Support Vector Networks

Vinay Kaundinya Ronur Prakash

Abstract Traditionally pattern recognition is a field that stems out of engineering and has eventually found its way into computer science especially through the appication of machine learning algorithms. Typically pattern recognition problem can be solved as a classification problem. From a single perceptron to today's deep learning algorithm, there have been many state-of-the-art solutions developed to tackle such classification problems. In this report, we describe the working of one such learning machines called *Support Vector Networks*.
Support Vector Networks, or as otherwise called *Support Vector Machines*(SVM) solves a two-group classification problem by constructing a hyperplane that optimally separates the two classes. SVM maps all the input vectors into a high dimensional feature space. It then finds a hyperplane as a decision surface in an N-dimensional feature space that distinctly classifies the data points into two classes. Constructing an hyperplane separating the data that generalizes well over high-dimensional feature spaces and to computationally be able to treat such feature spaces stand as two main challenges for SVM. An *Optimal Hyperplane* is constructed which separates data linearly without any errors. In the paper *Support-Vector Networks* [1] the idea of constructing optimal hyperplanes have been extended to training data with error, with the concept of *Hyperplanes with Soft Margin*. In this report we look at construction of convolution of dot-products in a higher dimensional space. Experiments were performed on *US Postal Services Database* and *NIST Database* and the results were then compared with classical machine learning algorithms.

Vinay Kaundinya
Paderborn University, e-mail: vinaykaundinya95@gmail.com

1 Introduction

Almost everything coming out of the technology world these days is a consequence of Human beings hard-wired to recognize and categorize patterns around them. This has led to inventions and research on artificial systems to read poorly handwritten text, parse and understand human speech, recognize faces in a crowd, classify people as old or young, etc, since time immemorial. Today, pattern recognition plays a crucial role in diagnosing diseases, inspiring new ways to safeguard data, and even discovering new planets.

Machine Learning as a subfield of Artificial Intelligence, along with statistics is concerned with the development of techniques and methods which enable the machine to learn and perform such tasks and activities. Some of the early solutions for pattern recognition and classification involved the use of single neuron perceptron or a basic unit of what is known as today's neural network model. Here we discuss and demonstrate one of the benchmark machine learning algorithms that was experimented in the field of Optical Character Recognition for classifying non linearly separated data, which led to many more advancements in the field of Machine learning in general and two group classification in specific. Support Vector Machines(SVM) or also known as Support Vector Networks are among one of the most popular and talked about machine learning algorithms, as powerful and universal as neural networks.

In case of a classic two-category classification problem, any linear classifier separates the space, with a hyperplane, into two regions, each of which is also called a class. To define Hyperplanes, they are nothing but decision boundaries that help classify the given data points. Data points fall on either side of the hyperplane to form two different classes. The dimension of a hyperplane always depends upon the dimension of the input data points or simply the number of input features. If the number of features is 2, then the hyperplane is just a line. If it is 3, then the hyperplane becomes a $2d$ plane and so on as in Fig. 1.

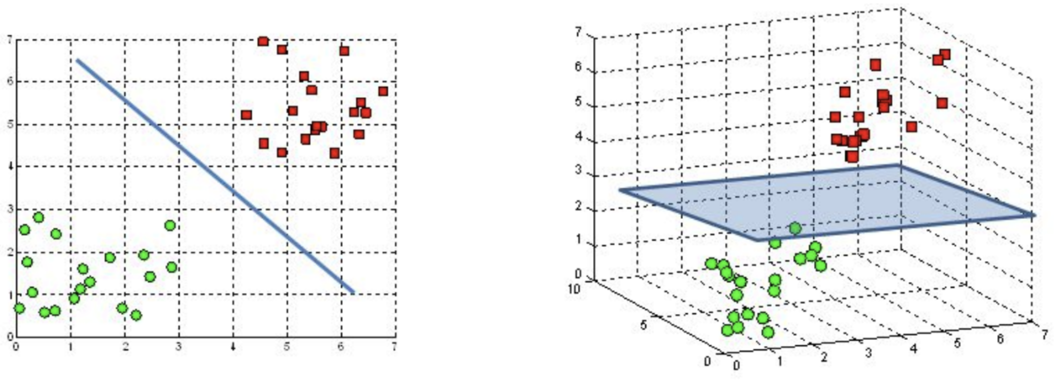

Fig. 1 An example of $1d$ and $2d$ Hyperplanes, for 2 and 3 input features respectively.

We can also see that a number of hyperplanes can be drawn to separate the classes into two. Intuitively we can say that, the plane that is furthest from the data points of both classes is the most optimal hyperplane. Support vectors are data points that are closer to the hyperplane. Support vectors influence the position and orientation of the hyperplane. Support vectors decides the margin of the classifier. Deleting the support vectors will change the position of the hyperplane.

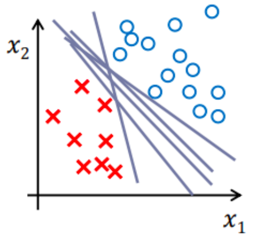

Fig. 2 An example of different possible hyperplanes.

Thus a support vector network (or SVM) aims to map the input space into a high dimensional feature space through some non linear mapping and constructs an optimal separating hyperplane in the feature space. This linear decision surface in feature space, ensures high generalization ability of the model and also corresponds to a non linear decision surfaces in input space.

In this report we understand the functioning of an SVM in optimally finding an hyperplane, to classify non linearly separated input data points in recognizing a character. Further in this report we describe the experiments conducted for digit recognition using the proposed SVM approach on real US Postal Service Database and NIST Database.

2 Related work

This section gives an overview of work done in the field of pattern recognition, as a classification problem, until the SVM approach was conceptualized.

