AN ABSTRACT OF THE THESIS OF

Shriprakash Sinha for the degree of Master of Science in Computer Science presented on March 5, 2004.

Title: Leaf Shape Recognition via Support Vector Machines with Edit Distance Kernels

Abstract approved:

Signature redacted for privacy.

Dr. Thomas G Dietterich

Edit distances are a well-established technique for classification problems. They have been employed successfully in many classification problems including chromosome classification and hand-written digit recognition. Virtually all machine learning algorithms represent the objects to be classified as vectors of features. However, edit distances provide only a measure of the difference between two objects—they do not provide a feature-based representation. Recently, kernel-based algorithms such as support vector machines (SVMs) have been developed. A kernel is a measure of similarity between two objects. This thesis explores various ways in which edit distances can be converted into kernels and combined with support vector machines to solve difficult pattern recognition problems. The thesis compares the performance of SVMs, the k-nearest neighbor algorithm, and the weighted k-nearest neighbor algorithm on a problem of leaf shape classification. The goal of the leaf classification problem is support content-based image retrieval from the Oregon State University Herbarium, which is a collection of plant specimens.

The thesis shows that SVMs and standard k-nearest neighbor provide high classification accuracy. SVMs, with an edit distance kernel, provide the most accurate method.

Keywords: Edit Distance Kernels, Pattern Recognition, Nearest Neighbour, Weighted Nearest Neighbour, Support Vector Machines, Content Based Image Retreival.

Leaf Shape Recognition via Support Vector Machines with Edit Distance Kernels

by

Shriprakash Sinha

A Thesis

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed March 5, 2004
Commencement June 2004

ACKNOWLEDGMENT

I would like to thank my advisor Professor Thomas G. Dietterich for guiding me in this research. Had it not been for his constant supervision, despite constrained time schedule, I would not have been able to succeed. I also owe a sincere thanks to Pengcheng Wu my colleague and Ashit Gandhi my predecessor on the project, with whom I have discussed and talked on wide range of topics concerning pattern recogntion. Also, thanks to NSF, for supporting me on this research.

I would also like to thank Professor Prasad Tadepalli, Professor Bella Bose and Professor C.K. Koc under whom I have taken courses. Lastly, my sincere gratitude towards my parents, who helped and supported me in fulfilling my desire of higher studies abroad.

Shriprakash Sinha

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

Figure Page

LIST OF FIGURES (Continued)

Figure Page

To Divine God,

my professor & mentor Dr. Thomas G. Dietterich,

and my parents Rita Sinha & Prabhat Sinha.

Leaf Shape Recognition via Support Vector Machines with Edit Distance Kernels

1. INTRODUCTION

1.1. Background and Motivation

Content Based Image Retrieval (CBIR henceforth) is a fast emerging field of research in the area of pattern recognition. The basic idea of CBIR is to retrieve maximum information based on the content or the feature in the image. As CBIR systems are being developed to store huge amounts of data in the form of images, a question comes to mind how to classify a new image into an existing or a new category. The question itself hints at pattern recognition and classification as a solution. In this endeavor, we have tried to find a solution to the above problem and introduce a new technique for feature selection, to reduce the time required to recognize and classify. The next section introduces some of the problems we try to tackle in order to materialize the effort.

1.2. Problem Statement

To understand the problem at the grass root level, we initially developed a small CBIR system which is a collection of images of six different species of plant leaves obtained from the OSU Herbarium [16]. With many images in our possession, it was impossible to classify a new image by just doing a visual search and identification of the best corresponding species. To solve this problem, we

came up with an idea of applying a supervised learning technique. Since the technique demands the presence of data in the form of (X_i, y_i), where X_i is a vector of features that significantly describe a sample and y_i its corresponding label, we had to come up with an efficient way of representing features. Another question that was necessary to ponder upon was how to select features that revealed maximum information and yet be present in small amount. A solution to all these we suppose would help us to identify the category to which an image belongs, incurring minimum loss in time and developing a highly accurate and automated CBIR system.

Various algorithms are present in machine learning and we choose three algorithms namely, k-Nearest Neighbor, Weighted k-Nearest Neighbor and Support Vector Machines. In the following sections we will try to analyze and develop a solution, methodically employing the mentioned three algorithms and then try to present some of the advantages gained.

2. IMAGE PROCESSING OF LEAF SAMPLES

2.1. Basic Idea

We consider boundary shape as the main criterion for representing the information provided by the digitized images of plant leaves. The idea is to extract a 1D representation of the 2D image. Other transformations of higher complexity like getting smooth curvature using B-splines and segmentation of shapes as described by Petrakis et al. [9], to preserve the accuracy of the information, could be used; but are not under consideration in this research. In order to achieve this, a boundary detection process is performed, where the boundary is extracted from a leaf silhouette. A scalar boundary transformation follows, where it is converted into a series of angles measured along the edge of the leaf.

2.2. Edge Detection

The extraction of boundary is a process that comes after a point on the edge of the leaf has been detected. Noise, in the form of seeds, flowers, stems and annotations about the leaf species are also present in the image. We preprocess the image and do a morphological erosion of the noise by tuning a threshold parameter. The threshold function used here is a trivial one, based on the simple idea that leaves are generally green or brown in color and blue is very rare. So the threshold function employed here is the following:

$$\frac{Red_{pixel} + Green_{pixel}}{Blue_{pixel}} > \theta \tag{2.1}$$

where θ is the tuning parameter. (2.2)

If the computed value at a pixel is greater than the tuning parameter, then the pixel is turned ON else it is turned OFF.

2.3. Edge Extraction

Hitherto, we used the idea as developed by my predecessor Ashit Gandhi. The edge extraction process is somewhat different than the way used by Ashit. We will be pointing out the loopholes that we found in his technique and provide a better solution. We begin with an example considering a leaf image of the circinatum species as shown in figure 2.1.

FIGURE 2.1. An Isolated Circinatum Leaf Sample

2.3.1. Old Idea

The boundary detection process begins when the raster hits an ON pixel in the image. To detect the next pixel on the boundary, a square window of size three is chosen and the traversal direction is fixed to be clockwise. Also to find the next pixel only four directions are searched for, i.e. right, left, up or down. This is the first disadvantage, as it is difficult to visualize and track the traversal direction. Figure 2.2, helps to show this idea.

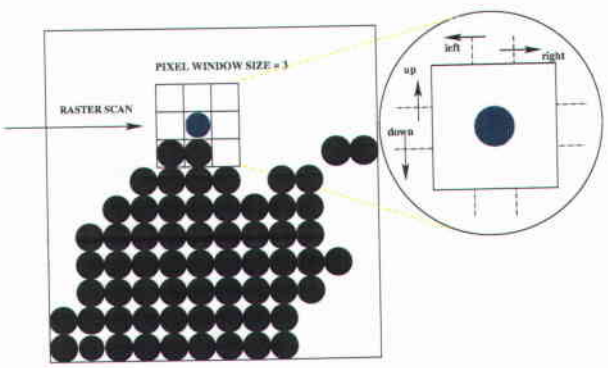

FIGURE 2.2. Four Directions to find pixel in Window of size = 3

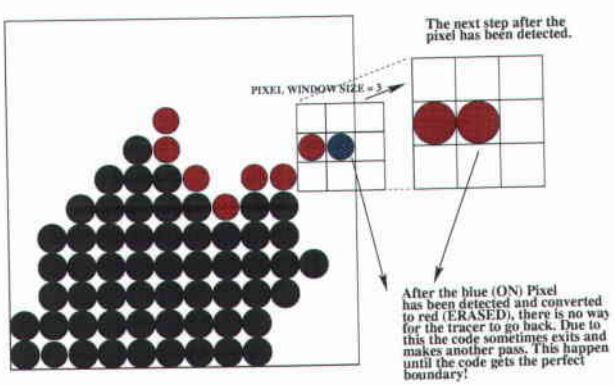

FIGURE 2.3. Edge Detection and Extraction employing Old Idea

The technique runs into trouble if it comes to a point after which it is unable to find an ON pixel. This is a major problem, as the detection process stops to trace the full boundary even if the image extraction was perfect. The stopping of the code can be visualized by having a look at figure 2.3. Figure 2.2 and figure 2.3 show an imaginary data used to elucidate the concept behind the technique. The old edge detecting process shows only a broken up boundary! We earlier mentioned the example of isolated circinatum leaf. Figure 2.4 shows the stage up to which the boundary detection and extraction process was performed. Thus several iterations through the image were performed before the final exact boundary was retrieved. This is definitely not an efficient method.

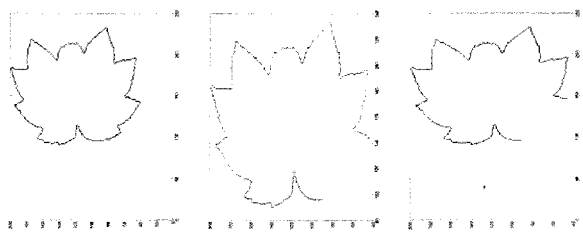

FIGURE 2.4. Extracted Boundary for $\theta = 3.25$

2.3.2. New Idea

We now propose a new idea to solve the problem encountered in the previous section. To increase the visualization power, we introduced the idea of having eight standard directions: North, NorthEast, East, SouthEast, South, SouthWest, West, NorthWest. Though the idea does come with a better way of viewing the directions, it does impose relatively more constraints while finding the right pixel. Figure 2.5 depicts these ideas.

