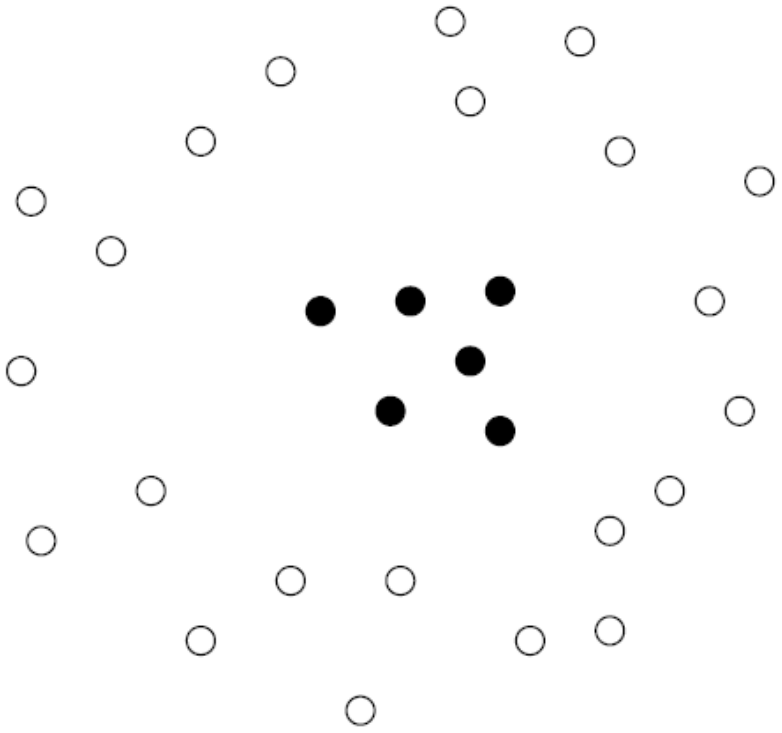


Figure 3.7 Non-separable training set

3.6.1 Feature Space

The training set $\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ is a set of labeled examples. Suppose that one is able to define a set of real functions $\phi_1, \ldots, \phi_M$ on the space of objects. These functions are called *features.* Then any object *X* can be mapped to a real vector *¢(x)* with dimension *M* as follows:

$$\vec{x} = (x_1, \ldots, x_m) \rightarrow \phi(\vec{x}) = (\phi_1(\vec{x}), \ldots, \phi_M).$$

The features ¢i can be any function. In particular they don't need to be linear. Moreover the number of features *M* can be larger than the dimension m of the objects X.

After mapping all the points from the training set to the feature space, one gets a set of points:

$$\phi(\mathcal{S}) = \{(\phi(\vec{x}_1), y_1), \ldots, (\phi(\vec{x}_N), y_N)\},$$

in the feature space $\mathbb{R}^M$. The interesting fact about the features space is that *the training set ¢(S) can be linearly separable in the feature set even if the training set* is *not linearly separable in the original space.* An example is illustrated in *Figure 3.8*.

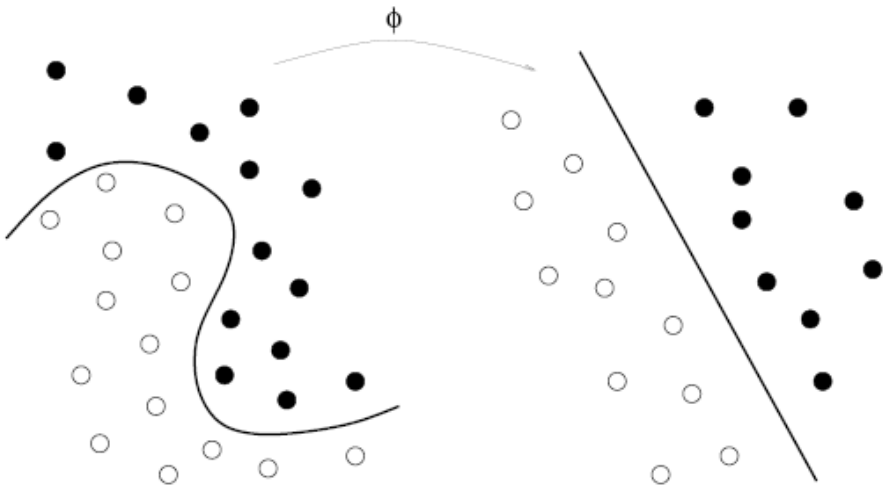


Figure 3.8 Linearly separable set in feature space

Example 4 *Suppose the initial data are two-dimensional points, i. e., m = 2 and* $\vec{x} = (x_1, x_2)$*. Consider the following mapping:*

$$\vec{x} = (x_1, x_2) \rightarrow \phi(\vec{x}) = (x_1^2, x_1 x_2, x_2^2)$$

Then a general linear hyperplane in the feature space is defined by a vector $\vec{w} = (w_1, w_2, w_3)$ *and a number b through the equation:*

$$\vec{w}.\phi(\vec{x}) + b = 0$$

This equation can be explicated as:

$$w_1 x_1^2 + w_2 x_1 x_2 + w_3 x_2^2 + b = 0$$

Even though this is a linear equation in the feature space, this corresponds to a polynomial equation in the input space $\mathbb{R}^2$*. As a result the set of linear classifier in the feature space is in fact the set of polynomial classifier in the input space. For example, the disk centered in O with radius R is defined by the equation:*

$$x_1^2 + x_2^2 < R^2$$

which corresponds to the vector $\vec{w} = (1, 0, 1)$ *and* $b = -R^2$ *in the feature space. A linear classifier once mapped to this feature space can easily separate. Hence the example shown in Figure 3.7.*