There has been extensive research and development in the field of pattern recognition and image processing since 1930s. RA Fischer in 1936, proposed a model that considers two normally distributed sets of values, each of which has n dimensional(n features) vectors(x). Mean vectors for the two normally distributed populations were then computed as $m1$ and $m2$. We know that covariance is a measure of how much two values vary (for ex: the height of a person and the weight of a person in a population). The author then shows that with covariance matrices $\sum_1$ and $\sum_2$ for the two sets of values, the optimal solution is a quadratic function as in 1. We know that in case of a linear function, we need to determine only n parameters, but in case of a quadratic function it can be as much as $n(n+3)/2$ parameters.

$$F_{quad}(x) = sign\left[\frac{1}{2}(x-m_1)^T \sum_1^{-1}(x-m_1) - \frac{1}{2}(x-m_2)^T \sum_2^{-1}(x-m_2) + \ln \frac{|\sum_2|}{|\sum_1|}\right] \tag{1}$$

He also showed that, when the covariance matrices of the two sets of values are equal, i.e $\sum_1 = \sum_2$, then equation 1 results in a linear equation. Since the complexity is higher in computing roots of a quadratic equation, RA Fischer suggests to use the linear function even when $\sum_1 \neq \sum_2$, by calculating $\sum$ as shown below.

$$F_{lin}(x) = sign\left[(m_1-m_2)^T \sum^{-1}(x) - \frac{1}{2}(m_1^T \sum^{-1} m_1 - m_2^T \sum^{-1} m_2)\right] \tag{2}$$

$$\sum = \tau \sum_1 + (1-\tau) \sum_2 \tag{3}$$

Here τ is a constant whose value was later found in 1960s, which approximately took a value $0 < \tau < 1$. Author suggests to use Linear decision function 2 even in case of the two datasets are not normally distributed, showing that construction of a decision function becomes primal in pattern recognition problems.

Although there has been a lot of attention given to neural networks and deep learning in the last decade, its history dates back to 1940s when the first artificial neural model was developed. Since then there have been many approaches and Rosenblatt's perceptrons is one of them. According to Rosenblatt's method, a perceptron is combination of connected units called neurons. Each of these units then find a optimal hyperplane individually and the approach as a whole implements a piecewise linear hyperplane. Here the weights(α_i) are associated with each layer(ith layer) including the output layer. The output layer is made adaptive and all the other weights have a fixed value. Then the linear decision function is constructed by transforming the input vectors (x) into a high dimensional feature space(z). Even in this approach, linear decision function was associated with hyperplanes in a feature space and thus we know the importance of construction of a decision function or hyperplanes in any classification approach.

$$F_{lin}(x) = sign\left(\sum_i \alpha_i z_i(x) \right) \tag{4}$$

Developing on the Rosenblatt's approach(where only the output layer weight was adaptive), many more approaches were developed where weights of all layers were made adaptive. This adaptive nature of weights for each layer was introduced after back-propagation model was discovered around 1980s.

3 Problems Tackled

An important function of an SVM is not only to construct a hyperplane in feature space, but to construct one that generalizes well. As the dimensionality of the feature space increases(it can go as high as a billion dimensions), finding an optimal hyperplane that separates the two classes distinctly becomes more difficult. So finding an optimal hyperplane that generalizes well over the training data poses as the first problem, also known as the *conceptual* problem. At the same time, computationally treating such huge dimensional feature space pose as another problem, also known as the *technical* problem.

We know that the conceptual problem is to define an optimal hyperplane. In this context an optimal hyperplane is defined as an linear decision function, that maximizes the margin between the classes and to achieve such an hyperplane minimal number of support vectors are considered as shown in Fig. 3.

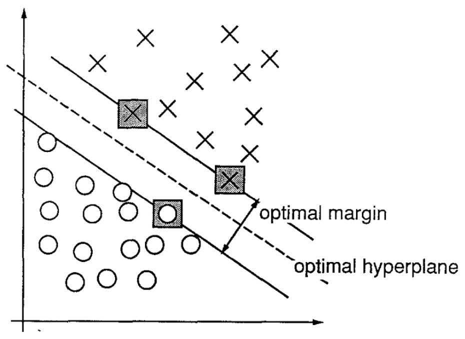

Fig. 3 An optimal hyperplane with minimal support vectors and maximal margin

This problem was solved by *Vapnik* and for an optimal hyperplane the expected probability of committing an error(E[Pr(Er)]) is shown to be always less than or equal to the ratio of expected number of support vectors(E[Num of SV]) and number of training vectors(Num of TV). From this ratio we can say that the generalization ability of SVM increases even for a high dimensional space, if number of support vectors used in constructing an optimal hyperplane is small. Here an optimal linear decision function is defined in feature space(z)(5).

$$w_0 \cdot z + b_0 = 0 \tag{5}$$

Where w_0 is the weight applied for the optimal hyperplane in the feature space(z is a vector in feature space). b_0 is a bias constant that is applied which has influence on the shape of the optimal hyperplane being constructed and $\cdot$ denotes dot product. The optimal weight(w_0) can then be a combination of all the support vectors.

$$w_0 = \sum_{i=1}^{supportvectors} \alpha_i z_i \tag{6}$$

Replacing equation 6 in 5 we get the equation for a linear decision function $I(z)$.

$$I(z) = sign\left(\sum_{i=1}^{supportvectors} \alpha_i z_i \cdot z + b_0 \right) \tag{7}$$

Here $z_i \cdot z$ represents the dot product of the support vector z_i and the vector z in feature space. After solving the problem of constructing optimal hyperplanes that generalize well over the input training data, it was then shown that the technical problem of dealing with high dimensional input can be solved by interchanging the order in which transformation of the vectors and dot products between the vectors happen. Here as solution, the vector in input space is first compared with support vectors and then a non-linear transformation of the result of the comparison is made. Comparison between vectors in input space can be done using similarity measures, or by taking a dot product etc.