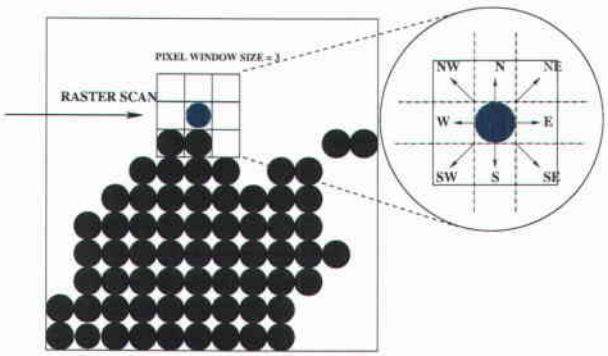

FIGURE 2.5. Eight Directions to find pixel in Window of size = 3

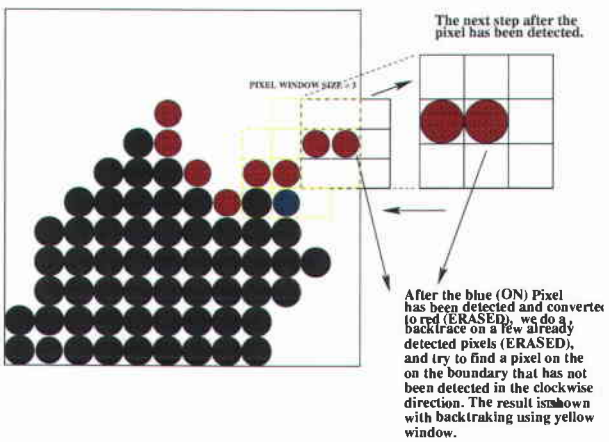

FIGURE 2.6. Edge Detection and Extraction employing New Idea

The previous technique never backtracks if it comes to a point after which it is unable to find an ON pixel. The new version integrates the backtracking technique where the edge detection process traces back a few already traversed pixels (ERASED) until it finds an ON pixel, in Clockwise direction, in a pixel window of size 3. Due to this, the process now finds the perfect boundary even at low threshold value. Figure 2.6 helps to visualize the backtracking method. Also there is only a single traversal for each isolated leaf. For the isolated leaf,

earlier taken as an example, figure 2.7 shows the single boundary retrieved in one iteration.

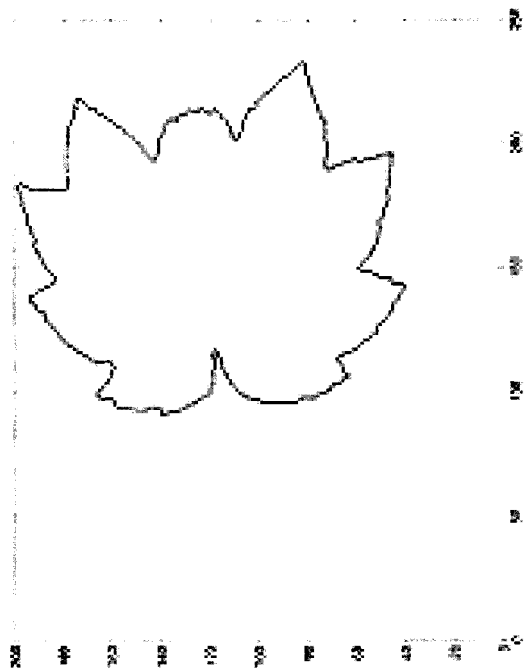

FIGURE 2.7. Extracted Boundary for $\theta = 3.25$

A few subtle points do arise, if one compares the full extracted boundary in figure 2.4 and figure 2.7. The latter technique does get hold of more information by detecting the right pixels, than the former technique. We think this may be due to the added constrains in the directions introduced in the window. Also the boundaries retrieved by the new technique are much sharper than those retrieved by the old technique. We show some of the retrieved boundaries for values of the tuning parameter θ, in figure 2.8. There are a few disadvantages, like the selection of threshold function and the range of values selected for the tuning

parameter. Since the threshold function employed is very trivial and not robust, it is very sensitive to the values of θ. Higher values, for example in the above case, say for $\theta = 3$, sometimes can cause poor behavior, due to too much information extraction. Thus care must be taken while tuning θ.

2.3.3. Dimensionality Reduction of Features from 2D to 1D

Once the boundary is extracted, the 2D representation is converted into a 1D representation in the form of a series of angles measured along the edge. Ashit achieved this by calculating the angle formed by lines joining two points ten pixels apart in opposite direction, to the pixel under consideration, on the edge boundary. We use his method.

Based on the ideas of Morgan [3], Wagner and Fischer [4] proposed the concept of edit distance by exploiting the mathematical definition of traces. The algorithm to compute edit distance forms the central idea of Dynamic Programming. Some algorithms and applications of edit distances can be found in Sellers [5], Marzal [6] and Bunke [7], [8]. We execute the dynamic time wrapping (DTW) algorithm (provided by Ashit Gandhi) to calculate the edit distance, $DP(i, j)$, (i and j being leaves of different or similar species), between two 1D representations. Introductory material can be found in Durbin et al. [10], that attacks the problem of matching in DNA, RNA and other biological sequence data.

This transformation of features present in 2D to 1D helps to reduce the amount of features and yet retain the relevant information.

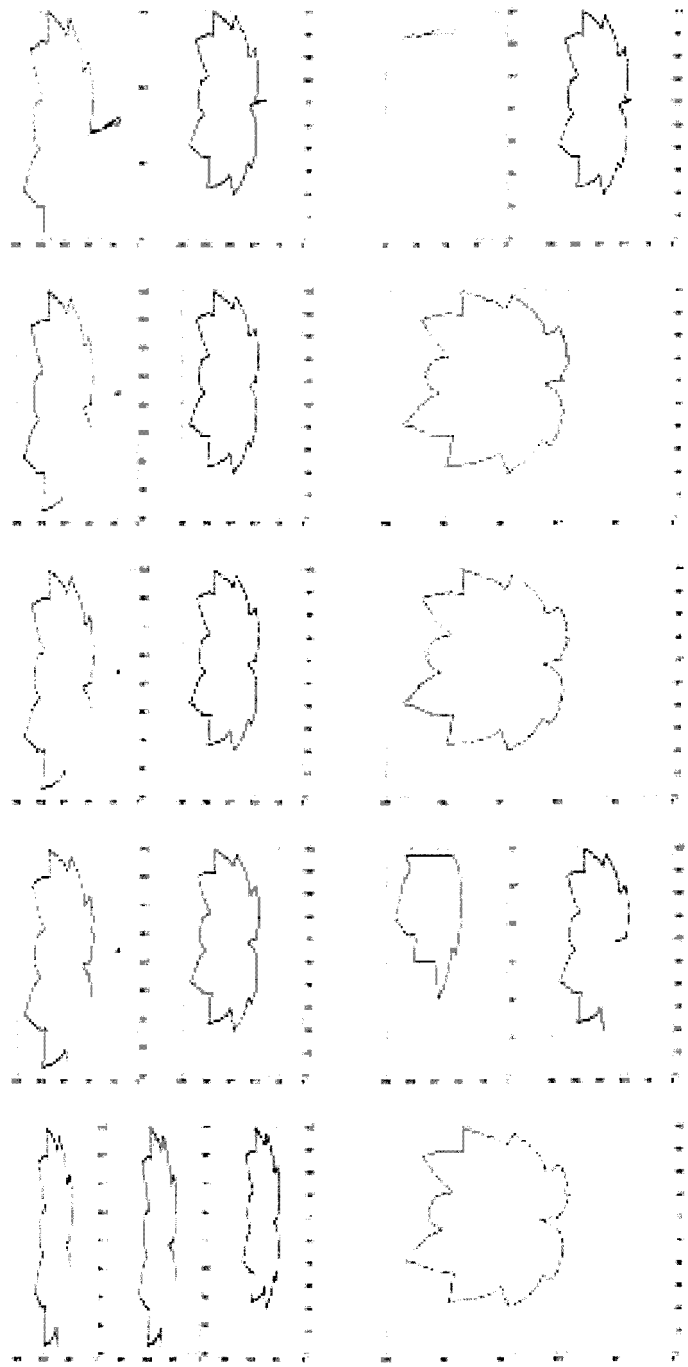

FIGURE 2.8. $\theta \in \{2.25, 2.50, 2.75, 3.00, 3.25\}$, Left column show boundaries from Old Technique and Right column show boundaries from New Technique

3. OPTIMIZATION PROBLEMS

3.1. Introduction

In the current and the next two chapters, we will be discuss the pattern recognition algorithms that we apply for classification purpose. The main focus in this chapter is on optimization problems, the concept of Lagrange multipliers and KKT conditions, which will be later used to explain the details about the SVMs.