As a result the SVM approach can be generalized to non-linear classification by the following steps:

- Define a mapping ϕ to a feature space;
- Build a linear SVM in the feature space.

Definition: *A* kernel *K(.,.)* is *a function such that for any points (x, x') in the input space:*

$$K(\vec{x}, \vec{x}') = \phi(\vec{x}).\phi(\vec{x}')$$

where ϕ is *a mapping to a feature space.*

Example 5 *Consider the mapping:*

$$\vec{x} = (x_1, x_2) \rightarrow \phi(\vec{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

Then the corresponding kernel is:

$$\begin{aligned} K(\vec{x}, \vec{x}') &= \phi(\vec{x}).\phi(\vec{x}') \\ &= (x_1^2, \sqrt{2}x_1x_2, x_2^2).\left((x_1')^2, \sqrt{2}x_1'x_2', (x_2')^2\right) \\ &= x_1^2(x_1')^2 + 2x_1x_2x_1'x_2' + x_2^2(x_2')^2 \\ &= (x_1x_1' + x_2x_2')^2 \\ &= (\vec{x}.\vec{x}')^2 \end{aligned}$$

3.6.2 Implicit Mapping to a Feature Space

A kernel *K(.,.)* always corresponds to a dot product in a particular feature space defined by a mapping ¢(.). Hence the most natural way to build a kernel is first to define a mapping ¢(.), and then to compute the corresponding kernel (as we did in Example 5). However the computation of the kernel $K(\vec{x}, \vec{x}')$ can be sometimes much easier than the explicit computation of the mapping ¢(.). In Example 5, the kernel is simply:

$$K(\vec{x}, \vec{x}') = (\vec{x}.\vec{x}')^2$$

while mapping is:

$$\phi(\vec{x}, \vec{x}') = (x_1^2, x_1x_2, x_2^2)$$

In other words, it is possible to directly compute the kernel $K(\vec{x}, \vec{x}')$ between two objects without computing their images $\phi(\vec{x})$ and $\phi(\vec{x}')$.

This property is the main reason why it is very useful to use kernels: they are usually simple to calculate, but can correspond to complex feature spaces. Therefore we talk about *implicit mapping* to a feature space, because the data are mapped to a feature space where their dot product can be computed, while their images are not explicitly computed.

3.7 Popular Kernels

Let us now discuss two very popular kernels which are used in most SVM packages for classification purpose.

3.7.1 Polynomial Kernels

Two general polynomials kernels are defined as:

$$K_{Poly1}(\vec{x}, \vec{x}') = (\vec{x}.\vec{x}')^d$$

and

$$K_{Poly2}(\vec{x}, \vec{x}') = (\vec{x}.\vec{x}' + c)^d$$

where *d* is the degree of the polynomial and c is a constant in the second kernel.

We already met the polynomial kernel *K Poly1* of degree 2 in Example 5, it corresponds to a feature space spanned by all products of 2 variables, i.e., $\{x_1^2, x_1 x_2, x_2^2\}$. It is easy to see that the kernel *Kpoly2* of degree 2 corresponds to a feature space spanned by all products of at most 2 variables, i.e., $\{1, x_1, x_2, x_1^2, x_1 x_2, x_2^2\}$.

More generally the kernel *Kpoly1* corresponds to a feature space spanned by all products of exactly *d* variables, while the kernel *Kpoly2* corresponds to a feature space spanned by all products of at most *d* variables.

3.7.2 Radial Basis Function Kernel

This kernel is defined by

$$K(\vec{x}, \vec{x}') = \exp\left(-\frac{||\vec{x} - \vec{x}'||}{2\sigma^2}\right)$$

where σ is a parameter. It corresponds to a feature space with infinite dimension which cannot be completely explicated; hence this is a typical example where the explicit mapping to the feature space can not be computed while the dot product in that space is easy to compute.

Almost any shape can be obtained with this kernel. Observe that the smaller the parameter σ the more complex the decision boundary can be. The larger the σ the smoother the decision boundary.

3.8 Using the Support Vector Machine

The general approach to using a Support Vector Machine is as follows:

- Define your problem as a classification problem.
- Prepare a training set

$$\mathcal{S} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$$

- Choose a kernel K(.,.)
- Use the numerical methods to solve the problem. i.e., device a classifier
- Classify new objects with the help of classifier

There is usually no automatic way to choose a kernel and to adjust the corresponding parameters. Therefore one usually has to try different kernels and different parameters and to test their efficiency on his data. A good method is to randomly split the data into a training set and a test set, to train the SVM on the training set and to test it on the test set.