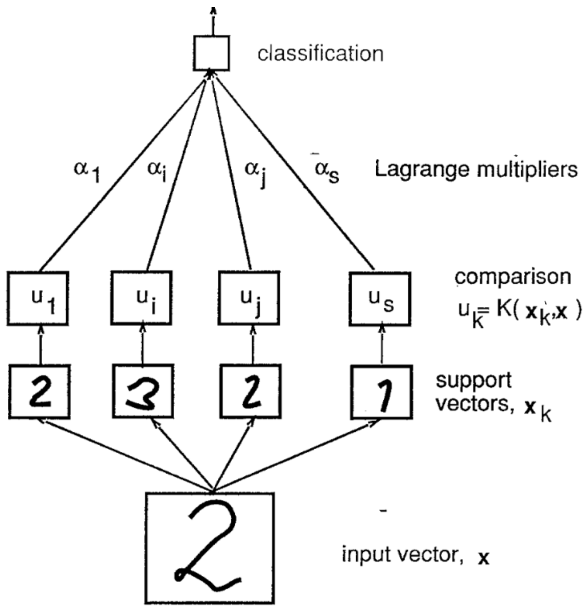

Fig. 4 A pictorial representation of how a SVM model tackles *technical* problem.

4 Hyperplanes

Until now we understand that a classifier(SVM in this case), mainly looks to classify a data point into a distinct class. SVM looks to find an *Optimal hyperplane* that can separate the two classes with maximal margin. This works efficiently in case of input data that is linearly separable or otherwise known as separation of training data without errors. However in case of classification of training data with errors, the separation can be seen to be only approximately linear allowing a small amount of error as shown in *cite soft image here*. Here a straight line or a hyperplane cannot drawn to classify the data points. SVM tackles such training data with errors, with the formulation of a hyperplane with *Soft margin*. In this section we dive into the algorithms of finding an Optimal Hyperplane(in case of training data without errors) and a hyperplane with Soft margin(for the case of training data with errors). We not only look at constructing hyperplanes in Input space but also in feature space. The authors talk about a method of *Convolution of Dot-product* in feature space.

4.1 Optimal Hyperplane Algorithm

This is the first version of SVM which looks to classify two classes that are linearly separable. For such classes, a hyperplane is constructed that separates the two classes with maximum margin as shown in the fig 5.

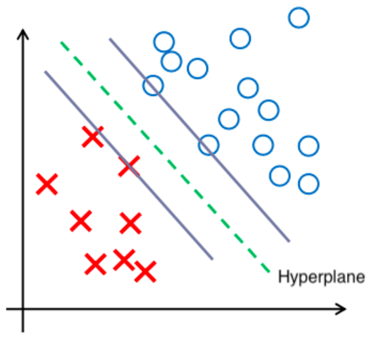

Fig. 5 Construction of optimal hyperplane in case of linearly separable classes.

Let us consider the training data, $x_1 = (x_{1,1}, ..., x_{1,p})$ ... $x_l = (x_{l,1}, ..., x_{l,p})$, where $x_1, ..., x_l$ are the input vectors each of length p, as p denotes the number of features. Let labels be $y_1, ...y_l$ where $y_i \epsilon \{-1, 1\}$. That means each input vector x_i is associated with one label y_i which can take either 1 or -1 as values.

$$(y_1, x_1), (y_2, x_2), ..., (y_l, x_l), \qquad y_i \epsilon \{-1, 1\} \tag{8}$$

for the above training data, given a vector w and a scalar quantity b, the hyperplane can be shown as a function of x_i as follows,

$$\begin{aligned} f(x_i) = w \cdot x_i + b > 0, &\qquad then\ y_i = 1, \\ f(x_i) = w \cdot x_i + b < 0, &\qquad then\ y_i = -1, \end{aligned} \tag{9}$$

If $f(x_i)$ is further away from 0, it is easier to classify and we are more certain of its class. If it is closer to 0, then it is less certain of the correctness of the classification. In case of linearly separable training data, equation 9 now transforms as shown below,

$$\begin{aligned} w \cdot x_i + b \geq 1, \quad &then\ y_i = \ 1, \\ w \cdot x_i + b \leq -1, \ &then\ y_i = -1, \end{aligned} \tag{10}$$

Here the value of $f(x_i)$ has to be strictly positive ($\geq$ 1) or strictly negative($\leq$ -1) for it to be classified into one of the 2 classes. Equation 10 can be written in a compact form. This then acts as the constraint for linear separability.

$$y_i(w \cdot x_i + b) \geq 1, \qquad i = \ 1, ..., l \tag{11}$$

An optimal hyperplane is defined in 12,

$$w_0 \cdot x + b_0 = 0 \tag{12}$$

But if the data can be optimally separated linearly and classified into two classes, that means there exists a number of hyperplanes that can be chosen from. To find the optimal hyperplane that maximizes the margin, we need to find the direction of hyperplane such that there is maximum distance between the training vectors of 2 classes. We define this distance between the two classes as $\rho(w, b)$. So here we need to take the points from the 2 classes that are exactly on the margin(i.e distance between points when $y_i = 1$ and $y_i = -1$).

$$\rho(w, b) = min_{(x:y=1)} \frac{x \cdot w}{\mid w \mid} - max_{(x:y=-1)} \frac{x \cdot w}{\mid w \mid} \tag{13}$$

From equation 10 we can say that, $w \cdot x_i \geq 1 - b$ and $w \cdot x_i \leq -1 - b$. Replacing the same in above equation we get,

$$\rho(w, b) = \frac{1 - b}{\mid w \mid} - \frac{-1 - b}{\mid w \mid} = \frac{2}{\sqrt{w \cdot w}} \tag{14}$$