3.2. Mathematical Formulation

Optimization problems arise in almost every area of engineering. The goal is to achieve an almost perfect and efficient result, while carrying out certain procedures of optimization. Our main source of reference on this topic derives from [17]. We will be using notations used in [17]. In mathematical terms the general form of optimization problem can be represented as :

$$\begin{aligned} \text{minimize} \quad &: \quad f(\mathbf{x});\ \mathbf{x} \in \mathbf{R}^n \\ \text{such that} \quad &: \ g_j(\mathbf{x}) \leq 0;\ \text{j} = 1,.....,\text{p} \\ &: \ h_j(\mathbf{x}) = 0;\ \text{j} = 1,.....,\text{q}. \end{aligned} \tag{3.1}$$

where f, g_j and h_j are the objective function, equality constraints and inequality constraints. Generally, the number of constraints is less than the number of variables used to formulate the optimization problem. For a problem to be linear, both the constraints and the objective function need to be linear. Quadratic problems require only the objective function to be quadratic, while the constraints

remain linear in formulation. Besides these, if any one of the functions is nonlinear, then the problem becomes nonlinear in nature. A graphical view of the types of the problems can be seen in fig. 3.1).

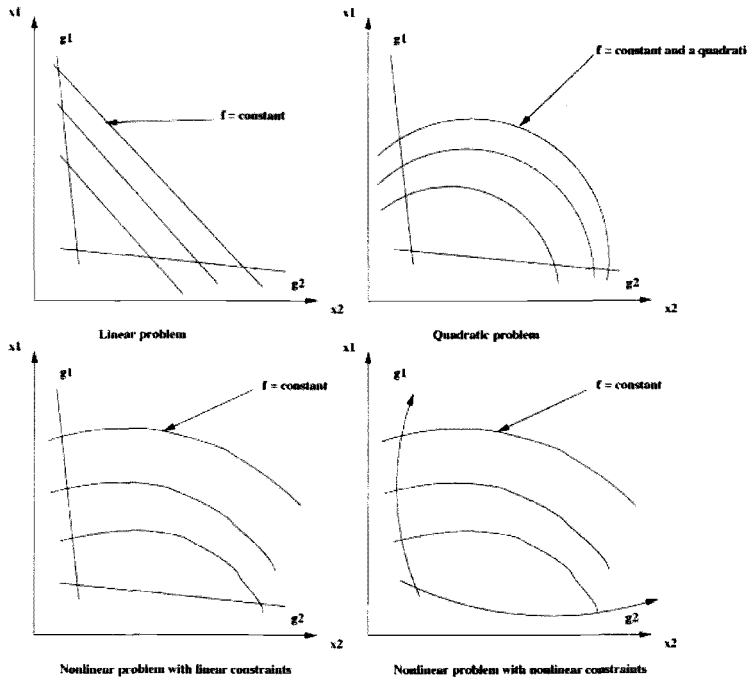

FIGURE 3.1. Kinds of optimization problems.

3.3. Lagrange Multipliers

In unconstrained optimization problems, where the first order derivatives are assumed continuous, the solution is found by solving:

$$\nabla_x f = \frac{\partial f}{\partial x_i} = 0;\ \text{i} = 1,.....,\text{n}. \tag{3.2}$$

where f is a function of x. Since most of the optimization problems are constrained, the concept of Lagrange multipliers is introduced in order to solve the

problem. Thus, the Lagrangian formulation, for Eqn. 3.1 becomes:

$$L(\mathbf{x}, \lambda, \mu) = f(\mathbf{x}) + \sum_{j=1}^{p} \lambda_j g_j(\mathbf{x}) + \sum_{j=1}^{q} \mu_j h_j(\mathbf{x}) \tag{3.3}$$

where L is the Lagrangian, $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are the vectors of the Lagrange multipliers for inequality and equality constraints, respectively.

Next comes the solving of the Lagrangian. We try to derive a solution in terms of variables used and show that the final solution achieved by Equ. 3.1 and Eqn. 3.3 remains the same. For the sake of derivation, we assume that each of the vectors $\boldsymbol{x}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ have a single element and also there exists a single optimal solution. We will then generalize the solution to vectors containing various elements. Let x^*, λ^* and μ^* be the optimal solution for the Lagrangian. Let $x^!$ be the optimal solution for $f(x)$. To begin with, our Lagrangian has the form:

$$L(x, \lambda, \mu) = f(x) + \lambda g(x) + \mu h(x) \tag{3.4}$$

Derivation:

- **Step 1:** Differentiate the Lagrangian in Eqn 3.4 w.r.t x and equate it to zero.

$$\frac{\partial L}{\partial x} = \frac{\partial f}{\partial x} + \lambda \frac{\partial g}{\partial x} + \mu \frac{\partial h}{\partial x} = 0 \tag{3.5}$$

- **Step 2:** Find x in terms of λ and μ, such that $x = x(\lambda, \mu)$.

- **Step 3:** Differentiate the Lagrangian in Eqn. 3.4 w.r.t λ and equate it to zero.

$$\frac{\partial L}{\partial \lambda} = g(x) = 0 \tag{3.6}$$

- **Step 4:** Differentiate the Lagrangian in Eqn 3.4 w.r.t μ and equate it to zero.

$$\frac{\partial L}{\partial \mu} = h(x) = 0 \tag{3.7}$$

- **Step 5:** Substitute $x(\lambda, \mu)$ in Equ. 3.6 and Eqn. 3.7 to get two equations in two unknowns λ and μ and solve to get the optimal values.

$$g(x(\lambda, \mu)) = 0 \tag{3.8}$$

$$h(x(\lambda, \mu)) = 0 \tag{3.9}$$

Let λ^*, μ^* be the solution. Substituting these in $x = x(\lambda, \mu)$, we get x^*.

- **Step 6:** Combining Eqn. 3.6 and Eqn. 3.7 in Eqn. 3.4, along with λ^*, μ^* and x^*, we have:

$$L(x^*, \lambda^*, \mu^*) = f(x^*) + \lambda^* g(x^*) + \mu^* h(x^*) = f(x^*) \tag{3.10}$$

Since, it is assumed that there exist only one optimal solution we have:

$$\begin{aligned} L(x^*, \lambda^*, \mu^*) &= f(x^*) = f(x^!) \\ x^* &= x^! \end{aligned} \tag{3.11}$$

Lastly, since $g(x)$ in Eqn. 3.6 is a inequality constraint, we have:

$$\begin{aligned} \lambda g(x) &= 0 \\ \lambda &\geq 0 \end{aligned} \tag{3.12}$$

3.4. Dual Functions

For sake of simplicity, let us for a moment ignore the equality constraint. Then the Lagrangian becomes:

$$L(x, \lambda, \mu) = f(x) + \lambda g(x). \tag{3.13}$$

It is sometimes easy to transform the Lagrangian into a simpler form, in order to find an optimal solution. We can represent the Lagrangian as a Dual function in such a manner that the optimal solution defined as minimum of $L(x, \lambda^*)$ w.r.t x where $\lambda = \lambda^*$, can be represented as the maximum of dual function $D(\lambda)$ w.r.t λ. For a given λ, the dual is evaluated by finding the minimum of $L(x, \lambda)$ w.r.t x. Thus to find the optimal point we evaluate:

$$\max_{\lambda} D(\lambda) = \min_{x} L(x, \lambda^*) \tag{3.14}$$

So the basic steps to solve the dual problem are as follows:

- **Step 1:** Minimize $L(x, \lambda)$ w.r.t x, and find x in terms of λ.
- **Step 2:** Substitute $x(\lambda)$ in L s.t. $D(\lambda) = L(x(\lambda), \lambda)$.
- **Step 3:** Maximize $D(\lambda)$ w.r.t λ.

3.5. Karush Kuhn Tucker Conditions

The derivation in the last part (Eqn. 3.4 to Eqn. 3.12)gives us a set of equations that need to be evaluated along with the consideration of constraints present. These set of equations and constraints in terms of the Lagrangian, form the Karush Kuhn Tucker Conditions. We give here the generalized KKT conditions and explain the necessary details.

$$\begin{aligned}
&\frac{\partial L}{\partial x_i} = \frac{\partial f}{\partial x_i} + \sum_{j=1}^{p} \lambda_j \frac{\partial g_j}{\partial x_i} + \sum_{j=1}^{p} \mu_j \frac{\partial h_j}{\partial x_i} = 0 \; : \; i = 1, \ldots, n \\
&\frac{\partial L}{\partial \lambda_j} = g_j(\mathbf{x}) = 0 \; : \; j = 1, \ldots, p \\
&\frac{\partial L}{\partial \mu_j} = h_j(\mathbf{x}) = 0 \; : \; j = 1, \ldots, q \\
&\lambda_j g_j = 0 \; : \; j = 1, \ldots, p \\
&\lambda_j \geq 0 \; : \; j = 1, \ldots, p
\end{aligned} \tag{3.15}$$

where L is Eqn. 3.3.

The KKT conditions specify a few points which are as follows:

1. The first line states that the linear combination of objective and constraint gradients vanishes.

2. A prerequisite of the KKT conditions is that the gradients of the constraints must be continuous (evident from second and third lines in Eqn. 3.15).

3. The last two lines in Eqn. 3.15 state that at optimum either the constraints are active or the constrains are inactive.

For more details please refer to [17].

4. SUPPORT VECTOR MACHINES

4.1. Introduction

Armed with the knowledge of optimization problems and concept of Lagrange multipliers, we now delve into the workings of support vector machines. Burges [13] provides a good introduction to SVMs and is our main reference. Interested readers should refer to Cristianini [18], Scolkopf [19] and Vapnik [20] for detailed references.

