An Introduction to Support Vector Machines

Joseph Dylan Bringley [*]

[*] DePaul University

Submitted to Proceedings of the National Academy of Sciences of the United States of America

This paper examines Support Vectors Machines from a mathematical viewpoint. The math will be discussed and explained explicitly in order to increase readability. SVM will be discussed alongside of Kernel Methods, which are a necessary component of the SVM algorithm. Soft-Margin SVM will not be discussed in this paper.

Support Vector Machines | Kernel Methods | Classification

Introduction

Classification is a long studied idea in statistics and machine learning. The basic idea behind classification, is given a some data set in which class membership is known, we wish to predict which class a new observation will fall. Support Vector Machines were developed in 1963 by two mathematicians named, Vladimir Vapnik and Alexey Chervonenkis, and they try to do just that. Let's try and see what they're all about.

The Algorithm

Let's start off assuming a simple class for SVM, and build upon it as this paper continues. Start with two classes of linearly separable data, and suppose we wish to draw some boundary which correctly separates the classes. If the data is linearly separable, then there are infinitely many lines we can draw to accomplish this task.

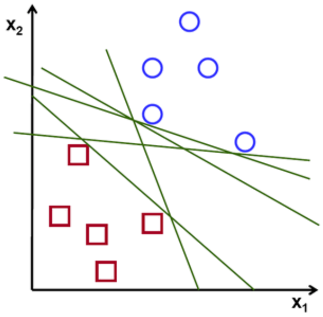


Fig. 1. Many lines can correctly classify the data.

The goal of SVM is to find the optimal line (or in a multidimensional case, hyperplane) that separates the two classes. This hyperplane should maximize the margin between the classes' closest points, which we refer to as support vectors. We call this optimal line the decision boundary, and write it as:

$$w^T x + b = 0 \quad [1]$$

And the support vectors as:

$$w^T x + b = \pm 1 \quad [2]$$

where ±1 is the classification label, provided we think of our two classes as positive and negative examples. To further illustrate the idea, these equations imply that all points above the positive support vector have a value greater than 1 and all points below the negative support vector have a value of less than -1.

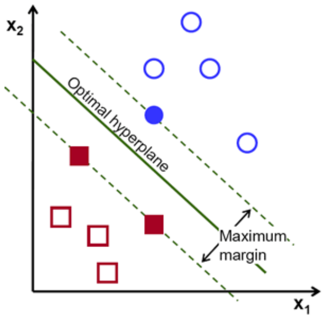


Fig. 2. Points on the Dotted Lines our are Support Vectors

In determining our decision boundary, we know that we desire the hyperplane with the maximum distance between the plane and the points closest to it.

If we take any point on the plane, x and compare it to a point closest to the plane, say x', we can find the distance between these by projecting their difference onto w. First we must normalize w, which we will refer to as $\hat{w}$. Thus we can denoted the distance between these two points as:

$$Distance = |\hat{w}^T(x' - x)| \quad [3]$$

Rewriting w as an un-normalized vector we can obtain:

$$Distance = \frac{1}{||w||}|w^T x' - w^T x|$$

Adding and subtracting the constant term b yields:

$$Distance = \frac{1}{||w||}|w^T x' + b - w^T x - b|$$

Inspecting this formula we see that the term subtracted, $w^T x + b$, is the value for a point on the plane, which due to the definition of the decision boundary, happens to be zero. The first term is the equation of a point closest to the plane, which must have an absolute value of 1. This nicely gives us:

$$Distance = \frac{1}{||w||}$$

Reserved for Publication Footnotes

Our goal is to maximize this distance. However, due to the nature of the formula this is not such an easy problem to solve. But, if we change this maximization to a minimization problem, it becomes much easier. So let's redefine our goal:

$$Minimize: \quad \frac{1}{2}||w||^2$$

Subject to the constraint:

$$\min_{n=1,2,\ldots,N} |w^T x_n + b = 1|$$

Why this constraint? Well, remember that we declared the support vectors are the points closest to the decision boundary and will always have an absolute value of 1. Given some linearly separable data, there must be atleast one point from each class which is closest to the decision boundary.

We can re-write this constraint as:

$$y_n(w^T x_n + b) \geq 1$$

Because y_n is the class label.

Now, anyone familiar with multi-variable calculus knows how to solve the problem of minimizing a function subject to constraints...LaGrange Multipliers.