Now it is clear that for a optimal hyperplane, we need to get the maximal distance(ρ) and to get the maximal distance we look to minimizing $w \cdot w$ in 14. Thus we can consider this as a classic optimization problem.

$$\begin{aligned} min \quad \phi = w \cdot w \\ s.t \quad y_i(w \cdot x_i + b) \geq 1, \ i = \ 1, ..., n \end{aligned} \tag{15}$$

For standard optimization,we choose to construct a Lagrangian here. In order to find the maximum or minimum of a function $f(x, y)$ subjected to some constraint $g(x, y) = 0$, Lagrangian function is of the form,

$$\mathcal{L}(x, y, \Lambda) = f(x, y) - \Lambda g(x, y) \tag{16}$$

In our case, we construct a Lagrangian, where $\Lambda^T = (\alpha_1, ..., \alpha_l)$ is a vector of Lagrangian multipliers.

$$\mathcal{L}(w, b, \Lambda) = \frac{1}{2} w \cdot w - \sum_{i=1}^{l} \alpha_i [y_i(w \cdot x_i + b) - 1] \tag{17}$$

So in order to solve this optimization problem, we need to minimize 17 with respect to w and b and maximize with respect to Lagrangian multiplier Λ. Taking partial derivatives on 17 we get below equations,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} \mid_{w=w_0} = w_0 - \sum_{i=1}^{l} \alpha_i y_i x_i = 0 \\ \frac{\partial \mathcal{L}}{\partial b} \mid_{b=b_0} = \sum_{i=1}^{l} y_i \alpha_i = 0 \end{aligned} \tag{18}$$

From 18 we can express that the optimal hyperplane is linear combination of input vectors. But only training vectors for which $\alpha_i > 0$ is counted in the summation as show below,

$$w_0 = \sum_{i=1}^{l} \alpha_i y_i x_i \tag{19}$$

Substituting 19 and 18 in 17 we get,

$$\begin{aligned} W(\Lambda) &= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} w_0 \cdot w_0 \\ W(\Lambda) &= \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \end{aligned} \tag{20}$$

The above equation can now rewritten in terms of the vector Λ and we get the following,

$$W(\Lambda) = \Lambda^T 1 - \frac{1}{2} \Lambda^T D \Lambda \tag{21}$$

1 denotes a l dimensional unit vector and D is matrix of size $l \times l$. D is as given below. With $\Lambda^T = (\alpha_i, ..., \alpha_l)$ and $Y^T = (y_1, ..., y_l)$, now conditions in 18 translate to constraints as in 23

$$D_{ij} = y_i y_j x_i \cdot x_j \tag{22}$$

$$\begin{aligned} \Lambda &\geq 0 \\ \Lambda^T Y &= 0 \end{aligned} \tag{23}$$

We therefore have to maximize equation 21 subject to constraints in 23. Now according to Kuhn-Tucker theorem, at the optimal point i.e $(w_0 b_0 \Lambda_0)$, any Lagrangian multiplier and its constraint are related by the below equality,

$$\alpha_i [y_i (w_o \cdot x_i + b_0) - 1] = 0 \qquad i = 1, ..., l \tag{24}$$

From 24 we can achieve non zero values for α_i only when,

$$y_i (w_o \cdot x_i + b_0) - 1 = 0 \tag{25}$$

We know to achieve an optimal hyperplane we need not consider all data points but only points that satisfy 25. Such vectors for which α_i can take only strictly positive values and satisfies condition 25 are called as *support vectors.* Using the Kuhn-Tucker algorithm, we can also find a relation between the maximal value $W(\Lambda)$ and the distance of separation(ρ_0). From 18 and 19 we get,

$$w_0 \cdot w_0 = \sum_{i=0}^{l} \alpha_i^0 y_i x_i \cdot w_0 = \sum_{i=0}^{l} \alpha_i^0 (1 - y_i b_0) = \sum_{i=0}^{l} \alpha_i^0 \tag{26}$$

Substituting the above equation in 20 to obtain the maximum value for $W(\Lambda_0)$

$$W(\Lambda_0) = \sum_{i=1}^{l} \alpha_i^0 - \frac{1}{2} w_0 \cdot w_0 = \frac{w_0 \cdot w_0}{2} \tag{27}$$

With reference to 14, for optimal hyperplane(ρ_0) we can then show that,

$$W(\Lambda_0) = \frac{2}{\rho_0^2} \tag{28}$$

As we know, maximizing the equation 21 subject to constraints in 23, can either lead to an optimal hyperplane separating the training data or the data cannot be separated by any hyperplane. For a large dataset, maximizing such a functional can be tricky, here we first divide the training data into small portions. We determine and solve the quadratic equation for the first portion of the data, If this portion of data cannot be separated by a hyperplane, we can assert that the full set of data cannot be separated. However if the portion data is separated, then that becomes the optimal hyperplane for that portion of data. Now we combine all the support vectors from the first portion with all the training vectors of the second portion. For this set, a new hyperplane is then determined and the process is continued until all the portions of data are covered. During this process as more and more training vectors are included, the gap between the two classes narrows down.

4.2 Soft margin Hyperplane Algorithm

Soft margin hyperplane construction is an extension of approach applied in strictly separable case(4.1). Fig6 shows data points that cannot be separated by drawing a straight line.

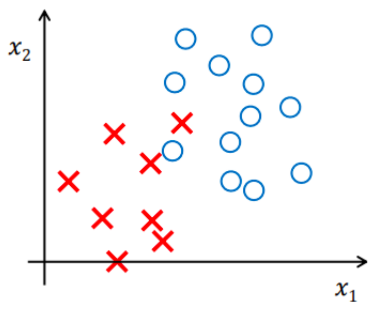

Fig. 6 Case of linearly separable classes but with some data points as errors or wrongly classified.