4.2. Support Vector Machines

4.2.1. Separable Case

Let us suppose that we are presented with a data set that is linearly separable. We assume that there are m examples of data in the format $\{\mathbf{x}_i, y_i\}$, s.t. $\mathbf{x}_i \in \mathbf{R}^n$; $i = 1, ..., m$, where $y_i \in \{-1, 1\}$ is the corresponding true label of $\mathbf{x}_i$. We also suppose there is an existence of a linear hyperplane in the n dimensional space that separates the positively labeled data from the negatively labeled data. Let this separating hyperplane be given by

$$\mathbf{w} \cdot \mathbf{x} + b = 0. \tag{4.1}$$

where, $\mathbf{w}$ is the normal vector $\perp$ to the hyperplane and $|b|/||\mathbf{w}||$ is the shortest perpendicular distance of the hyperplane to the origin. $||\mathbf{w}||$ is the Euclidean

norm of $\mathbf{w}$. The *margin* of a hyperplane is then defined as the minimum of the distance of the positively and negatively labeled examples, to the hyperplane. For the linear case, the SVM searches for the hyperplane with largest margin. We now have three conditions, based on the location of an example $\mathbf{x}_i$ w.r.t the hyperplane:

$$\mathbf{w} \cdot \mathbf{x}_i + b = 0 : \text{ example lying on the hyperplane.} \tag{4.2}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 : \text{ positively labeled example.} \tag{4.3}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 : \text{ negatively labeled example.} \tag{4.4}$$

Combining the equality and the two inequalities we have:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0 \tag{4.5}$$

Since the SVMs search for the largest margin, we now try to find a mathematical expression of the margin. Considering the examples that satisfy equality in Eqn. 4.3, the distance of the closest positive example can be expressed as $|1 - b|/||\mathbf{w}||$. Similarly, considering the negative examples that satisfy equality in Eqn. 4.4, the distance of the closest negative example can be expressed as $|-1 - b|/||\mathbf{w}||$. On summation of the two shortest distances, we get the margin of the hyperplane as $2/||\mathbf{w}||$. Since the labels are $\{-1, 1\}$, no example lies inside the hyperplanes representing the margin in this case. Taking into account that the SVM searches for the largest margin, we can say that it can be achieved by minimizing $||\mathbf{w}||^2$, subject to the constraints in Eqn. 4.5. Examples lying on the hyperplanes of the margins are termed *support vectors*, as their removal would change the margin and thus the solution. Figure 4.1 represents the conceptual points about separating hyperplanes.

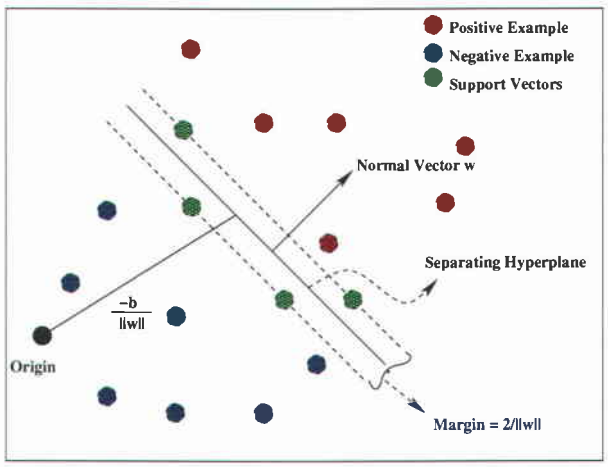

FIGURE 4.1. Linear hyperplane for separable data. Adapted from Burges (A Tutorial on Support Vector Machines)

4.2.2. Lagrangian Representation: Separable Case

Clearly, the previous paragraph shows that finding the margin is a problem of optimization as the goal is to minimize $||\mathbf{w}||^2$ subject to constraints in Eqn. 4.5. Employing the ideas of Chapter 3, the Lagrangian for the above problem, is:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{m} \alpha_i \tag{4.6}$$

where $\frac{1}{2}||\mathbf{w}||^2$ is the objective function, $\boldsymbol{\alpha}$ is the Lagrangian multiplier and the Eqn. 4.5 is the inequality constraint. Since the minimization of the objective function is required, we employ the ideas of the derivation of KKT conditions (Eqn. 3.15) to Eqn. 4.6. In short, we would require the $L(\mathbf{w}, b, \boldsymbol{\alpha})$ to be minimized w.r.t $\mathbf{w}$ and b and also require its derivative w.r.t all α_i's to vanish. Thus the KKT

conditions take the form:

$$\begin{aligned}
&\frac{\partial L}{\partial w_j} = w_j - \sum_i \alpha_i y_i x_{ij} = 0 && : \text{ j = 1,....,n} \\
&\frac{\partial L}{\partial b} = -\sum_{i=1}^{m} \alpha_i y_i = 0 && : \text{ i = 1,....,m} \\
&\frac{\partial L}{\partial \alpha_i} = -\sum_{i=1}^{m} y_i(\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{m} 1 = 0 && : \text{ i = 1,....,m} \\
&\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1) = 0 && : \text{ i = 1,....,m} \\
&\alpha_i \geq 0 && : \text{ i = 1,....,m}
\end{aligned} \tag{4.7}$$

Thus solving the SVMs is equivalent to solving the KKT conditions. While $\mathbf{w}$ is determined by the training set, b can be found by solving the penultimate equation in Eqn. 4.7 for which $\alpha_i \neq 0$. Also note that examples that have $\alpha_i \neq 0$ form the set of support vectors.

The dual problem for the same Lagrangian is:

$$D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{4.8}$$

Solving for Eqn. 4.8 requires maximization of D w.r.t α_i, subject to second line of Eqn. 4.7 and positivity of α_i, with the solution given by first line of Eqn. 4.7.

To classify or predict the label of a new example $\mathbf{x}_{new}$, the SVM has to evaluate $(\mathbf{x}_{new} \cdot \mathbf{w} + b)$ and check the sign of the evaluated value. A positive sign would lead to assignment of a $+1$ label and a negative sign to -1.

4.2.3. Nonseparable Case

For many classification problems, the data present is nonseparable. To extend the idea to nonseparable case, some amount of cost is added, which takes care of particular cases of examples. This is achieved by introducing slack in the constraints Eqn. 4.3 and Eqn. 4.4 ([13], [20]). The equations then becomes

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1 - \xi_i : \text{ positively labeled example.} \tag{4.9}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 + \xi_i : \text{ negatively labeled example.} \tag{4.10}$$

For an error to occur, the ξ_i value must exceed unity. To take care of the cost of errors, a penalty is introduced which changes the objective function from $||\mathbf{w}||^2/2$ to $||\mathbf{w}||^2/2 + C(\sum_i \xi_i)^k$. Thus $\sum_i \xi_i$ represents the upper bound on the training error. For quadratic problems, k can be 1 or 2.

4.2.4. Lagrangian Representation: Nonseparable Case

Since the formulation of the Lagrangian and its dual follow the same procedure, as mentioned in chapter 3, we only mention the equations. The Lagrangian for nonlinear nonseparable case is:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} - \sum_{i=1}^{m}\mu_i\xi_i \tag{4.11}$$

The corresponding KKT conditions for the Eqn. 4.11 are:

$$\begin{aligned}
&\frac{\partial L}{\partial w_j} = w_j - \textstyle\sum_i \alpha_i y_i x_{ij} = 0 && : \ j = 1, \ldots, n \\
&\frac{\partial L}{\partial b} = -\textstyle\sum_{i=1}^{m} \alpha_i y_i = 0 && : \ i = 1, \ldots, m \\
&\frac{\partial L}{\partial \alpha_i} = y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i = 0 && : \ i = 1, \ldots, m \\
&\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 && : \ i = 1, \ldots, m \\
&\alpha_i\{y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i\} = 0 && : \ i = 1, \ldots, m \\
&\mu_i\xi_i = 0 && : \ i = 1, \ldots, m \\
&\alpha_i \geq 0 && : \ i = 1, \ldots, m \\
&\xi_i \geq 0 && : \ i = 1, \ldots, m \\
&\mu_i \geq 0 && : \ i = 1, \ldots, m
\end{aligned} \tag{4.12}$$

The dual formulation $k = 1$ for the Lagrangian just discussed is:

$$D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{4.13}$$

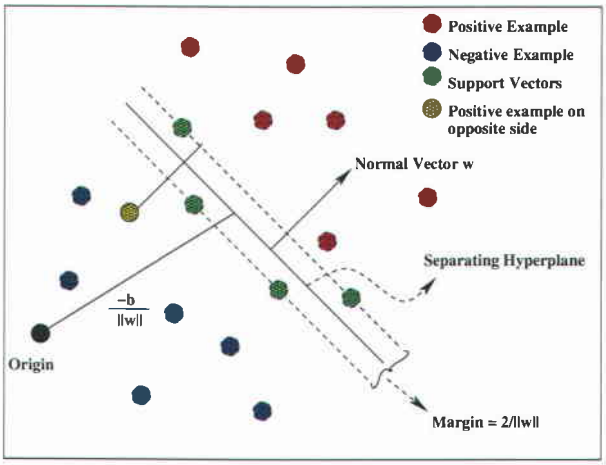

FIGURE 4.2. Linear hyperplane for nonseparable data. Adapted from Burges (A Tutorial on Support Vector Machines)

All the previous conditions remain same, except that the Lagrangian multiplier α_i now has a upper bound of value C. The solution for the dual is given by $\mathbf{w}= \sum_{i=1}^{N_s} \alpha_i y_i \mathbf{x}_i$. N_s is the number of support vectors. Figure 4.2 depicts the nonseparable case.