Therefore, the function we want to minimize can be expressed using 3 variables: w, b, and α where α is the LaGrange multiplier. The equation we want to minimize can then be expressed as follows:

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i(y_i(w^T x_i + b) - 1) \qquad [4]$$

Finding the partial derivatives of the function, L we obtain:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{N} \alpha_i x_i y_i = 0 \qquad [5]$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0 \qquad [6]$$

Which imply:

$$w = \sum_{i=1}^{N} \alpha_i x_i y_i$$

And:

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

Using these results, we can re-write the LaGrangian in terms of just α, specifically:

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j x_i^T x_j \qquad [7]$$

While this may seem intimidating at first, keep in mind that every x and y is a known number or vector. Using quadratic programming, the extrema of this function can be found quite easily. The returned result is the row-vector α containing N entries. Interestingly enough, the vector will consist of mostly 0 entries. The reason being is due to the constraint:

$$\alpha_n(y_n(w^T x_n + b) - 1) = 0$$

Examine the constraint first while ignoring *alpha*. For our support vectors, we know that regardless of the value of *alpha*, the product will always be zero. For the internal points, that is, the points the lie away from the support vectors, the absolute value is always greater than 1. So in order for this constraint to be true, *alpha* must then be zero.

From here, finding out w and b is as simple as plugging in *alpha* for correct equations.

Our final hypothesis is:

$$g(x) = sign(w^T x + b) \qquad [8]$$

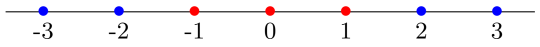
Given some new point x, we can determine the class membership be computing if the value of the above equation is positive or negative.

Kernel Methods

Now what if we ask the question: "What if the data isn't linearly separable?" Watch any lecture online about support vector machines and you will see they are so closely tied to another topic, Kernel Methods. Since often the data we encounter in real life comes with many features and is not linearly separable, kernel methods are a way attack such a problem.

They work by mapping the data from the original space into a higher dimension, such that in the higher dimensional space, the data becomes linearly separable. Let's first see an example of this mapping before we even begin to discuss the procedure.

Figure (1)

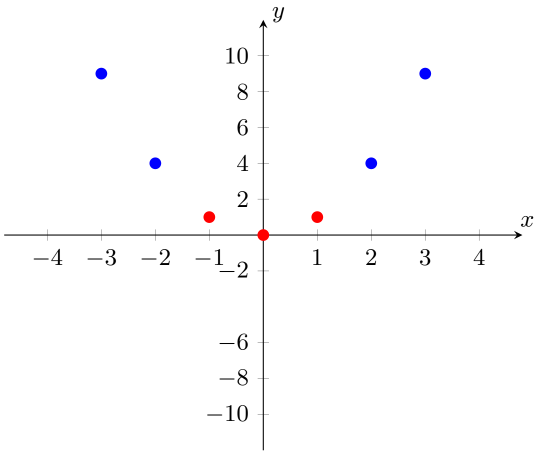


Examining the above figure, we see some 1 dimensional data set. If we wish to separate the blue and red dots, it is apparent there in no line that can accomplish this.

Now consider mapping these points into a 2 dimensional space such that:

$$x \mapsto (x, x^2)$$

The graph now becomes...



Which of course is linearly separable! Exactly what we wanted. The method can definitely work, but let's delve deeper into the process.

Think about the idea of having some data that is non-linearly separable in the x-y plane. If we desire to map this data to say 100,000 dimensions, the equations we want to minimize:

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

becomes:

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} \alpha_i \alpha_j y_i y_j z_i^T z_j$$

Notice the only change was the vectors x_i and x_j mapped to vectors z_i and z_j in the $Z-$Space. Also note, the operation between these two vectors is the inner product, returning just one real number. This implies that the dimensionality of the problem remains unchanged regardless of what dimension we map the original data to. The length of the vector *alpha* will always be of size N, where N is the total number of training examples. This result is of extreme importance, since we can go to any dimension we want, with no cost to us.

Keep in mind, when changing the dimensionality, w will correspond to the $Z-$ space, implying that the support vectors belong in the $Z-$space as well. Determining our support vectors with such high dimensionality is not a problem however. Remember that they correspond to the *alpha* values greater than zero.

So all we really need from the $Z-$Space is the inner product between the two vectors that live in that space. Examining the constraints for quadratic programming:

$$\alpha_n \geq 0 \quad for \quad i = 1, 2, \ldots, N$$

$$\sum_{i=1}^{N} \alpha_n y_n = 0$$

We notice these do not rely on any vector z, good news for us.

To find the inner product of these two vectors in the $Z-$Space though, we will need their coordinates. If we are mapping to a high dimensional space you can see this is a large demand that requires much computation, especially when we are dealing with a lot of data points.

So the question then becomes: Can we compute the inner product in the $Z-$Space without actually visiting the space?

