Learning with Supportive Vectors

An Introduction to Support Vector Machines and their Applications

Purushottam Kar
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, INDIA
purushot@cse.iitk.ac.in

ABSTRACT

Support Vector Machines have acquired a central position in the field of Machine Learning and Pattern Recognition in the past decade and have been known to deliver state-of-the-art performance in applications such as text categorization, hand-written character recognition, bio-sequence analysis, etc. In this article we provide a gentle introduction into the workings of Support Vector Machines (also known as SVMs) and attempt to provide some insight into the learning mechanisms involved. We begin with a general introduction to mathematical learning and move on to discuss the learning framework used by the SVM architecture.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Concept learning; A.1 [**General Literature**]: INTRODUCTORY AND SURVEY

General Terms

Theory, Algorithms

Keywords

Machine Learning, Support Vector Machines, Kernels

1. THE LEARNING METHODOLOGY

Before we move into algorithms that learn, let us take a look at what mathematical learning (or machine learning) means and why is it an interesting field of study. Consider the following computational problems and try to see if one can write C++ programs (feel free to replace C++ with your favorite language) to distinguish between the following -

1. A correct and an incorrect Java program
2. A palindrome word (like "detartrated") and a non-palindrome word (like "house")
3. Graphs which have a Hamiltonian path and those which do not

One finds that for all these classification problems, one can write programs (however inefficient they may be). The set of problems given above are often known as **Classification Problems**. A classification problem is simply the problem of distinguishing between objects belonging to some fixed number of classes (viz. the class of palindrome words and the class of non-palindromes). Now consider the problem of distinguishing between the following -

1. An image of a handwritten 4 and a handwritten 9
2. A spam email and a non-spam e-mail
3. A positive movie review and a negative movie review

We find that apriori there does not seem to exist a set of well-defined rules that characterize the classes involved in the above tasks. Consequently one cannot simply sit down and write an algorithm to perform these classification tasks (the author feels that even standing might not help). Moreover in some of the tasks, the classification itself is not well defined (for example, the author might find all the e-mails from the deans to be spam-like but a good student might read them with due diligence). In such cases the best one can hope to do is try to inductively learn some of the latent (or hidden) patterns and rules that govern the classification. This learning one does when provided with **Training Sets**. For example, in the spam classification case, we provide our Thunderbird client (or in some woeful cases our Outlook client) with examples of e-mails we consider to be spam and examples of non-spam e-mails. The learning algorithms that come packaged with these clients then try to discover the pattern(s) underlying our choices using the training examples and use these learnt rules to classify a new e-mail as spam/non-spam. Thus the main objective in machine learning is to let the training data decide the eventual classification algorithm.

2. LINEAR CLASSIFIERS

In mathematical learning, these (unknown) underlying rules or patterns are abstracted as a mathematical function whose output on the objects of interest (images, e-mails, movie reviews etc.) decides their classification. Thus instead of trying to learn individual rules and patterns (which is the goal in a subfield of Artificial Intelligence called Inductive Rule Inference), we strive to directly approximate this unknown function as closely as possible. The simplest of such functions are linear functions. To understand this better, consider the simple case when our objects are vectors in a 2-dimensional Euclidean space and belong to either the BLACK class or the WHITE class as shown in Figure 1. The example contains 14 WHITE objects and 14 BLACK objects which have been given to us as the training set. It can be clearly seen that the hyperplane drawn separates the BLACK instances from the WHITE ones and classifies the elements of the data set perfectly with no errors. Such a classifier is said to be **consistent** with the training set.[1]

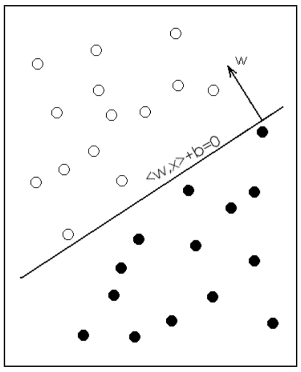

Figure 1: A Linear Classifier

In order to construct a mathematical representation of the classifer, consider the linear function $f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle + b$ (where $\langle \cdot, \cdot \rangle$ denotes the dot product and $b \in \mathbb{R}$). This function classifies a vector $\vec{x}'$ as BLACK if $f(\vec{x}') < 0$ and WHITE otherwise. To be more precise, we can say that our classifier is the function $g(\vec{x}) = \text{sgn}(f(\vec{x}))$ where sgn is the signum function. Notice that the hyperplane drawn is not the only one that is consistent with the dataset. In fact there are infinitely many hyperplanes which classify these 28 points correctly. This also shows us that a classifier that is error-free on the training set may make errors on new data points.