Here we deal with training data that cannot be separated without errors. As the errors become inevitable we look to minimize the error as much as possible. Along with all the notations that were considered in section 4.1, we introduce a non negative variable ξ to express the error(7). Thus, $\xi_i \geq 0$ for $i = 1, 2, ..., l$. We consider a function $\phi(\xi)$ and look to minimize this function with a set of constraints.

$$\phi(\xi) = \sum_{i=1}^{l} \xi_i^{\sigma} \tag{29}$$

Here, constraints are that σ is a constant > 0 but as small as it can be. And thus equation 11 is modified in the next step,

$$\begin{aligned} y_i(w \cdot x_i + b) \geq 1 - \xi_i, \qquad i = &\ 1, ..., l \\ \xi_i \geq 0, \qquad i = &\ 1, ..., l \end{aligned} \tag{30}$$

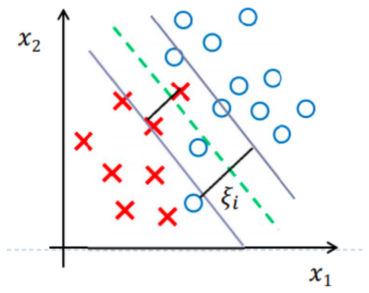

Fig. 7 Hyperplane with soft margin for linearly separable classes, with some errors.

When σ is sufficiently small, 29 denotes the number of training errors. We can find a minimal subset of training errors by minimizing the function in 29. We know that if we can separate these errors from the set of training data, we can then separate the remaining training vectors without errors. Constructing an optimal hyperplane for such a set of training vectors involves minimizing a function formally defined here,

$$\frac{1}{2}w^2 + CF\Big(\sum_{i=1}^{l}\xi_i^{\sigma}\Big) \tag{31}$$

where, C is a constant and $F(u)$ is a monotonic convex function(Any function on an interval is called a monotonic convex function, only if its derivative is monotonically non-decreasing on that interval). For the case where σ is small and C is sufficiently large, then a w_0 and b_0 that minimizes equation 31, constructs a hyperplane that separates all the training vectors with a maximum margin after minimizing errors on the training data. We consider $\sigma = 1$ and a large C in maximizing equation 31, subjecting to constraints in 30 and the authors call such a solution as *soft margin hyperplane*.

Here we start with the case where $F(u) = u^k$ and $k > 1$ then extend this result to monotonic function $F(u)$. We maximize the following function subject to 30,

$$\phi = \frac{1}{2}w \cdot w + CF\Big(\sum_{i=1}^{l}\xi_i\Big)^k \tag{32}$$

Lets take a Lagrangian for the above equation and we get,

$$\mathcal{L}(w,\xi,b,\Lambda,R) = \frac{1}{2}w \cdot w + C\Big(\sum_{i=1}^{l}\xi_i\Big)^k - \sum_{i=1}^{l}\alpha_i[y_i(w \cdot x_i + b) - 1 + \xi_i] - \sum_{i=1}^{l}r_i\xi_i \tag{33}$$

where $R^T = (r_1, ..., r_l)$ is a vector of multipliers that enforces the constraint, $\xi_i \geq 0$. We minimize 33 with respect to w_i, b, ξ_i and maximize with respect to α_i and r_i variables. Taking partial derivatives of 33 with condition $w = w_0, b = b_0$ and $\xi = \xi_0$, we get the following equations.

$$\frac{\partial \mathcal{L}}{\partial w}\mid_{w=w_0} = w_0 - \sum_{i=1}^{l}\alpha_i y_i x_i = 0 \tag{34}$$

$$\frac{\partial \mathcal{L}}{\partial b}\mid_{b=b_0} = \sum_{i=1}^{l}\alpha_i y_i = 0 \tag{35}$$

$$\frac{\partial \mathcal{L}}{\partial \xi}\mid_{\xi=\xi_0} = kC\Big(\sum_{i=1}^{l}\xi_i^0\Big)^{k-1} - \alpha_i - r_i \tag{36}$$

if we consider,

$$\sum_{i=1}^{l} \xi_i^0 = \left(\frac{\delta}{kC}\right)^{\frac{1}{k-1}} \tag{37}$$

then equation 36 becomes,

$$\delta - r_i - \alpha_i = 0 \tag{38}$$

Replacing equations 34 through 38 in Lagrangian 33 we get the following,

$$W(\lambda, \delta) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j x_i \cdot x_j - \frac{\delta^{k/k-1}}{(kC)^{1/(k-1)}}\left(1 - \frac{1}{k}\right) \tag{39}$$

Now to find a hyperplane with soft margin, we need to maximize 39 with respect to α_i and r_i, subject to following conditions.

$$\begin{aligned} w_0 &= \sum_{i=1}^{l} \alpha_i y_i x_i \\ \sum_{i=1}^{l} \alpha_i y_i &= 0 \\ \delta &= r_i + \alpha_i \end{aligned} \tag{40}$$

If we can rewrite equation 39 using vector notation we get,

$$W(\Lambda, \delta) = \Lambda^T 1 - \left[\frac{1}{2}\Lambda^T D \Lambda - \frac{\delta^{k/k-1}}{(kC)^{1/(k-1)}}\left(1 - \frac{1}{k}\right)\right] \tag{41}$$

Constraints are now as follows,

$$\Lambda^T Y = 0 \tag{42}$$

$$\Lambda + R = \delta 1 \tag{43}$$

$$\Lambda \geq 0 \tag{44}$$

$$R \geq 0 \tag{45}$$

In order to maximize 41, from the conditions 43 and 45 we can equate,

$$\delta = \alpha_{max} = max(\alpha_1, ..., \alpha_l) \tag{46}$$

Now 41 becomes,

$$W(\Lambda, \delta) = \Lambda^T 1 - \left[\frac{1}{2}\Lambda^T D \Lambda - \frac{\alpha_{max}^{k/k-1}}{(kC)^{1/(k-1)}}\left(1 - \frac{1}{k}\right)\right] \tag{47}$$

In order to simplify the above equation, authors consider $k = 2$ and we get,

$$W(\Lambda, \delta) = \Lambda^T 1 - \frac{1}{2}\left[\Lambda^T D \Lambda - \frac{\alpha_{max}^2}{C}\right] \tag{48}$$

Thus to find a soft margin classifier, we find Λ that maximize 48 such that $\Lambda \geq 0$ and $\Lambda^T Y = 0$.