Chapter 4
Discussion of the Problem at Hand

4.1 Problem Background

In 1866, Gregor Mendel discovered genetics. Mendel's experiments on peas unveiled some biological elements called genes, which pass information from generation to generation.

Later in 1869 DNA was discovered. But it was not until 1944 that McCarty demonstrated DNA is the carrier of all genetic information. This remarkable discovery is referred to as the start of bioinformatics. In 1963, another historic discovery enabled great advances in both biology and bioinformatics, when James Watson and Francis Crick deduced the three dimensional structure of DNA, which is represented as a double helix.

Later in 1960, the genetic code, namely how the mapping of DNA to protein is done was elucidated. And since then there have been extensive studies on techniques to predict the promoters. Promoters are functional regions responsible for the initiation and regulation of DNA transcription. Accurate prediction of promoters is fundamental to understanding gene expression patterns. Many biological techniques exist for the same, but for the sake of quicker and low-resource-consuming techniques we are betting on the computer power. The idea is to learn a classifier for the promoter by examination of a collection of positive and negative examples. The basic idea is to first recognize potentially distinguishing attributes or patterns and then learn which combinations of these attributes discriminate positive from negative examples. The idea is quite natural and there have been several attempts along these lines mainly focusing on the classification task. Several researchers have studied the problem of recognizing promoters and various approaches have been reported for it, such as Neural Networks, Hidden Markov Models, Graph-based Induction Method, etc.

4.2 What is Intended?

Developing promoter recognition algorithms is a challenging problem since the understanding of transcriptional processes is incomplete. A major deficiency of such available programs is the very high number of False Positive (FP) predictions for any significant level of True Positive (TP) predictions. So the aim is to keep a balance between the two.

The framework for promoter recognition consists of two tasks:

1. Algorithmically recognizing the unusual patterns or attributes of a number of types within the training data set.
2. Development of a classifier module that will be adept at classifying the newly provided data into either of the two classes, namely promoters and non-promoters.

Specifically, we learn sequence motifs that discriminate positive from negative examples, and an effective method to apply the acquired knowledge in order to discriminate between future data. The learning relies on novel probabilistic models.

As for now, we cannot bet on any of the machine learning techniques to be capable enough to provide 100% accurate results for the promoter prediction problem. Though, it makes sense to use such techniques for the very reason: it saves us a lot of time as well as resources by reducing the size of the set of probable promoters, which may further be tested using the conventional techniques known to provide fully accurate results. No doubt, the consideration is to make such statistical computer aided techniques as accurate as the conventional techniques available. And this is the direction where machine learning is heading.

The foundations of Support Vector Machines (SVM) have been developed by Vapnik (1995) and are gaining popularity due to many attractive features, and promising empirical performance. The formulation embodies the Structural Risk Minimization (SRM) principle, which has been shown to be superior [6] to traditional Empirical Risk Minimization (ERM) principle, employed by conventional neural networks. SRM

minimizes the upper bound on the expected risk, as opposed to ERM that minimizes the error on the training data. It is this difference which equips SVM with a greater ability to generalize, which is the goal in statistical learning.

Furthermore, the use of kernels makes the work even more interesting. There are a variety of kernels that can be employed for the purpose, as mentioned in the preceding chapter. The obvious question that arises is that with so many different mappings to choose from, which is the best for a particular problem? It is a known fact that any particular kernel can provide good enough results for a specific task. The polynomial kernel has done its part in the promoter recognition field, and gained sufficient popularity, just before it got a contender known as radial basis function(Gaussian kernel). They both still occupy prominent positions in the classification tasks in bioinformatics field. Our aim is to testify the claims providing them such a good reputation among the researchers, and find out which one is a better contender among the two. To make it reasonable, we also included the linear kernel in this study. But wait, our problem does not end here. It's not just the various kernels we are concerned with. What good is a kernel function without proper parameters fed to it? So we need to test all these kernel functions with a range of parameters and look for the ones that suit them particularly. There is no such hard and fast rule that can tell us about suitable parameter values for any of the kernels for any specified task. So the way to go is to try a kernel with different parameters and to test their efficiency on the provided data.

A wise idea will be to use cross validation, because when a clustering program is created in a supervised situation, it is necessary to be sure that it can perform in an unsupervised situation. In ***cross-validation***, a portion of the data is set aside as training data leaving the remainder as testing data. The quality of performance of the program on the testing data reflects how well it would perform in an unsupervised setting.

The ***holdout method*** is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set (it has never seen these output values

before). The errors it makes are accumulated which are used to evaluate the model. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made.

K-fold cross validation is one way to improve over the holdout method. The data set is divided into *k* subsets, and the holdout method is repeated *k* times. Each time, one of the *k* subsets is used as the test set and the other *k-1* subsets are put together to form a training set. Then the average error across all *k* trials is computed. The advantage of this method is that it matters less how the data gets divided. Greater the value of K, better it is.