Such a problem (in this case that of finding a linear classifier) where there exist multiple solutions is said to be an ill-posed in mathematical literature. Ill-posed problems do not bode well for learning tasks because the outcome of a learning algorithm in such cases is typically sensitive to initial conditions. For example the Perceptron Algorithm [Ros58] is an algorithm that given these 28 training vectors, finds ("learns") a linear hyperplane classifying these vectors correctly by starting with a (possibly bad) hyperplane and improving it iteratively to make it classify each vector correctly. However the hyperplane it ends up learning depends upon the hyperplane it started out with. Similar is the case with the Artificial Neural Networks framework [DHS06].

In such situations it is difficult to say anything about the learnt classifier with respect to its performance on new data (i.e. how frequently is it expected to make an error) which is what we are most interested in. In other words nothing much can be said about the **Generalization Performance** of the classifier on unseen data. The way out of this is to make the problem well-posed (i.e. have a unique solution) by way of **Regularization**. In very broad terms regularization entails assigning with each one of the (infinitely many) solutions, a goodness value and then searching for the best solution, i.e. the one with the highest value of goodness. Of course in order to be effective, the regularization step should be such that the best solution according to the goodness measure used is unique and has useful properties.

In the following section we shall study at the regularization step used to arrive at the SVM algorithm in some detail.

3. LARGE MARGIN CLASSIFIERS

Given a hyperplane $\langle \vec{w}, \vec{x} \rangle + b = 0$ and a vector $\vec{x_0}$, the **Geometric Margin** of the point[2] with respect to this hyperplane is defined to be the distance of the point from the hyperplane (actually this quantity is taken with sign but we shall ignore this technicality for the moment). A little bit of high school geometry shows us that this is given by the expression $\frac{|\langle \vec{w}, \vec{x}_0 \rangle + b|}{\|\vec{w}\|}$ (where $\| \cdot \|$ gives the length of a vector). A point that is correctly classified by the hyperplane is said to have a positive margin of $+\frac{|\langle \vec{w}, \vec{x}_0 \rangle + b|}{\|\vec{w}\|}$ whereas a misclassified point has a negative margin of $-\frac{|\langle \vec{w}, \vec{x}_0 \rangle + b|}{\|\vec{w}\|}$.

Now it is clear that any hyperplane correctly classifying all the points will have a non-negative margin with respect to all the points. Suppose we have a training set that is **Linearly Classifiable** i.e. there exists a hyperplane correctly classifying all the points. For any hyperplane let us take the minimum margin of that hyperplane on any training point as the regularization parameter (using the interpretation developed above, this the the closest any data point gets to the hyperplane). Now let us look for the hyperplane with the maximum value of this parameter (in particular we know that the maximum value will be positive since we know the data set to be linearly classifiable). It can be shown that there is a unique hyperplane with the largest value of this parameter. In the figure below (Figure 2) we take the same training set as in Figure 1 and draw the hyperplane with the maximum margin.

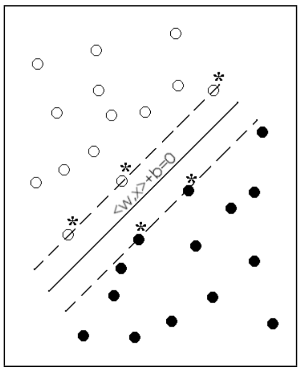

Figure 2: The Maximum Margin Linear Classifier - the starred points are the support vectors

This unique hyperplane is known as the maximum margin

[1]Note that we may been given a dataset which no hyperplane is able to classify perfectly - such datasets are called **Non-linearly Classifiable**. However for now lets live in a simpler world where training sets are **Linearly Classifiable**.

[2]The reader is requested to bear with the definition of some technical terms - however it is assured that there will be few of them and that they will be properly explained.

classifier of the training set and it possesses several interesting properties. The vectors which lie closest to this hyperplane (i.e. the ones having minimum margin) are called the **Support Vectors**. It turns out that the direction vector of the hyperplane i.e. $\vec{w}$ is simply a linear combination of the support vectors alone i.e. $\vec{w} = \sum_{\vec{x}_i \in SV} \alpha_i \vec{x}_i$ where SV is the set the support vectors. This fact has important consequences for the classifier which we shall discuss briefly later.