4.3 Convolution of Dot-Product

We look at a newer approach for situations where the data cannot be linearly separated as seen in fig 8. In such cases we transform the input into a higher dimensional space and then constructs a linear hyperplane in that high dimensional space. The "input space" is defined as collection of all the possible inputs. Feature spaces, on the other hand, include the feature vectors from a given set of data. They may not contain all the possible inputs for a model.

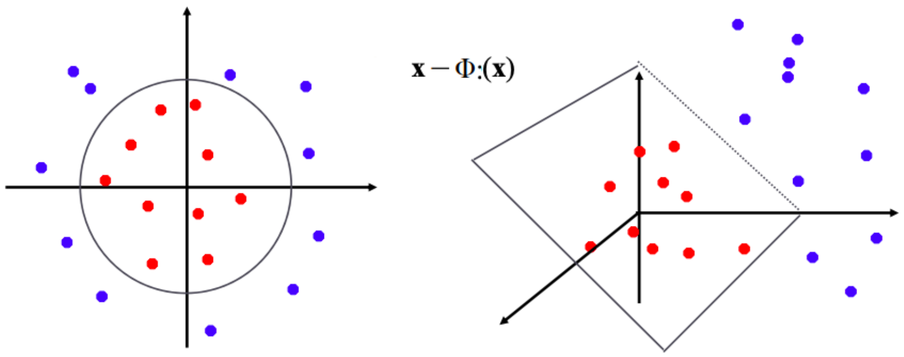

Fig. 8 Non-linearly separable classes in input space and linearly separable data in feature space.

This algorithm transforms a n-dimensional input vectors in input space to N-dimensional vectors in feature space, using a ϕ function.

$$\phi : \Re^n \to \Re^N$$

The vector in input space is first transformed into N-dimensional feature space,

$$x \mapsto \phi(x)$$

Classification of such vectors in feature space is done by taking sign of the following function,

$$f(x) = w \cdot \phi(x) + b \tag{49}$$

According to method proposed under Soft margin hyperplanes section, we can define w as a linear combination of support vectors in feature space,

$$w = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i) \tag{50}$$

Now equation 49 becomes,

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i \phi(x_i) \cdot \phi(x) + b \tag{51}$$

We now understand that any classification function $f(x)$ depends on the dot product of the vector in input space(x) and transformed vector in feature space(x_i). Now in order to find the dot product, we apply the Hilbert-Schmidt Theory. Now we equate the dot product with a function $K(u, v)$,

$$\phi(u) \cdot \phi(v) \equiv K(u, v) \tag{52}$$

Here $K(u, v)$ is a symmetric function according to the Hilbert-Schmidt theory and can be described as following,

$$K(u, v) = \sum_{i=1}^{\infty} \lambda_i \phi_i(u) \cdot \phi_i(v) \tag{53}$$

where $\lambda_i \epsilon \Re$ and ϕ_i are eigenvalues. For 52 to represent a dot product in a feature space, the eigenvalues in the expansion 53 have to be positive. For positive eigenvalues we need to satisfy the following necessary and sufficient condition,

$$\int\int K(u, v)g(u)g(v)dudv > 0$$

Here all g need to satisfy the following condition,

$$\int g^2(u)du < \infty$$

This is called $Mercer's$ Theorem and any function that satisfies the $Mercer's$ theorem can be used as dot products in Convolution. History says that a variety of functions($K(u, v)$) that satisfies the $Mercer's$ theorem were used. Using such different functions can lead to different learning machines and can construct different types of decision surfaces. In general, the following gives us a function for such decision surfaces. These functions are called Potential functions, Kernel functions etc.

$$f(x) = \sum_{i=1}^{l} \alpha_i y_i K(x, x_i)$$

Here x_i is an image of x in feature space, since this is a function for forming decision surfaces, x_i is also a support vector. Since we know using different Kernel functions result in different learning machines, lets look at different kernel functions. Kernel function for constructing a polynomial classifier of degree d can be shown to be,

$$K(u, v) = (u \cdot v + 1)^d$$

In an experiment by Aizerman, Braverman and Rozonoer, the considered function for the convolution of dot products in features is given by,

$$K(u, v) = exp\big(- \frac{\mid u - v \mid}{\sigma}\big)$$

A convolution function for a typical two layered neural network is defined. Here b is a scaling parameter of input and c controls the threshold of mapping.

$$K(u, v) = tanh(b(u \cdot v) - c)$$

5 Experiments and Results

In this section we look at experiments that were performed by the authors(of [1]), in order to demonstrate the working of proposed Support Vector Machine. 2nd degree polynomial decision surfaces were constructed for different sets of data points. Real life problem of Digit Recognition were also dealt using the proposed support vector network approach.

5.1 Experiments in the Plane

To construct a polynomial of 2nd degree the following dot product form was used. With the help of this different sets of data points were classified and visualized.

$$K(u, v) = (u \cdot v + 1)^d \tag{54}$$

Here $d = 2$, as we are looking at a 2nd degree polynomial. After classification, the data points of two classes were then represented using black and white bullets. All the support patterns or support vectors were then marked with a double circle. Data points that were misclassified are denoted by a cross. The following figure shows a visualization of classification done for two different examples.