But again, it will get too exhaustive if we use K-fold cross validation for a reasonable value of k. So we instead split the available data into train and test sets randomly, because of which there is no bias of any particular type of data being predominant in either train or test dataset.

Chapter 5
Material and Method

In this chapter, we shall propose our prediction method and explain how we get our datasets. In brief, we tackle the following classification problem: The input consists of a training set of sequences with positive and negative examples. A subset of examples will be randomly selected from the data set to be used for testing the classifier. The goal is to determine the kernel for classifier that will best discriminate between positives and negatives on the test set. There will be a two-phase scheme for this problem: In the first phase we use the training data to learn *attributes* (features) that are prevalent in the positive sequences compared to the negative ones. In the second phase we train our support vector machine (SVM) for the classification problem using the learned attributes as sequence features. We will compare our results in the following chapter.

5.1 Datasets

We take the *E. coli* sequences as our datasets from *http://regulondb.ccg.unam.mx* provided by [10]. The dataset contains 950 sample promoter sequences. Their lengths are more than 200 nucleotides. But for the sake of simplicity we cut them short to 81, by considering each sequence from position -60 to +21. It is believed that this is the range that will influence our result, as promoters lie in this range, and transcription initiation takes place starting from position 1. We generate the negative data set ourselves with the help of a small C program which generates the sequences with equal probability of occurrence of either of 'A', 'C', 'G', or 'T'.

A DNA sequence consists of four types of nucleotides: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). The promoter prediction program tries to use many kinds of statistical methods to find significant features of promoters, and then check if these features are useful to classify the difference between the promoter and the non-promoter

sequences. In our work, we do not try to find new obvious features, but try to find a classification method by performing some operations on sample sequences.

As we mentioned, a DNA sequence consists of nucleotides (A, C, G, and T) and we need to convert it into some form that can be easily interpreted by our promoter recognition program. So we develop a simple computer program that will convert a DNA sequence into a string of binary numbers. The notation followed will be: 'A' = "0001", 'C' = "0010", 'G' = "0100", 'T' = "1000". This program will provide us with an equivalent binary notation (according to the terminology proposed) for a set of DNA sequences, once we feed a '.txt' file to this program containing the sequences, each separated by a line feed. The output from the program will be a '.txt' file consisting of the binary coded DNA sequences, each separated by a linefeed. And we generate our negative data set itself in the binary form.

The data in final text files will be consumed by the learning machine, to interpret the positive as well as negative examples and learn from them.

5.2 Classification Program

We use the implementation of support vector machine in C code made available for free non-commercial use by Thorsten Joachims [7] on *http://svmlight.joachims.org/*. It's named SVM Light, and we will use the classification modules for our purpose. We will use this code in Matlab Environment, and develop a wrapper function which will be an interface to the C modules required. Our developed Matlab code will allow us to access all the modules needed for data input, transformation, training, classifier development, testing from a single interface and provide us the results in the form desired. We name our Matlab wrapper module as "wrapperSVM", and changes in the required parameters can be made in the same file, as well as required kernel can also be set in this very module.

5.3 Methodology

The two common measures of classifier performance are sensitivity and specificity. Sensitivity (also known as recall) is the fraction of positive examples that are correctly retrieved, while specificity is the fraction of negative examples that are correctly retrieved. The former measure describes the effectiveness of the classifier at finding the examples of interest, while the latter characterizes the performance of the classifier at discarding the other examples. So we are looking for a SVM kernel which yields better recall values, while still maintaining a good specificity of the system.

We will use the same data set for all the kernels, and try to find out suitable parameter values for each of them. Meanwhile, we will list the performance of each of the kernels for all the parameter values we experiment with, though we will compare the different kernels on the basis of the best results we acquire from them, whatever the parameters' values may be.

Initially, we vary the parameter 'C' which denotes the cost of wrong predictions from 1 to 100 (at 1, 10, 25, 50, and 100) for all the kernels. For polynomial kernel, we vary the degree of polynomial from 1 to 5 and test for all combinations of degree and cost. In the same manner, we vary the value of gamma for rbf kernel starting from 0.1, 0.01, 0.001, 0.0001, etc. As we get to know better about the suitability of parameter values to various kernels, we will change our track and opt for the optimum parameter values for each.

For each of the combination of parameter values, we will mention in a tabular form, the number of support vectors, the number of true positives, false negatives, true negatives, and false positives as well as the sensitivity and specificity of each. And finally we will report the results on the basis of the best performance we achieve form each of the kernels.

Conclusions and Future Scope

We compared the performance of linear, polynomial, and rbf (radial basis function) kernels using the support vector machine SVM Light. And rbf kernel is found to be way ahead of the others, as can be seen in the results provided here in a tabulated form. We experimented on all the kernels using varying value sets of parameters, and all the results achieved while the processes are mentioned herewith. The key words used for representation are as follows.

C - Cost of wrong predictions

SV - Number of Support Vectors used.