4.3. Kernels and Space Dimensionality Transformation

The above cases were for linear separating hyperplanes. In order to generalize for nonlinear cases, Boser et.al [21] employed the idea of Aizerman [22] as follows; Since the Dual in Eqn. 4.13 and its corresponding constraint equations employ the dot product of the examples, $\mathbf{x}_i \cdot \mathbf{x}_j$, it was proposed to map the data in a higher dimensional space using a function ϕ s.t. the algorithm would depend only on dot products in the higher space. Next, the existence of a function called *kernel*, dependent on $\mathbf{x}_i$ and $\mathbf{x}_j$, was assumed s.t. the value reported by the kernel

was equal to the value resulting from the dot product in the higher space. A mathematical representation of the above concept is

$$\phi : \mathbf{R}^n \mapsto H \tag{4.14}$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \tag{4.15}$$

where H is a higher dimensional space.

This technique drastically reduces the amount of work required while dealing with nonlinear separating hyperplanes, concerning search for appropriate ϕ. Instead, one only works with $K(\mathbf{x}_i, \mathbf{x}_j)$, in place of $\mathbf{x}_i \cdot \mathbf{x}_j$. For classification purpose, where the sign of the function $(\mathbf{x}_{new} \cdot \mathbf{w} + b)$ is evaluated, the formulation employing kernels become:

$$\begin{aligned} f(\mathbf{x}_{new}) &= (\mathbf{w} \cdot \mathbf{x}_{new} + b) \\ &= \textstyle\sum_{i=1}^{N_s} \alpha_i y_i \phi(\mathbf{s}_i) \cdot \phi(\mathbf{x}_{new}) + b \\ &= \textstyle\sum_{i=1}^{N_s} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}_{new}) + b \end{aligned} \tag{4.16}$$

where $\mathbf{s}_i$ are the support vectors.

4.4. Edit Distance Based Kernels

4.4.1. Problem of Abundant Features

We have been discussing the workings of SVMs so far, but we haven't talked much about how the representation of the data could affect the training and the testing phases. We considered the data to be present in the format:

$\{\mathbf{x}_i, y_i\}$: (example, label) pair

$\mathbf{x}_i \in \mathbf{R}^n$: i = 1,....,m

$y_i \in \{-1, 1\}$: corresponding to true label of $\mathbf{x}_i$

Based on the discussion in Chapter 2, concerning the sequential data retrieved from the leaf images in a CBIR system, it is found that a single leaf image can have very large number of features (here angles of a leaf boundary). Since the main idea is to build a fast, accurate and highly automated CBIR system, the presence of huge amount of features just for a single example, can lead to immense loss in time, while training and testing. In order to reduce the loss in time, an efficient way of feature representation is required. The idea is to retrieve maximum information, yet keeping the number of features to the bare minimum.

To get around with the problem , we employed the technique of edit distances, as mentioned in Section 2.3.3 (Chapter 2). Edit distances, specify the cost of matching a sequence X against a sequence Y, via execution of DTW. In a way, they specify the information between two sequences by a single value. What we are achieving is a drastic reduction in the number of features from say n to 1, while retaining the same information.

4.4.2. Transforming Features to Higher Space

Recently, edit distances and their modified versions have been employed in pattern recognition algorithms such as SVMs and k-NNs. Jaun and Vidal [11] proposed the use of normalized edit distance for *k-AESA* to classify human banded chromosomes. Bahlmann et al. [12] proposed the idea of combining the concept of DTW and SVMs to build a new SVM kernel for online hand writing recognition. Our independent research work, though based on similar ideas, shows how edit distances simplify the feature representation of sequential data and prove the dominance of SVMs over the Nearest Neighbors.

After the edit distances have been computed, the problem of constructing kernels arises. We build a *Gram Matrix*, $\mathbf{G}$, the elements of which are *kernel* functions, $K(\cdot,\cdot)$, of i and j (Please refer back to Section 2.3.3, Chapter 2 for meaning of i and j). We tried two different kernels.

$$\begin{aligned} K(i \cdot j) &= \frac{1}{1+\frac{Avg(DP(i,j),DP(j,i))}{\lambda}} \quad : \text{Inverse Kernel} \\ K(i \cdot j) &= e^{-\frac{Avg(DP(i,j),DP(j,i))^2}{\lambda^2}} \quad : \text{Gaussian Kernel} \end{aligned} \tag{4.17}$$

$Avg(a,b) = \frac{a+b}{2}$, i and j are plant leaves, and λ is a tuning parameter. The *Gram Matrix* takes the form:

$$\mathbf{G} = \begin{bmatrix} 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 0 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \end{bmatrix}_{nXn} \tag{4.18}$$

Figure 4.3 and visualizes how the kernels represent the similarity between any two leaves. All leaves show maximum similarity when matched against themselves. Leaves with greater dissimilarity have lower kernel values. As presented in figure 4.3 the inverse kernel function is more steep and restrictive in representing the similarity measure as compared to its counterpart, the Gaussian kernel. Figure 4.4 provides the same information with 3D perception in mind. The symmetry of the Gram matrix is seen in figure 4.5. The symmetry is a property that is required for designing any kernel.

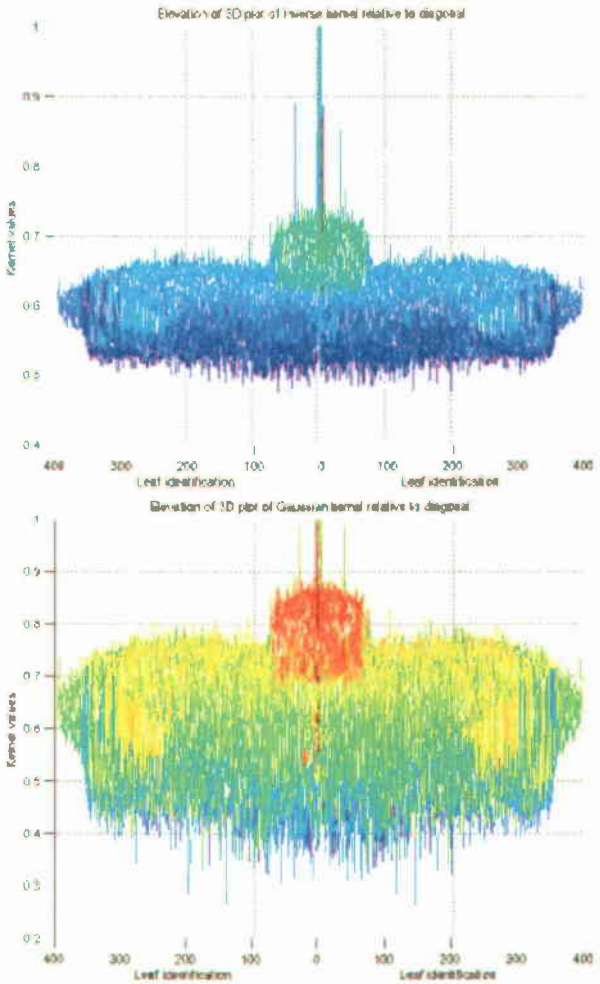

FIGURE 4.3. 2D view of inverse and Gaussian kernels, respectively.

4.4.3. Mercer Conditions and Mercer Kernels

The kernel functions represent the $1D$ representations of i^{th} and j^{th} leaves as some function of $DP(i,j)$ in a higher dimensional space. Since some of the eigen-values were negative, the **G** matrix, though being symmetric was not conditionally positive. In the literature of SVMs, a kernel that does not satisfy the *Mercer Condition*, i.e. the conditional positivity of the **G** matrix, is not a *Mercer*

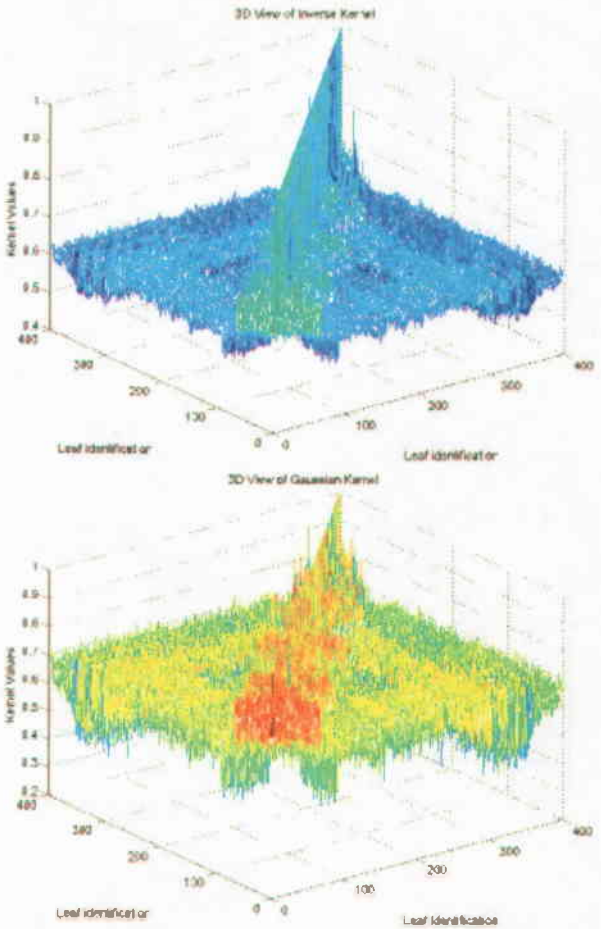

FIGURE 4.4. 3D view of inverse and Gaussian kernels, respectively.