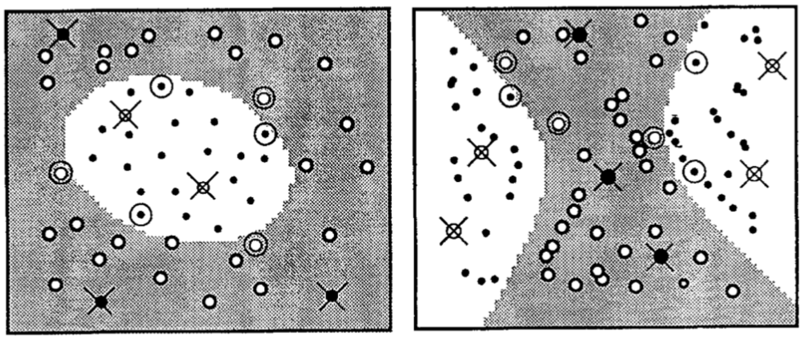

Fig. 9 Examples of data points being classified using a dot product with $d = 2$

The above figure shows the most optimal solution for a 2nd degree polynomial. We can also notice that the number of support vectors used is very small relative to the number of entire training vectors.

5.2 Digit Recognition Experiments

In this section, we look at how the support vector networks method was applied on Digit Recognition problem using two real life databases. Polynomials of different degree were used in the experiment on two databases. The two databases that were considered during the experiment were of different sizes also.

US Postal Service Database was the smaller databases and it consisted of 7300 training vectors and 2000 test vectors. All the images in the database were of 16×16 resolution. Whereas the *NIST Database* is the larger database consists of images of 28×28 resolution. This database contains 60,000 training vectors and 10,000 test vectors. The constructed classifiers for both the databases were then compared with other state of the art learning machines of the authors' time.

US Postal Service Database: In this experiment is based on a database that has been reported by several researchers and contains information from actual mail pieces. US Postal Services Database has been experimented with various classifiers and approaches. J. Bromley and E. Sackinger(1991) obtained results based on Human performance and an error of 2.5% was recorded. A decision tree approach called CART was used by Daryl Pregibon and Michael D. Riley at Bell Labs., Murray Hill, NJ. and recorded an error rate of 17%. Corinna Cortes carried out experiments with $C4.5$ algorithm based approach and Bernard Schoelkopf recorded with the best $2layer$ neural network. All the classifiers and recorded error rates are listed in table 1. Different preprocessing techniques like centering, de-slanting and smoothing were used on the database to cover for all the inconsistencies

Table 1 List of approaches and respective error rates recorded on US Postal Services Database [1].

Classifier/Approach	Error Rate
Human Performance	2.5
Decision Tree, CART	17
Decision Tree, C4.5	16
Best 2 layer neural network	6.6
Special Architecture 5 layer network	5.1

in it. In the experiment conducted by the authors of [1], a smoothing kernel as a Gaussian with a standard deviation(σ) of 0.75 was used.

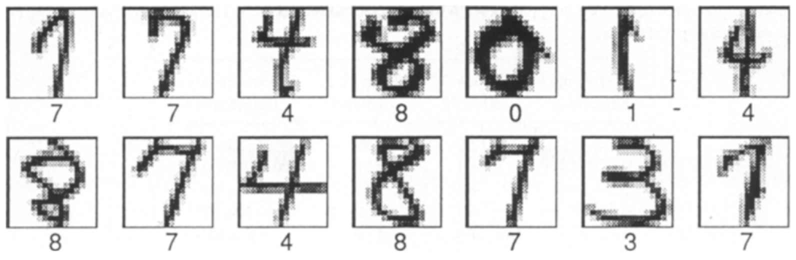

Fig. 10 Representation of samples and associated labels from US Postal Services Database

As part of the experiments in [1], various degree of polynomial classifiers($1 - 7$) with a 256-dimensional input was used. The data in the following table 2 reads that 7th degree polynomial has 190 support vectors and a 3rd degree polynomial has 148 support vectors, which suggests that the number of support vectors increases slowly. Since the dimensionality of the feature space does not affect the performance of SVM, the increase in feature space dimensionality has no effect. A

Table 2 Performance of various degree of polynomial classifier on US Postal Services Database.

Degree of Polynomial	Error Rate(%)	No.of Support Vectors	Dimensionality of feature space
1	12	200	256
2	4.7	127	~33000
3	4.4	148	$\sim 1\times10^{6}$
4	4.3	165	$\sim 1\times10^{9}$
5	4.3	175	$\sim 1\times10^{12}$
6	4.2	185	$\sim 1\times10^{14}$
7	4.3	190	$\sim 1\times10^{16}$

linear classier(of degree 1) has 34 misclassifications and the number decreases to 4 for the 2nd degree classifier. A linear classifier has a maximum of 3% as an upper bound of error probability and the error probability during testing is only 1.5%. Training time taken by an SVM in construction of polynomial classifiers of degree($2 - 7$) depends on the number of support vectors and hence performs faster than the best neural network($LeNet1$ with an error rate of 5.1%, constructed by LeCun in 1990).

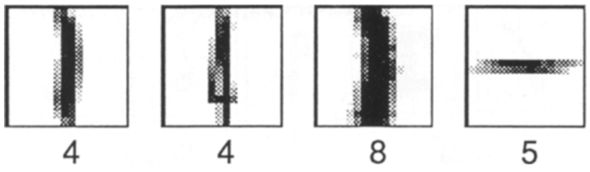

Fig. 11 4 misclassified patterns by 2nd Degree polynomial classifier, along with their respective labels.