TP - True positive predictions

FP - False positive predictions

TN - True Negative predictions

FN - False negative predictions

Sensitivity - TP/(TP+FN). It is the fraction of positive examples that are correctly retrieved.

Specificity - TN/(TN+FP). It is the fraction of negative examples that are correctly retrieved.

Linear Kernel

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	857	792	158	824	126	0.8336	0.8673
2	856	791	159	826	124	0.8326	0.8694
5	855	792	158	825	125	0.8336	0.8684
10	849	789	161	826	124	0.8305	0.8694
25	852	788	162	825	125	0.8294	0.8684
50	851	789	161	826	124	0.8305	0.8694
100	852	789	161	825	125	0.8305	0.8684

Polynomial kernel (Degree = 1)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	854	792	158	824	126	0.8336	0.8673
5	851	792	158	825	125	0.8336	0.8684
10	851	788	162	825	125	0.8294	0.8684
25	851	789	161	825	125	0.8305	0.8684
50	850	789	161	825	125	0.8305	0.8684
100	849	789	161	825	125	0.8305	0.8684
125	851	789	161	825	125	0.8304	0.8684

Polynomial kernel (Degree = 2)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1535	950	0	950	0	1	1
5	1535	950	0	950	0	1	1
10	1535	950	0	950	0	1	1
25	1535	950	0	950	0	1	1
50	1535	950	0	950	0	1	1
100	1535	950	0	950	0	1	1

Polynomial kernel (Degree = 3)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1783	950	0	950	0	1	1
5	1783	950	0	950	0	1	1
10	1783	950	0	950	0	1	1
25	1783	950	0	950	0	1	1
50	1783	950	0	950	0	1	1
100	1783	950	0	950	0	1	1

Polynomial kernel (Degree = 4)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1894	950	0	950	0	1	1
5	1894	950	0	950	0	1	1
10	1894	950	0	950	0	1	1
25	1894	950	0	950	0	1	1
50	1894	950	0	950	0	1	1
100	1894	950	0	950	0	1	1

Polynomial kernel (Degree = 5)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1897	950	0	950	0	1	1
5	1897	950	0	950	0	1	1
10	1897	950	0	950	0	1	1
25	1897	950	0	950	0	1	1
50	1897	950	0	950	0	1	1
100	1897	950	0	950	0	1	1

RBF kernel (Gamma = 0.1)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1897	950	0	950	0	1	1
5	1897	950	0	950	0	1	1
10	1897	950	0	950	0	1	1
25	1897	950	0	950	0	1	1
50	1897	950	0	950	0	1	1
100	1897	950	0	950	0	1	1

RBF kernel (Gamma = 0.001)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1614	704	246	845	105	0.7410	0.8894
5	1220	774	176	809	141	0.8147	0.8515
10	1132	794	158	822	128	0.8357	0.8652
25	1031	813	137	844	106	0.8557	0.8884
50	1021	834	116	873	77	0.8778	0.9189
100	1090	880	70	911	39	0.9263	0.9589

RBF kernel (Gamma = 0.0001)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1089	778	172	806	144	0.8189	0.8484
5	1788	699	251	838	112	0.7357	0.8821
10	1583	709	241	838	112	0.7463	0.8821
25	1352	754	196	813	137	0.7963	0.8557
50	1187	768	182	805	145	0.8084	0.8473
100	1089	778	172	806	144	0.8189	0.8484
125	1069	778	172	810	140	0.8189	0.8526
150	1037	781	169	815	135	0.8221	0.8578
175	1017	782	168	814	136	0.8231	0.8568
200	1003	786	164	815	135	0.8273	0.8578
225	992	788	162	816	134	0.8294	0.8589
250	978	785	165	814	136	0.8263	0.8568
275	970	786	164	819	131	0.8273	0.8621
300	959	786	164	820	130	0.8273	0.8631
325	954	784	166	825	125	0.8252	0.8684
350	944	787	163	822	128	0.8284	0.8652
375	949	786	164	824	126	0.8273	0.8673
400	944	789	161	826	124	0.8305	0.8694
700	918	791	159	833	117	0.8326	0.8768
1000	908	799	151	835	115	0.8410	0.8789
2000	902	809	141	846	104	0.8515	0.8905
2100	904	810	140	847	103	0.8526	0.8915
2500	910	813	137	850	100	0.8557	0.8947
3000	918	819	131	851	99	0.8621	0.8957
4000	938	830	120	861	89	0.8736	0.9063

RBF kernel (Gamma = 0.0005)

Cost‘C’	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1803	697	253	838	112	0.7336	0.8821
5	1361	754	196	816	134	0.7936	0.8589
10	1204	772	178	807	143	0.8126	0.8494
25	1079	787	163	816	134	0.8284	0.8589
50	1000	798	152	833	117	0.8400	0.8768
100	970	810	140	843	107	0.8526	0.8873
125	969	819	131	849	101	0.8621	0.8936
150	976	826	124	862	88	0.8694	0.9073
175	975	830	120	866	84	0.8736	0.9115
200	984	837	113	877	73	0.8810	0.9231
225	994	842	108	880	70	0.8863	0.9263