Kernel. One can then question our results obtained by using above kernels! In fact, Burges [13] states that there can be presence of data such that the Hessian is indefinite and the dual objective function could become arbitrarily large. Also, one might find a training set which results in a positive semidefinite Hessian, in which case the training would converge perfectly well. Burges indicates that in such a case it would be hard to make a geometrical interpretation of the trans-

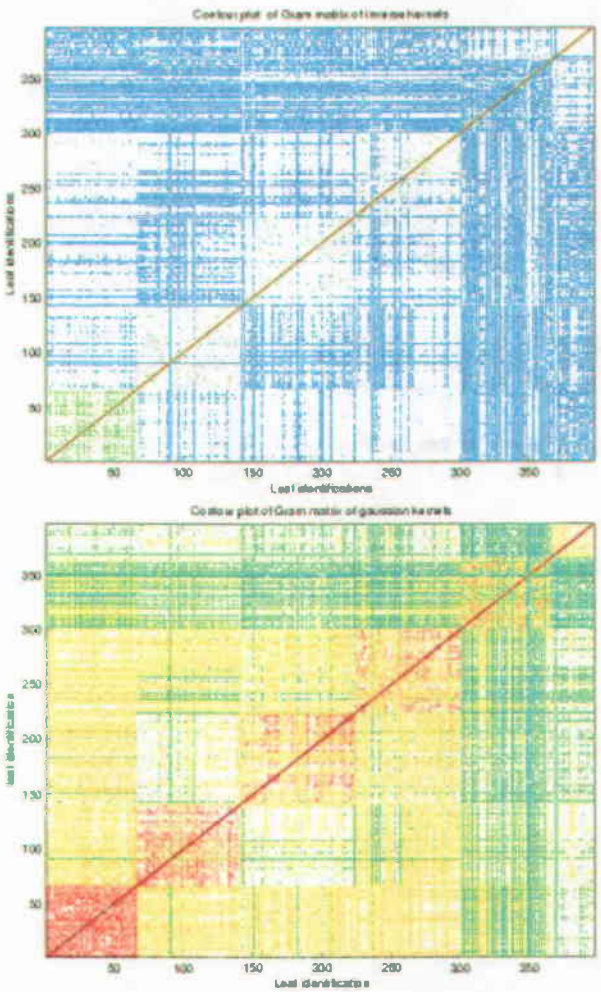

FIGURE 4.5. Contour plot of inverse and Gaussian kernels, respectively.

formation. This is exactly what is happening in our case. Though the kernels are not conditionally positive, the SVMs train well and converge to give good results.

4.5. One-Versus-Rest SVMs and Multiclass SVMs

Until now, we were only considering binary classification. Since the number of species of leaves is greater than two, we will now introduce idea of Multiclass SVMs using One-Versus-Rest SVMs.

4.5.1. One-Versus-Rest Support Vector Machine

The main idea behind One-Versus-Rest SVM (1vrSVM) is the consideration of examples of a single class out of N classes as having a positive label and the remaining N-1 classes as having negative label. This situation is considered for each and every class in turn, thus generating N binary training set. We employ One-versus-rest Support Vector Machines using SVM^{light} as provided by Joachims [15], utilizing edit distances as kernels. The 1vrSVMs train on the binary class $\{+1, -1\}$. The dual problem is:

$$Maximize : D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i \cdot x_j) \tag{4.19}$$

$$subject\ to : 0 \leq \alpha_i \leq C \tag{4.20}$$

$$: \sum_i \alpha_i y_i = 0 \tag{4.21}$$

D is the Lagrangian dual. α_i and C are the Lagrangian multiplier and error penalty parameter, respectively. Whenever a kernel function is called with particular indices of leaves, the SVM retrieves the corresponding value from the above generated *Gram Matrix*. While training, as is obvious, we need to tune the parameters which are being considered to achieve optimal generalization performance. Experiments for tuning of parameters will be elucidated later.

4.5.2. Multiclass Support Vector Machine

Once the training is done on all of the 1vrSVMs, prediction on test set is carried out using the support vectors generated from N 1vrSVMs. To classify a test example into one of the N classes, the class receiving the maximum weight age through prediction value, is the label of the test example. In mathematical formulation the multiclass SVM is:

$$f(x_{new}) = \underset{j \in N}{argmax} \{\sum_{i=1}^{N_s} \alpha_{j_i} y_{j_i} K(s_{j_i}, x_{new}) + b\} \tag{4.22}$$

$f(x_{new})$ is the predicted class. N, N_s, s_i and y_i are number of classes, number of support vectors in each of the 1vrSVM, support vector and the class label for s_i, respectively.

Leaf Shape Recognition via Support Vector Machines with Edit Distance Kernels

5. NEAREST NEIGHBOR ALGORITHMS

5.1. Introduction

Nearest Neighbor algorithms are the simplest pattern recognition algorithms, which classify an example based on the dissimilarity measure. In this chapter we present two simple versions of nearest Neighbor algorithms. In our case, the kernel values, now represent a dissimilarity measure between two leaves. As one can observe from Eqn. 4.18, the diagonal elements of the *Gram matrix* are zero. This is obvious, as the cost of matching two identical leaves has to be zero. So in terms of dissimilarity measure, one can say that the dissimilarity is nil, between two identical leaves. For similar but not identical leaves of same species, the dissimilarity measure would be greater than zero. For dissimilar leaves of different species, the dissimilarity measure should be highest in comparison to the previous two cases.

5.2. k-Nearest Neighbor Algorithm

With the kernel values as distance metrics, the k-NN algorithm tries to predict the class of a test example by taking a vote among the k-nearest neighbors. A mathematical formulation of the previous statement is as follows:

$$vote[i] = |\{j \mid j \in k \text{ Nearest Neighbors, class of } j \equiv \text{ class } i\}| \tag{5.1}$$

$$f(new) = \max_{i \in N} \{vote[i]\} \text{ for k Nearest Neighbors} \tag{5.2}$$

where new is the test example, $f(new)$ is the predicted class, and k is the number of nearest neighbors.

5.3. Weighted k-Nearest Neighbor Algorithm

The weighted version is similar to the normal k-NN, except for the fact that prediction decision is based on weighted vote. We employ a modification of the Nadaraya Watson Weighted k-Nearest Neighbor algorithm as described in [14]. The idea behind the employing the weighted version is to give higher weight to the nearest neighbors. The mathematical formulation of the weighted version is

$$f(new) = \frac{\sum_k K(k, new) * \text{class of } k}{\sum_k K(k, new)} \text{ for Wtd.k-Nearest Neighbor} \qquad (5.3)$$

where new is the test example, $f(new)$ is the predicted class, and k is the number of nearest neighbors. Here we have tried to apply the methods of Nadaraya Watson Weighted k-Nearest Neighbor employing the Gaussian and the inverse kernel function.

Leaf Shape Recognition via Support Vector Machines with Edit Distance Kernels

6. EXPERIMENTAL PROCEDURES AND RESULT

6.1. Introduction

We now possess enough knowledge of pattern recognition algorithms to apply these for classification of leaves from different species. This chapter provides information about the data set obtained from [16] and discusses the procedures carried out to tune the parameters, while training of algorithms. Finally, we present the results obtained on test data set.

6.2. Dataset

Plant leaves of six different species were collected and classified by Gandhi. The species under consideration are shown in table 6.1. We built *Gram Matrices* of 397 x 397 dimension with a total of 397 leaf patterns.

Class	Species Name	Number of leaves per species
1	*Acer Circinatum*	66
2	*Acer Glabrum*	75
3	*Quercus Garryana*	84
4	*Acer Macrophyllum*	75
5	*Acer Negundo*	66
6	*Quercus Kelloggi*	31

TABLE 6.1. Class Label and Species Name

6.3. Leave One Out Cross Validation

6.3.1. Training

In Leave One Out Cross Validation (LOO henceforth), out of n examples, training is done on $n-1$ examples and validation, i.e., prediction of the label, is done on the remaining one example. It is usually employed when the data is present in very small amount.

Procedure

1. Nine pairs of training and test sets are constructed with an increasing number of samples in the training set and a decreasing number of samples in the test set.