The reason behind the name Support Vector Machines is that the support vectors actually support the hyperplane in the physical sense of the term. If one imagines that each support vector $\vec{x}_i$ is applying a force of α_i (its corresponding coefficient in the representation of $\vec{w}$) on the hyperplane, then it turns out that the total force and torque on the hyperplane due to the support vectors is zero.

However this fact has little significance for learning. What does have significance is that if the margin of the maximum margin classifier is large then one is able to prove probabilistic bounds on the generalization error of the classifier. In other words one is able to make a mathematical statement of the following kind

> Assuming that the training set was chosen randomly from some probability distribution on the Euclidean space, it is very unlikely that a training set will get chosen whose maximum margin classifier has a large positive margin but still makes gross errors on unseen points.

We have glossed over a lot of details (and introduced inaccuracies) while making the above statement but going into any of these details would require a full technical article on Vapnik-Chervonenkis dimension (strictly speaking even this would not suffice) and Probably-Approximately-Correct (PAC) Learning. We provide pointers on these topics for the interested reader toward the end.

This basic SVM model can be tweaked to allow for situations where there is noise and a few outlier training points make it impossible for any linear hyperplane to classify the otherwise linearly classifiable data set. These models are known as Soft-margin SVMs (see Figure 3). The following figure is illustrative of such a situation. Again we choose not to go into the details of the topic.

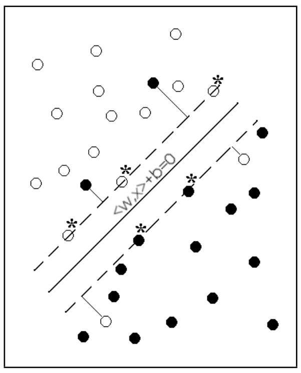

Figure 3: A Soft-margin Linear Classifier - the starred points are the support vectors

3.1 Learning the Maximum Margin Classifier

We briefly address the question of how to learn the maximum margin classifier given a training set of vectors in some Euclidean space (this might sound as a restriction for cases like spam detection where the objects are not real-valued vectors but we shall soon see how this drawback be elegantly overcome). Given a set of training vectors, we formulate a mathematical program (very much like formulations in the familiar linear programming paradigm) to express the problem of maximizing the margin. The program that gets formulated is a Linearly-constrained Quadratic Program and there are several efficient techniques available to solve the program. We give pointers to freely available solvers toward the end.

We now move on to the second most important feature of the SVM technique namely its adaptability to the **Kernel Trick**.

4. THE KERNEL TRICK

An apparent drawback in the description of the SVM algorithm is that it seems to work well only when the data is linearly classifiable or nearly so. What, we may ask, happens when our problem is not so well behaved?

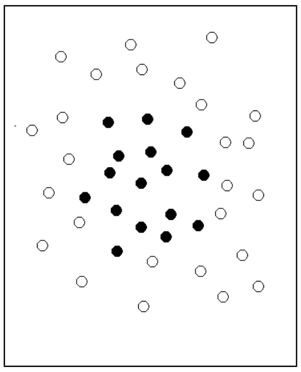

Figure 4: A Non-linear Classification Problem

In such cases what we typically do is apply a non-linear transformation to the data (which usually maps the data to a higher dimensional space) to make it linearly classifiable or very nearly so. To appreciate this method let us consider the problem of learning the Boolean function XOR in a geometric setting. Recall that for two binary digits (or bits) a and b, $XOR(a, b) = (a \wedge \neg b) \vee (\neg a \wedge b)$. Geometrically we can consider framing this problem in $\mathbb{R}^2$ and interpret the coordinate values as bit values. Thus we get the following classification problem with BLACK and WHITE denoting the two classes $XOR = 1$ and $XOR = 0$ respectively.

It turns out that this data set is not a linearly classifiable one as no line can separate the black points from the white ones. But consider the following non-linear map from $\mathbb{R}^2$ to $\mathbb{R}^3$. $\Phi : (x, y) \longmapsto (x^2, y^2, \sqrt{2}xy)$. Figure 6 shows that this map makes the data linearly separable.[3]

Although there are infinitely many such transformations to *linearize* the data, this particular transformation is special because it corresponds to a **Positive Definite Kernel**

[3]For those worried about whether this non-linear map would affect the generalization bounds, all it takes to fix things is a bit more involved theory from functional analysis.