RBF kernel (Gamma = 0.00001)

Cost'C'	SV	TP	FN	TN	FP	Sensitivity	Specificity
1	1900	732	218	800	150	0.7705	0.8421
5	1900	732	218	801	149	0.7705	0.8431
10	1900	732	218	801	149	0.7705	0.8431
25	1898	700	250	836	114	0.7368	0.8800
50	1785	698	252	839	111	0.7347	0.8831
100	1580	709	241	833	117	0.7463	0.8768
200	1396	748	202	818	132	0.7873	0.8610
500	1184	766	184	805	145	0.8063	0.8473
1000	1090	776	174	802	148	0.8168	0.8442
2000	1001	783	167	813	137	0.8242	0.8557
5000	923	783	167	823	127	0.8242	0.8663
1000	895	787	163	830	120	0.8248	0.8736
15000	878	788	162	829	121	0.8294	0.8726
20000	869	787	163	830	120	0.8284	0.8736
25000	866	792	158	829	121	0.8336	0.8726
30000	860	795	155	829	121	0.8368	0.8726
35000	862	793	157	831	119	0.9073	0.8747
36000	861	792	158	829	121	0.8336	0.8726

Interpretation

Best performance results:

	Sensitivity	**Specificity**
Linear kernel	0.8305	0.8694
Polynomial kernel	0.8305	0.8684
RBF kernel	0.8515	0.8905

Liner kernel performs the best at value of C = 50. As we increase the cost, overfitting occurs owing to increase in number of support vectors. The performance increases with increase in the cost of overfitting with 50 being the limit.

Polynomial kernel performs best at degree 1 and 'C' = 100, after which overfitting occurs. There is a slight increase in performance as the cost of overfitting increases, with a limit of 100.

RBF kernel performs best at value of gamma to be 0.0001 and C = 2000. Again, with a further increase in value of 'C' or a decrease in gamma, overfitting occurs. There is a visible increase in performance of RBF kernel as gamma reduces or cost increases, with the mentioned values to be the threshold.

As can be seen from the results, we have empirically proved that for promoter classification problem the polynomial kernel does not work well for any degree greater than 1, after which it ends up using almost all the elements of the training data as support vectors and overfits heavily. So it is nothing much than a linear kernel itself, and thus fades out of the competition with RBF kernel. RBF kernel emerges as a true winner in our work, and its performance is far better than that of polynomial or linear kernel, and SVM proves to be a wise method for the promoter prediction problem.

Future Scope

In this work we have found out that the polynomial kernel does not work so good for promoter recognition problem and a very well suited approach to the same is using SVM with RBF kernel. Work is required in this direction with techniques other than SVM, such as neural networks, boosting, bagging, RBF networks and similar methods to look for the most suitable one for promoter recognition problem.

We generated the non-promoter sequence data ourselves, as is done by many researchers today; the reason being, there is no technique available at present to know if a sequence is actually “non-promoter”. If we take sequences from real genomes that are so far believed to be non-promoters, it will be incorrect approach, as the same sequence may be proved to be a promoter at a later stage by bio-chemical testing. Actually, it is believed that there are more promoters to be discovered. So we need some way to prove if a sequence is a non-promoter.

References

[1] Bennett, Kristin P. and Campbell, Colin (2000)**,** "Support Vector Machines: Hype or Hallelujah?"**,** SIGKDD Explorations 2(2)**,** ACM SIGKDD

[2] Blattner F.R., Plunkett III G., Bloch C.A., Perna N.T., Burland V., Riley M., Collado-Vides J., Glasner J.D., Rode C.K., Mayhew G., Gregor J., Davis N.W., Kirkpatrick H. A., Goeden M.A., Rose D.J., Mau B., Shao Y. (1997), "The complete genome sequence of Escherichia coli K-12", Science 5 September 1997 277, p. 1453-1462

[3] Boser, B. E., Guyon, I. M., and Vapnik, V. (1992), "A training algorithm for optimum margin classifiers", Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, ACM Press, New York, p. 144-152

[4] Burges, C. J. C. (1998), "A tutorial on support vector machines for pattern recognition", Data Mining and Knowledge Discovery, 2(2), Kluwer Academic Publishers, Boston, p. 121-167

[5] Cortes, C. and Vapnik, V. (1995), "Support-vector networks", Machine Learning, 20(3), Springer Netherlands, p. 273-297

[6] Gunn S.R., Brown M., and Bossley K.M. (1997), "Network performance assessment for neurofuzzy data modeling", Intelligent Data Analysis, volume 1208 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, p. 313–323

[7] Joachims, T. (1998), **"**Making large-scale support vector machine learning practical", Advances in Kernel Methods: Support Vector Machines by B. Scholkopf, C. Burges, A. Smola, MIT Press, Cambridge

[8] Nilsson, Nils J. 1996, "Introduction to Machine Learning – An early draft to a proposed textbook", Robotics Laboratory, Dept. of Computer Science, Stanford University, Stanford, California