2. Carrying out LOO within the training set, best values of λ and k are found for k-NN and Wtd.k-NN.

3. Carrying out LOO within the training set, best values of λ and C are found for 1vrSVMs.

4. Finally, classification is done for each test set, for all the three algorithms for the chosen values of parameters.

For k-NN and Wtd. k-NN, the optimal value of k is found to be 1. The λ value for the former algorithm did not have any effect on the validation result as kernel values varied proportionally. For the weighted version of nearest neighbor, λ varied between 200 and 350, implying that the average cost of matching was inside this range. For 1vrSVMs, the choosen C values were in the range from 1 to 10. Again the λ values vary in the same range as before. Figure 6.1 shows

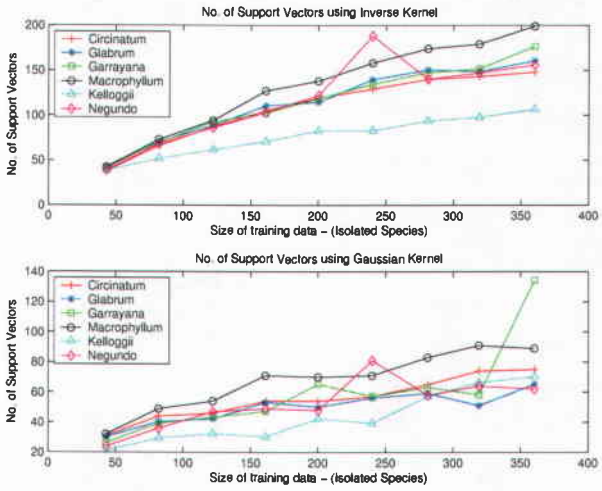

FIGURE 6.1. The top graph shows number of support vectors using inverse kernel, while the bottom graph shows the number of support vectors using Gaussian kernel ($\lambda = 350$). LOO is employed as the CV technique.

the number of support vectors for each of the 1vrSVMs as the size of the training data increases.

6.3.2. Testing

We now present the percent accuracy of k-NN, Wtd. k-NN and Multiclass SVM, for inverse kernel and Gaussian kernel functions, employing edit distances as a means of feature representation. In Figure 6.2, the upper graph shows the percentage accuracy curves of the three algorithms employing inverse kernel and the lower graph presents percentage accuracy curves using Gaussian kernel, on test data.

A quick view of the graphs show the dominance of SVMs over the other two algorithms. The weighted version of the nearest neighbors performs badly

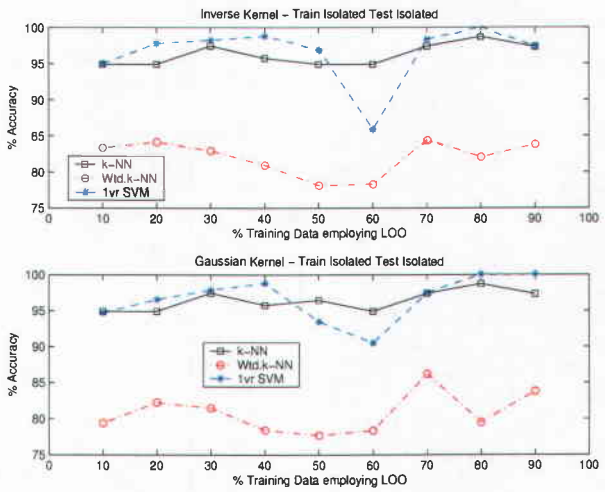

FIGURE 6.2. Percentage accuracy plot for test data, as the size of the training data increases. LOO is employed as the CV technique.

with respect to the other two algorithms. The performance of Multiclass SVMs show high variance as compared to the k-NN. High variance in test results can be attributed to using LOO to set the parameter λ. Though the parameter does not affect the nearest neighbor as the kernel values change proportionally, the same may not be true for SVMs as change in kernel values may lead to selection of different support vectors. As the size of the training data increases, the newly added training data play a significant role in validation of the remaining one example. With very minimal test data (about 10 to 30), the SVM and k-NN show almost 100 accuracy. Concerning the two kernels, the inverse kernel smoothes out the curves as compared to the Gaussian kernel.

6.4. K-Fold Cross Validation

6.4.1. Training

In a simple K-Fold Cross Validation (K-Fold henceforth), out of n examples, K folds of data approximately equal size are constructed. For each of the i^{th} fold in K folds, training is done on the remaining $K-1$ folds excluding the fold under consideration. Validation, i.e., prediction of the labels, is done on the left out fold, and parameters are tuned. It is usually employed when the data is present in small amount.

Procedure

1. Six K-Fold Cross Validation are considered.
2. For each of the K-Fold Cross Validation, K folds of approximately equal size are constructed.
3. For each of the i^{th} fold in K folds, training is done on the remaining $K-1$ folds and validation on the fold under consideration (i^{th} fold).
4. Best values of λ and k are found for k-NN and Wtd.k-NN, based on prediction accuracy on validation set. Similarly, best values of C and λ are found for Multiclass SVMs, based on prediction accuracy on validation set. This provides K different best values for the parameters.
5. Finally, average percentage accuracy is reported for all three algorithms, by taking into consideration, the prediction accuracy on K validation sets.

Almost same values of λ and C are found for the SVMs across the folds. For the two versions of nearest neighbors, we choose $k = 1$. λ values for nearest neighbors

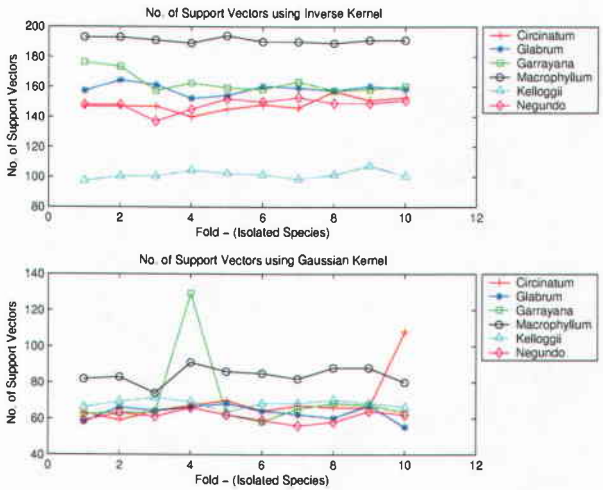

FIGURE 6.3. Top graph shows number of support vectors using inverse kernel, while the bottom graph shows the number of support vectors using Gaussian kernel ($\lambda = 350$). 10-fold CV is employed as the CV technique.

are found to be almost same as before. Figure 6.3 shows the number of support vectors for each of the 1vrSVMs for each of the 10 training folds in 10-fold CV.

6.4.2. Testing

We now consider the performance of the three algorithms. Figure 6.4 presents the percentage accuracy for the six K-Fold Cross Validation employing the inverse kernel and the Gaussian kernel function. Note that for nearest neighbors, we set $k = 1$, while λ's were cross validated. K-Fold certainly has an advantage over the LOO technique, as would be evident from a comparison of Figure 6.4 and Figure 6.2. Presence of approximately equally sized folds helps to smooth out any bad affect by bad data in any of the folds. There is a marked difference in the performance of the three algorithms. Though the behavior is

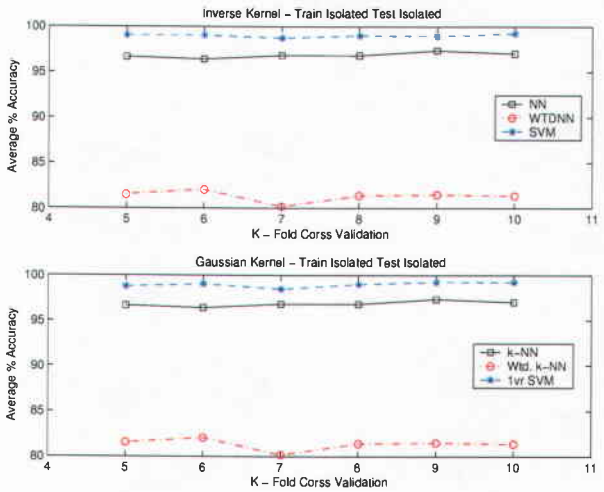

FIGURE 6.4. Percentage accuracy plot for test data, as the size of the training data increases. K-Fold is employed as the CV technique.

the same concerning the dominance of Multiclass SVMs over the two versions of nearest neighbors, all three show higher percentage of accuracy. Employing LOO, SVMs gave 95 to 96 percent accuracy on average as compared to the present average percentage accuracy of 98 to 99 percent. The weighted version of nearest neighbors now give an accuracy of more than 82 percent approximately, as compared to 80 percent accuracy on average using LOO. For k-NNs also there is an increase in the average accuracy by 2 percentage points.

It is also evident from Figure 6.4, that Multiclass SVMs now show a marked difference in performance compared to k-NN, by an margin of 2 to 3 percentage points. Considering the accuracy for CBIR systems based on the above pattern recognition algorithms and the employment of edit distances as features, we can rightly say that edit distances represent maximum information and yet are concise in nature. Secondly, the use of parametric edit distance based kernel, provide a powerful tool for simplifying the job of classification through Support Vector

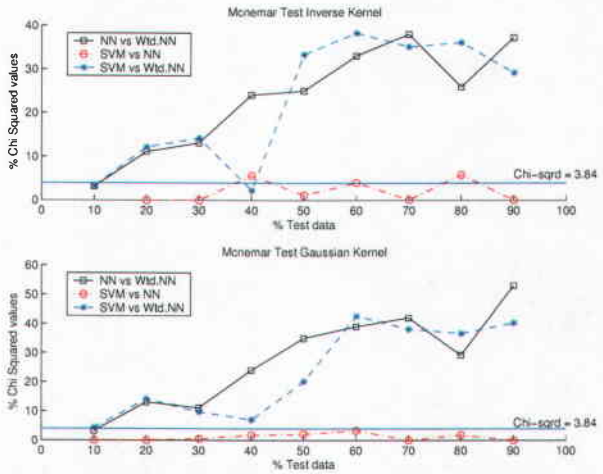

FIGURE 6.5. McNemar's Test using LOO technique

Machines. It can also be employed in Nearest Neighbors as the dissimilarity measure.