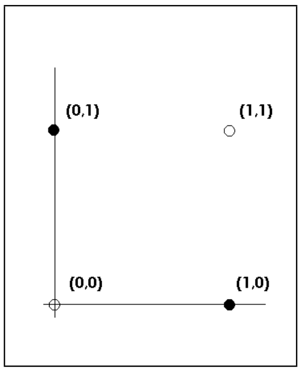

Figure 5: The XOR Classification Problem

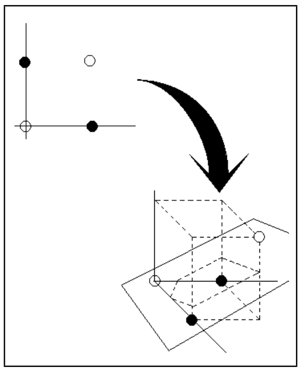

Figure 6: Making the XOR linearly separable

also known as a **Mercer Kernel**. A detailed definition of this term is beyond the scope of this article but we can appreciate the uniqueness of this linear transformation by looking at how the dot product looks like in this transformed space. Given two vectors $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^2$ one can easily see that $\langle \Phi(\vec{x}_1), \Phi(\vec{x}_2) \rangle = \langle \vec{x}_1, \vec{x}_2 \rangle^2$.[4]

This innocuous looking technicality has great significance for the SVM algorithm because of the fact that the SVM algorithm can be expressed in a way that only requires dot products between the training vectors to learn the maximum margin hyperplane and not the training vectors themselves. This is most fortunate since we can now map our vectors to very (very) high dimensional spaces where they have a greater chance of becoming linearly separable but not incur the computational cost of performing the map. However as pointed out this is possible only if the inner product of the mapped vectors in the higher dimensional space can be easily expressed in terms of the original vectors as was the case with the map Φ.

This trick is known as the Kernel Trick and the measure $K(\vec{x}_1, \vec{x}_2) = \langle \Phi(\vec{x}_1), \Phi(\vec{x}_2) \rangle$ corresponding to a map Φ is known as a kernel. But what do we do when we get a new vector to classify? Here's where the fact that the direction vector of the maximum margin classifier is simply a linear combination of some of the training vectors (more specifically the support vectors - see Section 3) comes in handy. To see how let $\vec{w} = \sum_{\Phi(\vec{x}_i) \in SV} \alpha_i \Phi(\vec{x}_i)$. Hence[5]

$$\begin{aligned} f(\vec{x}) &= \langle \vec{w}, \Phi(\vec{x}) \rangle + b \\ &= \left\langle \sum_{\Phi(\vec{x}_i) \in SV} \alpha_i \Phi(\vec{x}_i), \Phi(\vec{x}) \right\rangle + b \\ &= \sum_{\Phi(\vec{x}_i) \in SV} \alpha_i \langle \Phi(\vec{x}), \Phi(\vec{x}_i) \rangle + b \\ &= \sum_{\Phi(\vec{x}_i) \in SV} \alpha_i K(\vec{x}, \vec{x}_i) + b \end{aligned}$$

Thus the kernel measure is sufficient for classifying new points as well. Recall that our classifier is $\text{sgn}(f(x))$. In practice instead of choosing a map, one chooses an appropriate kernel measure which (under some conditions given by a result in Functional Analysis called the Mercer's Theorem) is the inner product of vectors when subjected to some non-linear map and use that to learn a hyperplane via the SVM method - the high dimensional map is always made implicitly and never computed as such. One can also design a kernel by explicitly defining a map Ψ and using the kernel $\widehat{K}(\vec{x}_1, \vec{x}_2) = \langle \Psi(\vec{x}_1), \Psi(\vec{x}_2) \rangle$.

Frequently one uses this trick to map vectors to infinite dimensional spaces (the Gaussian kernel measure $K(\vec{x_1}, \vec{x_2}) = e^{-\frac{\|\vec{x_1} - \vec{x_2}\|}{2\sigma^2}}$ where $\|\vec{x}\|$ is the length of the vector $\vec{x}$ and $\sigma > 0$ is one such kernel). Some care has to be taken when dealing with infinite dimensional spaces but we do not go into those details in this article. The choice of the kernel is the most important one while using the SVM algorithm as this influences the quality of the learnt classifier heavily. However, apart from the choice of the kernel, the SVM algorithm is almost fully automated requiring almost no human intervention. This should be contrasted with Artificial Neural Networks which typically require a lot of manual tuning during the learning process.