[9] Platt, J. (1998), "How to implement SVMs", IEEE Intelligent Systems Magazine, Trends and Controversies, Marti A. Hearst

[10] Salgado H, Gama-Castro S, Peralta-Gil M, Diaz-Peredo E, Sanchez-Solano F, Santos-Zavaleta A, Martinez-Flores I, Jimenez-Jacinto V, Bonavides-Martinez C, Segura-Salazar J, Martinez-Antonio A, Collado-Vides J., RegulonDB (version 5.0): Escherichia coli K-12 transcriptional regulatory network, operon organization, and growth conditions, Nucleic Acids Res. 2006 Jan 1;34(Database issue):D394-7

[11] Salton G. and McGill M. J. (1983)**, "**Introduction to Modern Information Retrieval", McGraw Hill, New York

[12] Scholkopf, C., Burges, J. C., and Smola, A. J. (1999), "Advances in Kernel Methods", MIT Press, Cambridge

[13] Shawe-Taylor, J. and Cristianini, N. (1999), "Further results on the margin distribution", Twelfth Annual Conference on Computational Learning Theory, California

[14] Thieffry D., Salgado H., Huerta A.M., and Collado-Vides J. (1998), "Prediction of transcription regulatory sites in the complete genome of Escherichia coli", Bioinformatics 14(5), p. 391-400

[15] Vapnik, V. (1998), "Statistical Learning Theory", Wiley, New York

Appendix A
Terms and Definitions

Here are some basic definitions and terms that a beginner needs to be versed with before starting with the theory of support vector machines.

A.1 A **hyperplane** is a concept in geometry. It is a generalization of the concept of a plane. In a one-dimensional space (such as a line), a hyperplane is a point; it divides a line into two rays. In two-dimensional space (such as the *xy* plane), a hyperplane is a line; it divides the plane into two half-planes. In three-dimensional space, a hyperplane is a ordinary plane; it divides the space into two half-spaces. This concept can also be applied to four-dimensional space and beyond, where the dividing object is simply referred to as a hyperplane

In the general case, a hyperplane is a higher-dimensional analog of a (two-dimensional) plane in three-dimensional space. A hyperplane in *n*-dimensional space can be described by a non-degenerate linear equation of the following form:

$a_1x_1 + a_2x_2 + ... + a_nx_n = b.$

Here, *non-degenerate* means that not all the a_i are zero. If b=0, one obtains a linear hyperplane, which goes through the origin of the space.

The two half-spaces defined by a hyperplane in *n*-dimensional space are:

$a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$

and

$a_1x_1 + a_2x_2 + ... + a_nx_n \geq b.$

A.2 Statistics is a mathematical science pertaining to collection, analysis, interpretation and presentation of data. Given a collection of data, statistics may be employed to summarize or describe the data; this use is called descriptive statistics. In addition, patterns in the data may be modeled, in a way that accounts for randomness and uncertainty in the observations, in order to draw inferences about the larger population; this use is called inferential statistics. Both of these uses may be termed applied statistics. There is also a discipline of mathematical statistics concerned with the theoretical basis of the subject.

A.3 Machine learning is an area of artificial intelligence concerned with the development of techniques which allow computers to "learn". More specifically, machine learning is a method for creating computer programs by the analysis of data sets. Machine learning overlaps heavily with statistics, since both fields study the analysis of data, but unlike statistics, machine learning is concerned with the algorithmic complexity of computational implementations.

A.4 Supervised learning is a machine learning technique for creating a function from training data. The training data consist of pairs of input objects (typically vectors), and desired outputs. The output of the function can be a continuous value (called regression), or can predict a class label of the input object (called classification). The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples (i.e. pairs of input and target output). To achieve this, the learner has to generalize from the presented data to unseen situations in a "reasonable" way.

A.5 Unsupervised learning is a method of machine learning where a model is fit to observations. It is distinguished from supervised learning by the fact that there is not *a priori* output. In unsupervised learning, a data set of input objects is gathered. Unsupervised learning then typically treats input objects as a set of random variables. It is very useful for data compression: fundamentally, all data compression algorithms either explicitly or implicitly rely on a probability distribution over a set of inputs.

A.6 Optimization

In mathematics, the term **optimization** refers to the study of problems that have the form

Given: a function $f : A \longrightarrow \mathbf{R}$ from some set A to the real numbers

Sought: an element x_0 in A such that $f(x_0) \leq f(x)$ for all x in A ("minimization") or such that $f(x_0) \geq f(x)$ for all x in A ("maximization").

Such a formulation is sometimes called a mathematical program (a term not directly related to computer programming). Many real-world and theoretical problems may be modeled in this general framework.

Typically, A is some subset of the Euclidean space $\mathbf{R}^n$, often specified by a set of *constraints*, equalities or inequalities that the members of A have to satisfy. The elements of A are called *feasible solutions*. The function f is called an objective function, or cost function. A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an *optimal solution*.

The domain A of f is called the search space, while the elements of A are called candidate solutions or feasible solutions.