NIST Database: To classify patterns in *NIST* database, here only one type of classifier was used. A 4th degree polynomial classifier was used on data with no pre-processing. Table 3 gives a details of performance for 10 classifiers trained on a set of 60000 patterns and tested on 10000 patterns from NIST database.

Table 3 Performance of 10 classifiers on NIST Database.

Titles	**Cl. 0**	**Cl. 1**	**Cl. 2**	**Cl. 3**	**Cl. 4**	**Cl. 5**	**Cl. 6**	**Cl. 7**	**Cl. 8**	**Cl. 9**
No.of Support Vectors	1379	989	1958	1900	1224	2024	1527	2064	2332	2765
Error %(Train Data)	7	16	8	11	2	4	8	16	4	1
Error %(Test Data)	19	14	35	35	36	49	32	43	48	63

For classifier 1(Cl. 1), there were 14 misclassifications in test patterns and on an average error was recorded to be 12 per class. For all 10 classifiers combined, error rate on the test set which contains 10000 patterns is 1.1%.

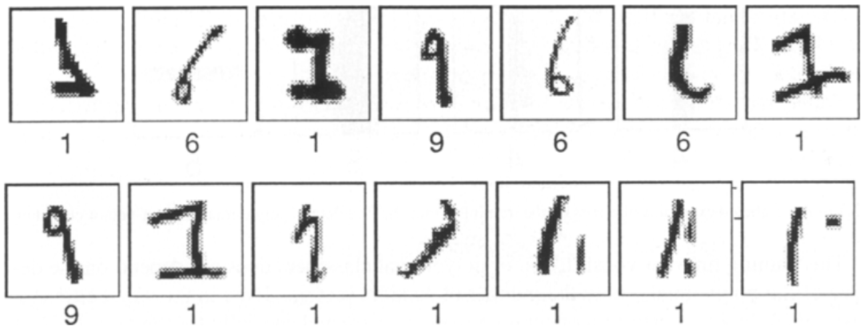

Fig. 12 14 misclassified patterns by 4th Degree polynomial classifier(Cl. 1), along with their respective labels.

The paper [1] also describes us important results from a benchmark study conducted by Bottou in 1994. The SVM results that was compared in the study is were provided by the authors of this paper. The SVM approach was compared with neural network classifiers like LeNet1 and LeNet4, $k = 3$ nearest neighbor classifier and also a linear classifier.

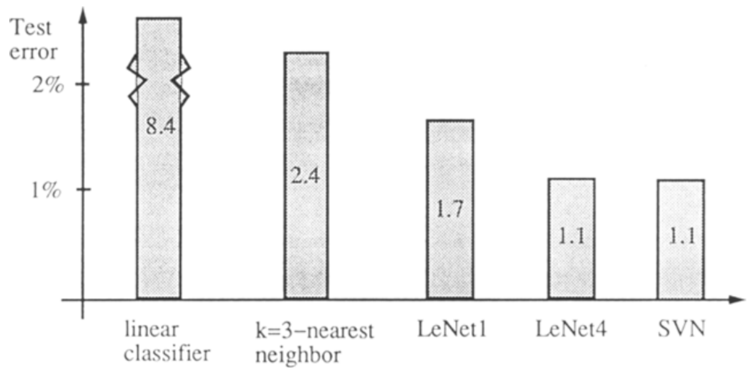

Fig. 13 Results of different classifiers that were compared in the benchmark study.

As stated in the benchmark study, support vector network as classifiers has outperformed LeNet1 and LeNet2(which were considered state of the art), by showing remarkable results.

6 Discussion

This report provides an understanding of a machine learning approach called *Support Vector Networks*. This approach separates the training data by constructing an hyperplane with a maximum margin. Through the course of the report we provided an overview on optimal hyperplane algorithm, used to separate training data without errors. Here the support vectors are extended to form an optimal hyperplane with maximum margin. A soft margin method for constructing hyperplanes was then introduced, which tackled training data with errors. We also looked into the convolution of dot-products approach for constructing a linear decision hyperplane in feature space. SVN approach was then applied on two popular databases to tackle digits recognition problem. The results then showed us that Support Vector Networks is a powerful approach and has the capability to outperform other classifiers. The ability of the learning machine to control its generalization ability makes it a remarkable, universal solution. Overall, SVM's are intuitive, theoretically well founded, and have shown to be practically successful.

References

1. Corinna Cortes and Vladimir Vapnik: Support-Vector Networks: Mach. Learn. https://doi.org/10.1007/BF00994018
2. M.O. Stitson, J.A.E. Weston, A Gammerman, V.Vork, V. Vapnik, "Theory of Support Vector Machines", Technical Report CSD-TR-96-17, Department of Computer Science, Royal Holloway College, University of London, Dec. 1996.
3. Fisher, R.A. (1936). The use of multiple measurements in taxonomic problems. Ann. Eugenics, 7:111-132
4. Bishop, Christopher, Pattern Recognition and Machine Learning, 2006.
5. Bernhard E. Boser and Isabelle Guyon and Vladimir Vapnik. A Training Algorithm for Optimal Margin Classifiers. Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992
6. Aizerman, M., Braverman, E., Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, 25:821-837.
7. Nagesh Singh Chauhan, A Friendly Introduction to Support Vector Machines https://www.kdnuggets.com/2019/09/friendly-introduction-support-vector-machines.html
8. Rohith Gandhi, Support Vector Machine — Introduction to Machine Learning Algorithms https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

9. Akash Kandpal, Getting Started with Data Science using Python https://codeburst.io/getting-started-with-maths-for-data-science-part1b-d1203447e4bc
10. Lagrange multipliers https://en.wikipedia.org/wiki/Lagrange_multiplier
11. Hilbert space https://en.wikipedia.org/wiki/Hilbert_space