6.5. McNemar's Test

In the previous section we stated that the SVMs show a marked difference in preformance compared to k-NNs while executing K-Fold Cross Validation. To prove the veracity of the stated point, we conducted McNemar's test for both the techniques (LOO and K-Fold). The results of the McNemar's test are shown in Figure 6.5 and Figure 6.6. In both the figures, the top graph specifies the use of inverse kernel function and the bottom graph specifies the use of the Gausian kernel function. Figure 6.5 does show that we did not find much statistical difference between the Multiclass SVMs and the k-NNs. This is true, as the performance of the SVMs and k-NNs is almost same while employing LOO. Regarding SVMs and the weighted version of k nearest neighbors and k nearest neighbors and the

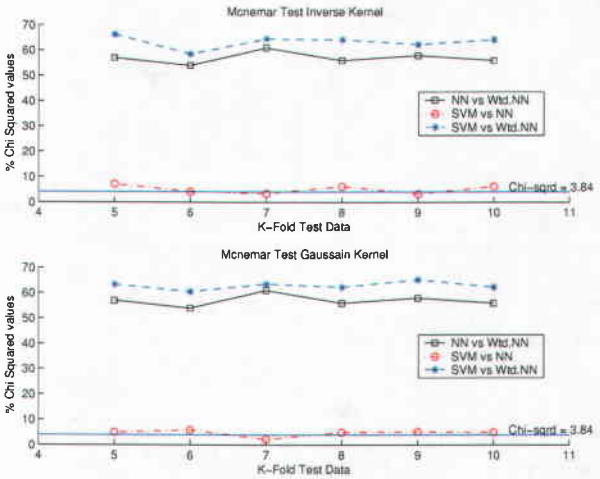

FIGURE 6.6. McNemar's Test using K-Fold technique

weighted version of k nearest neighbors, we do find statistical differences. The horizontal line in Figure 6.5 and Figure 6.6 specifies $\chi^2_{(1,0.95)} = 3.841459$.

In the case of the K-Fold Cross Validation technique, we do find statistical difference between SVMs and the k nearest neighbors, though they are just barely above the threshold. This is depicted in Figure 6.6. Some of the image samples are shown in Figure 6.7 and Figure 6.8.

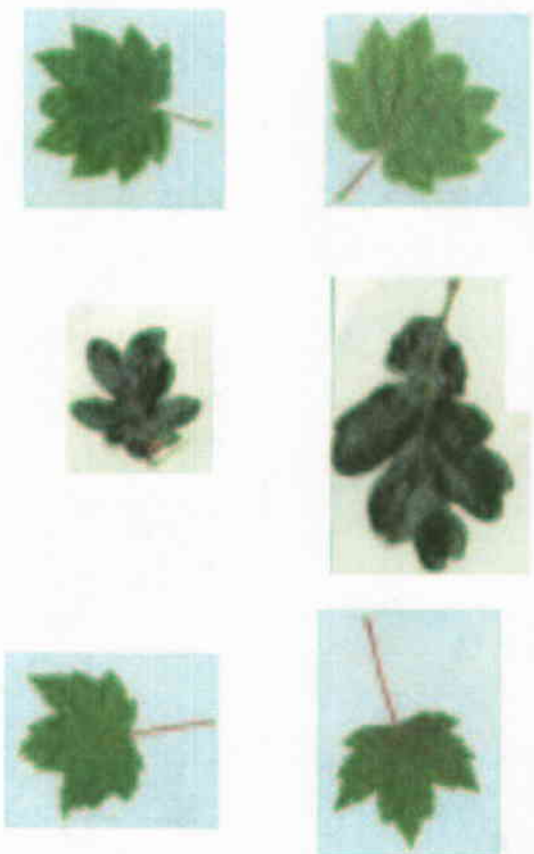

FIGURE 6.7. Leaf samples of circinatum, garryana and glabrum species from top to bottom.

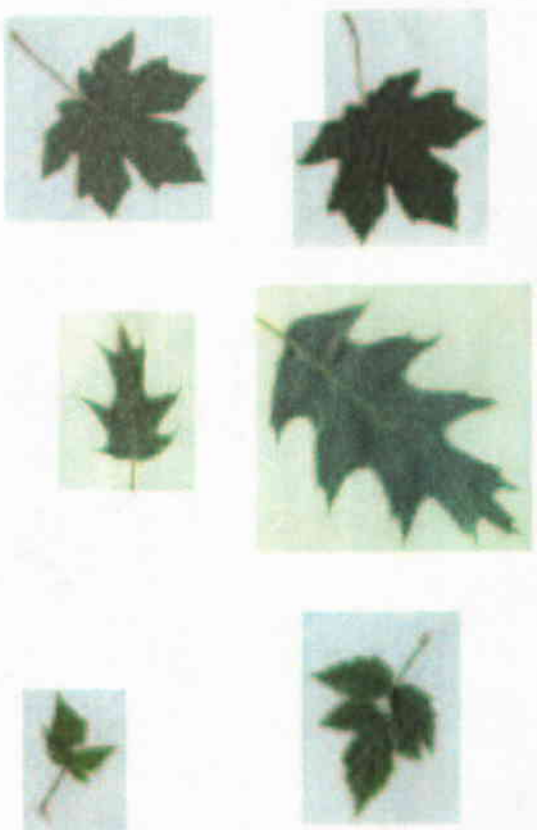

FIGURE 6.8. Leaf samples of macrophyllum, kelloggi and negundo species from top to bottom.

7. CONCLUSION

7.1. Conclusion

We conclude here with the following remarks:

- The use of a parametric edit distance based kernel provides a powerful tool for simplifing the job of classification through support vector machines.
- Edit distance based kernels can also be employed in Nearest Neighbors as a dissimilarity measure, though SVMs prove to be the dominating algorithm.
- The inverse kernel function proved to be slightly better than the Gaussian kernel function.
- Highly accurate CBIR systems can be developed that are completely automated.

BIBLIOGRAPHY

[1] Glantz H.T., On Recognition of Information With a Digital Computer. *J. ACM*, Vol.4, Aug.(1957), 178–188.

[2] Alberga C.N., String Similarity and Misspellings. *Comm. ACM*, Vol.10, No.5, May(1967), 302–313.

[3] Morgan H.L., Spelling Correction in Systems Programs. *Comm. ACM*, Vol.13, No.2, Feb.(1970), 90–94.

[4] Wagner R.A., Fischer M.J., The String-to-String Correction Problem. *J. ACM*, Vol.21, No.1, Jan.(1974), 168–173.

[5] Sellers P.H., The Theory and Computation of Evolutionary Distances: Pattern Recognition. *Journal of Algorithms*, Vol.1, (1980), 359–373.

[6] Marzel A., Vidal E., Computation of Normalized Edit Distances and Applications. *IEEE Trans.Pattern Analysis and Machine Intelligence*, Vol.15, No.9, Sept.(1993), 926–932.

[7] Bunke H., Bühler U., Applications of Approximate String Matching to 2D Shape Recognition. *Pattern Recognition*, Vol.26, No.12, (1993), 1797–1812.

[8] Bunke H., Csirik J., Parametric String Edit Distance and its Application to Pattern Recognition. *IEEE Trans. Systems,Man and Cybernetics*, Vol.25, No.1, (1995), 202–206.

[9] Petrakis E.G.M., Diplaros A., Milios E., Matching and Retrieval of Distorted and Occluded Shapes using Dynamic Programming. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.24, No.11, Nov.(2002), 1501–1516.

[10] Durbin R., Eddy S., Korgh A., Mitchison G., *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, (1998).

[11] Juan A., Vidal E., On Use of Normalized Distances and an Efficient k-NN Search Technique (k-AESA) for Fast and Accurate String Classification. *ICPR*, Vol.2, (2002), 680–683.

[12] Bahlmann C., Haasdonk B., Burkhardt H., On-line Handwriting Recognition with Support Vector Machines - A Kernel Approach. *IWFHR*, (2002), 49–54.

[13] Burges C.J.C., A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, Vol.2, (1998), 121–167.

[14] Hastie T., Tibshirani R., Freidman J., *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer series in Statistics, (2001).

[15] Joachims T., Making Large-Scale SVM Learning Practical. *Advances in Kernel Methods - Support Vector Learning*, Schölkopf B., Burges C.J.C., Smola A.(ed.). MIT Press, (1999).

[16] Oregon State University Herbarium, http://oregonstate.edu/dept/botany/herbarium.

[17] Ing. Kai-Uwe Bletzinger, Structural Optimization, Lecture Notes - Chapter 4 *Basic Mathematics.* TU Munich, Summer (2002).

[18] N. Cristianini, J. Shawe-Taylor, *An Introduction To Support vector Machines (and other kernel-based learning methods)*, Cambridge University Press, (2000).

[19] Bernhard Schlkopf, Alex Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*, MIT Press, Cambridge, MA, (2002).

[20] Valdimir N Vapnik, *Statistical Learning Theory*, John Wiley and Sons, (1998)

[21] B. E. Boser, I. M. Guyon, V. Vapnik, A Training Algorithm for Optimal Marginal Classifiers, *Fifth Annual Workshop on Computational Learning Theory, ACM*, (1992).

[22] A. Aizerman, E.M. Braverman, L. Rozonoer, Theoretical foundations of the Potential Function Method in Pattern Recognition Learning, *Automations and Remote Control* 25, 821-837 (1964).