A.7 Pattern recognition is a field within the area of machine learning. Alternatively, it can be defined as "the act of taking in raw data and taking an action based on the category of the data" [1]. As such, it is a collection of methods for supervised learning.

Pattern recognition aims to classify data (patterns) based on either *a priori* knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional space.

A complete pattern recognition system consists of a sensor that gathers the observations to be classified or described; a feature extraction mechanism that computes numeric or symbolic information from the observations; and a classification or description scheme

that does the actual job of classifying or describing observations, relying on the extracted features.

The classification or description scheme is usually based on the availability of a set of patterns that have already been classified or described. This set of patterns is termed the training set and the resulting learning strategy is characterized as supervised learning. Learning can also be unsupervised, in the sense that the system is not given an *a priori* labeling of patterns, instead it establishes the classes itself based on the statistical regularities of the patterns.

The classification or description scheme usually uses one of the following approaches: statistical (or decision theoretic), syntactic (or structural). Statistical pattern recognition is based on statistical characterizations of patterns, assuming that the patterns are generated by a probabilistic system. Structural pattern recognition is based on the structural inter-relationships of features.

Typical applications are automatic speech recognition, classification of text into several categories (e.g. spam/non-spam email messages), the automatic recognition of handwritten postal codes on postal envelopes, or the automatic recognition of images of human faces.

A.8 Statistical classification is a statistical procedure in which individual items are placed into groups based on quantitative information on one or more characteristics inherent in the items (referred to as traits, variables, characters, etc) and based on a training set of previously labeled items.

Formally, the problem can be stated as follows: given training data $\{(\mathbf{x_1}, y), \ldots, (\mathbf{x_n}, y)\}$ produce a classifier $h : \mathcal{X} \to \mathcal{Y}$ which maps an object $\mathbf{x} \in \mathcal{X}$ to its classification label $y \in \mathcal{Y}$. For example, if the problem is filtering spam, then $\mathbf{x_i}$ is some representation of an email and *y* is either "Spam" or "Non-Spam".

Statistical classification algorithms are typically used in pattern recognition systems.

Support Vector Machines are a set of related supervised learning methods used for classification and regression. When used for classification, the SVM algorithm creates a hyperplane that separates the data into two classes with the maximum-margin. Given training examples labeled either "yes" or "no", a maximum-margin hyperplane is identified which splits the "yes" from the "no" training examples, such that the distance between the hyperplane and the closest examples (the margin) is maximized.

Appendix B

Programs used for Data Generation and Conversion

Code used for converting genome sequences into equivalent binaries

```
#include <iostream.h>
#include <fstream.h>
#include <ctype.h>

void main()
{
        int BASE ;

        char infileName[40];
        char outfileName[40];

        ifstream infile;
        ofstream outfile;

        cout<<"Base : ?" ; cin >> BASE;
        cout<<"\nPlease give short file names (not more than 40 chars)\n";
        cout <<"\nInput file (ATGC chars) : ";
        cin>> infileName;
        cout <<"\nOutput file (binary) : ";
        cin>> outfileName;


  infile.open(infileName);
  outfile.open(outfileName);

  int count=0;
  char ch;

  while(! infile.eof())
  {
        infile>>ch;
    ch = toupper(ch);
    if( (ch=='A')||(ch=='C')||(ch=='G')||(ch=='T') )
    {
      if(ch=='A')
        outfile<<"0 0 0 1 ";
```

```
        else if(ch=='C')
          outfile<<"0 0 1 0 ";
        else if(ch=='G')
          outfile<<"0 1 0 0 ";
        else if(ch=='T')
          outfile<<"1 0 0 0 ";
        count++;
        if(count==BASE)
                    {
          outfile<<endl<<flush;
                              count=0;
                    }
      }
    }

    infile.close();
    outfile.close();

}
```

Code used for generating non promoter sequences

```
#include <iostream.h>
#include <fstream.h>
#include <ctype.h>
#include <time.h>
#include <stdlib.h>

void main()
{
          int BASE=81, r ;
          char outfileName[40];

          ofstream outfile;

          cout<<"\nPlease give short file names (not more than 40 chars)\n";
          cout <<"\nOutput file (binary) : ";
          cin>> outfileName;


    outfile.open(outfileName);

    int count=0;
    int max=0;
```

```
  char ch;

  randomize();

  while(max<950)
  {
    r = rand() % 40;
    if( (r>=0) && (r<40) )
    {
        if(r<10)
                outfile<<"0 0 0 1 ";
        else if(r<20)
                outfile<<"0 0 1 0 ";
        else if(r<30)
                outfile<<"0 1 0 0 ";
        else
                outfile<<"1 0 0 0 ";

        count++;
        if(count==BASE)
                {
                outfile<<endl<<flush;
                        count=0; max++;
                }
    }
  }

  outfile.close();

}
```

Paper(s) Communicated/Accepted/Published

- Jasneet Singh, R S Salaria. **"Analysis of E. coli Promoters Using Support Vector Machine"**, 1st International Conference on Digital Information Management (ICDIM) December 06-08, 2006, Christ College, Bangalore, India. **(Communicated)**