It turns out that the kernel trick is more than a just neat way of getting the benefits of working in a high dimensional space without incurring the computational costs. It is also a neat way of handling non numeric data and allows the SVM algorithm to work in a variety of situations. This is because the kernel can be thought of as a measure of similarity (the trivial kernel $K(\vec{x}_1, \vec{x}_2) = \langle \vec{x}_1, \vec{x}_2 \rangle$ is indeed a measure of similarity between the two vectors giving us the cosine of the angle between the vectors in case they are of unit length). Hence one can work with SVMs if one can find an appropriate measure of similarity between the objects concerned. For example when working with e-mails, one has to devise a way of finding out how similar two e-mails (spam or non-spam) are. In many situations, finding measures of similarity between two objects is a much more natural thing to do than converting objects like e-mails to numeric vectors. Of course the similarity measure we construct might not confirm to the Mercer's Theorem we mentioned earlier but even such non-kernel similarity measures are found to give good results in practice.

[4]Note that $\langle \Phi(\vec{x}_1), \Phi(\vec{x}_2) \rangle$ is a dot product in $\mathbb{R}^3$ whereas $\langle \vec{x}_1, \vec{x}_2 \rangle$ is a dot product in $\mathbb{R}^2$.

[5]Once again some technicalities have crept in. However these simply use the fact that the inner (dot) product is a bilinear map i.e $\langle \vec{x} + \vec{y}, \vec{z} \rangle = \langle \vec{x}, \vec{z} \rangle + \langle \vec{y}, \vec{z} \rangle$ and $\langle c_0 \vec{y}, \vec{z} \rangle = c_0 \langle \vec{y}, \vec{z} \rangle$ where $c_0 \in \mathbb{R}$. Actually bilinearity for real spaces dictates that $\langle \vec{x}, \vec{y} + \vec{z} \rangle = \langle \vec{x}, \vec{y} \rangle + \langle \vec{x}, \vec{z} \rangle$ and $\langle \vec{y}, c_0 \vec{z} \rangle = c_0 \langle \vec{y}, \vec{z} \rangle$ as well.

5. APPLICATIONS OF SVMS

SVMs have found widespread application in machine learning problems of which we survey two below. A more comprehensive survey is made impossible by lack of space.

5.1 Handwritten Digit Recognition

This application was mentioned in the seminal paper by Boser, Guyon and Vapnik that introduced the Support Vector Machine algorithm [BGV92]. The data set used was compiled by the United States Postal Service out of zip code numbers written on postal letters. This was a widely studied problem at that time and researchers had done a lot of work on Neural Networks and other algorithms trying to get better results on this dataset.

However the best they could do was achieve an error rate of around 12.7%. However Boser *et al* using simple polynomial kernels (where $K(\vec{x}_1, \vec{x}_2) = \langle \vec{x}_1, \vec{x}_2 \rangle^d$ for some integer $d \geq 1$), achieved an error rate of just 3.2%. This was taken as an illustration of the generalization error bounds at work.

5.2 Text Categorization

This is an example of how one can use the kernel trick to work with innovative similarity measures on the data. The problem was that of categorizing news articles from the Reuters-21578 collection into various topics (viz. sports, entertainment, politics). In this work Joachims [Joa98], used insight from the field of information retrieval research to design similarity measures which gave significant performance gains. Earlier methods like Decision trees and Nearest Neighbor based algorithms were at best able to give accuracy rates of around 80% on test data. The SVM based methods, on the other hand, were able to achieve accuracy rates of $> 86\%$.

In many other fields like bioinformatics, image based object recognition and page ranking for Internet search, SVM-based techniques have been applied with great success. We conclude our discussion here and move on to give references to sources which discuss SVMs and related methods in greater detail.

6. ACKNOWLEDGMENTS

The author thanks Vedula Vijaya Saradhi for comments on an earlier version of the paper.

7. REFERENCES

[BGV92] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *International Conference on Learning Theory*, pages 144–152, 1992.

[DHS06] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* John Wiley & Sons Asia Pvt. Ltd., 2006.

[Joa98] Thorsten Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *European Conference on Machine Learning*, pages 137–142, 1998.

[Ros58] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Ponder Yonder

An excellent introductory text in SVMs is the following

> Nello Cristianini and John Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, 2000.

The authors also maintain a website which has a lot of material on Kernel based methods including SVMs and links to freely available implementations of the SVM algorithm : `http://www.support-vector.net/`

For someone interested in learning more about concepts like PAC learning and the VC-dimension theory which allow SVMs to give generalization error bounds, the following is a good starting point

> Michael J. Kearns and Umesh V. Vazirani, *An Introduction to Computational Learning Theory*, The MIT Press, 1994.

Although not covered in this article, there are algorithms to perform regression and clustering that admit the Kernel trick. A good source to learn about these techniques is the following

> Bernhard Schölkopf and Alexander J. Smola, *Learning with Kernels*, The MIT Press, 2002.