Let's start off by assuming we are mapping a set of data points X such that, $x \in \mathbb{R}^n$ to some $p-$dimensional space. We can express this as :

$$\vec{x} = \phi(\vec{x}), \quad \forall x \in X$$

The inner product we would wish to compute then becomes:

$$\phi(\vec{x_i}^T) \cdot \phi(\vec{x_j})$$

Our goal is to show that this inner product can be evaluated in terms of the vectors in the original dimension n. This way, we avoid the process of mapping each data point. Now let us define the Kernel as a function of the two vectors x_i and x_j. Where:

$$K(x_i, x_j) = \phi(x_i^T) \cdot \phi(x_j) = z_i^T z_j$$

As an example, consider two, 2-dimensional vectors x and x' and a $2nd$ order polynomial transformation such that:

$$x = (x_1, x_2) \mapsto (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2) = z$$

The Kernel then becomes:

$$K(x, x') = z^T z' = 1 + x_1^2 x_1'^2 + x_2^2 x_2'^2 + x_1 x_1' + x_2 x_2' + x_1 x_2' x_1 x_2'$$

Now, let's consider the kernel:

$$K(x, x') = (1 + x^T x')^2$$

Notice although this involves an inner product it is still just a function. Taking the inner product of x and x' we get:

$$K(x, x') = (1 + x_1 x_1' + x_2 x_2')^2$$

Expanding we arrive at:

$$K(x, x') = 1 + x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_2' x_1 x_2'$$

Notice this term is very similar to the inner product in the $Z - Space$! In fact, if we just alter the mapping, such that:

$$x = (x_1, x_2) \mapsto (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2) = z$$

We find we get the same result! Thus, here is an example where we can get the inner product of two vectors in the $Z-$Space without actually visiting or knowing what the space is! This is what is called the kernel "trick". As you can see, it is extremely useful in reducing the computation needed for the problem.

The kernel function used here is an example of the polynomial kernel. Defined as :

$$K(x_i, x_j) = (1 + x_i^T x_j)^Q \qquad [9]$$

This idea of the polynomial kernel works for whatever dimension you wish. The inner product of two vectors in the $Z-$Space can be expressed as the inner product of two vectors in the $X-$Space. Thus, we have no need to know the space of the transform!

Other examples of popular kernels include:

$$K_{RBF}(\vec{u}, \vec{v}) = e^{(-\gamma||\vec{u}-\vec{v}||^2)} \qquad [10]$$

$$K_{Sig}(\vec{u}, \vec{v}) = tanh(\vec{u}\vec{v} + r) \qquad [11]$$

$$K_{Lin}(\vec{u}, \vec{v}) = \vec{u}\vec{v} \qquad [12]$$

Examining these kernels are outside the scope of this paper, but are important to the understanding of SVM.

Kernel Conditions

This brings on an interesting question, how do we know the kernel is valid? In other words, we need to be sure the kernel is equivalent to an inner product in the $Z-$Space.

There are some mathematical properties that can tell us if a kernel is valid or not. The two conditions are:

1. The Kernel is Symmetric.

2. The Matrix: $\begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \ldots & K(x_1, x_N) \\ K(x_2, x_1) & K(x_2, x_2) & \ldots & K(x_2, x_N) \\ \ldots & \ldots & \ldots & \ldots \\ K(x_N, x_1) & K(x_N, x_2) & \ldots & K(x_N, x_N) \end{bmatrix}$
is positive semi definite.

In other words:
1. $K(\vec{u}, \vec{v}) = K(\vec{v}, \vec{u})$
2. The above matrix must be greater than or equal to zero for all x_i where $i = 1, 2, 3, \ldots, N$ (Mercer's Condition).

Proving these two conditions for any kernel demonstrates that the $Z-$Space exists even if we do not know what it is.

Conclusion

Support Vector Machines are an important algorithm for everyone interested in machine learning to study. By studying SVM, one gains knowledge of kernel methods which are used in many more important algorithms. Some drawbacks of the SVM algorithm is that it is inherently a binary classier. Extensions must be made for multiple class cases which makes the procedure more difficult. The question of choosing the correct kernel also does not have a definitive answer. Cross-Validation is often helpful in this decision, but this idea was not explored in this paper.

1. Learning From Data. Web. 01 June 2016.
2. Hearst, Marti A., et al. "Support vector machines.” Intelligent Systems and their Applications, IEEE 13.4 (1998): 18-28.
3. Meyer, David, and FH Technikum Wien. "Support vector machines.” The Interface to libsvm in package e1071 (2015).
4. Moore, Andrew W. "Support vector machines.” Tutorial. School of Computer Science of the Carnegie Mellon University. Available at http://www. cs. cmu. edu/ awm/tutorials.[Accessed August 16, 2009] (2001).
5. Shawe-Taylor, John, and Nello Cristianini. Kernel methods for pattern analysis. Cambridge university press, 2004.