Support vector machines

DEPARTMENT OF ELECTRONICS AND ELECTRICAL COMMUNICATION ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Joshua Peter Ebenezer

October 4, 2018

Abstract

This white paper aims to give an explanation of how Support Vector machines work, and how their relevant parameters are derived. The material is all from Professor Mostafa's class at Caltech that is available on YouTube, and is meant to give a basic understanding of SVMs to anyone with a background in engineering and who has a basic knowledge of vector algebra and geometry.

Contents

1 Introduction

SVMs are used in the context of supervised machine learning. We assume that training data is available, comprising of examples that have a set of quantifiable features and a label associated with them. SVMs are used when the task is a binary labeling problem. Some of the 'desirable' properties that a machine learning algorithm must have are:

- It must have zero error on the training data.
- It must generalize well (i.e. perform well) on test data, which is data it has never seen before.
- It must be scalable (i.e. it should be able to handle very different sets of data (in terms of size or features) without much difficulty)

SVMs are high-dimensional linear classifiers. This does not mean that they are directly applied to the feature space, assuming the data to be linearly separable. Instead, a transform is applied to the feature space such that the data is represented in a high-dimensional space where the data *is* linearly separable. The SVM then is a hyper-plane that separates negative and positive examples in that high-dimensional space.

SVMs try to handle the concerns raised above, and do so by optimizing a particular quantity, the margin between the two classes. This is the distance from the hyperplane to the nearest data point on either side. It turns out that this quantity can be optimized at huge

2 Formulation

An SVM is a hyper-plane in a large dimensional space. Hence, it can be represented as

$$\mathbf{w}^T\mathbf{x} = 0 \tag{2.1}$$

where $\mathbf{x}_n$ is the vector of features, and $\mathbf{w}$ is a vector that is perpendicular to the plane, and hence defines it. Our objective is to find a $\mathbf{w}$ that obtains the greatest margin of separation between the classes. Let $\mathbf{x}_n$ be the point that is closest to the hyperplane. Finding the plane with the largest margin is equivalent to maximizing the distance between $\mathbf{x}_n$ and the plane. Before we begin quantifying this, we impose two conditions that do not affect the generality of the problem

1. Normalize $\mathbf{w}$ such that $|\mathbf{w}^T\mathbf{x}_n| = 1$. This does not affect the statement of the problem in any way because scaling $\mathbf{w}$ by any constant does not change the hyperplane equation. Hence, we can choose to scale the vector $\mathbf{w}$ such that the above condition is met.
2. Remove w_0, the first element of the vector $\mathbf{w}$, that accounts for the bias (and is multiplied with the homogeneous coordinate in the $\mathbf{x}$ vector). Replace it with

a constant bias term b that is not a part of the vector $\mathbf{w}$. The equation of the plane now becomes

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{2.2}$$

and the normalization condition is

$$|\mathbf{w}^T\mathbf{x}_n + b| = 1 \tag{2.3}$$

These technicalities will help in our treatment of the problem later on.

2.1 Finding the margin

Given an point $\mathbf{x}$ on the plane, the distance d from $\mathbf{x}_n$ to the plane will be the projection of the vector $\mathbf{x}_\mathrm{n} - \mathbf{x}$ on to the unit vector that is perpendicular to the plane, $\mathbf{w'} = \frac{\mathbf{w}}{||\mathbf{w}||}$, i.e.

$$\begin{aligned} d &= |\mathbf{w'}^T(\mathbf{x}_\mathrm{n} - \mathbf{x})| \\ &= \frac{|\mathbf{w}^T\mathbf{x}_\mathrm{n} - \mathbf{w}^T\mathbf{x})|}{||\mathbf{w}||} \\ &= \frac{|\mathbf{w}^T\mathbf{x}_\mathrm{n} + b - \mathbf{w}^T\mathbf{x} - b|}{||\mathbf{w}||} \\ &\overset{1}{=} \frac{|\mathbf{w}^T\mathbf{x}_\mathrm{n} + b|}{||\mathbf{w}||} \\ &\overset{2}{=} \frac{1}{||\mathbf{w}||} \end{aligned} \tag{2.4}$$

where 1 follows from that fact that $\mathbf{x}$ lies on the plane, and so $\mathbf{w}^T\mathbf{x} + b = 0$, and 2 follows from the normalization condition.

2.2 Maximizing the margin

So now our task boils down to maximizing

$$d = \frac{1}{||\mathbf{w}||} \tag{2.5}$$

subject to $\min_{1,2..N} |\mathbf{w}^T\mathbf{x}_n + b| = 1$. This doesn't look too bad to deal with, but the min might be problematic, and so can the modulus operator. There are some things we can do to simplify our job. Particularly, notice that

$$|\mathbf{w}^T\mathbf{x}_n + b| = y_n(\mathbf{w}^T\mathbf{x}_n + b) \tag{2.6}$$

because for any point for which $\mathbf{w}^T\mathbf{x}_n + b > 0$, we classify that point is a positive training example and hence y_n is $+1$ because our error on training data is 0, and similarly for any point for which $\mathbf{w}^T\mathbf{x}_n + b < 0$, y_n is -1.

Now we can rewrite our optimization objective as

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ such that $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$ for all n for all $n = 1, 2..N$.

This can be solved using the method of Lagrangian multipliers. However, since the constraint is a set of inequalities and not equalities, we impose the Karush-Kuhn-Tucker (KKT) conditions, which we will not go into here.

So we write the problem as
Minimize

$$\mathbf{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N} \alpha_n(y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) \tag{2.7}$$

w.r.t. $\mathbf{w}$ and b, and maximize w.r.t. each $\alpha_n \geq 0$, where α_n are the Lagrangian multipliers.

Note that when we deal with equalities, we simply take the gradient of the Lagrangian w.r.t. to the variables and set them to 0, but with the constraint that $\alpha_n \geq 0$, it is possible that at the values for which the gradient w.r.t. α is 0, the constraint is not met. However, it turns out that by setting the gradients w.r.t. $\mathbf{w}$ and b to zero, we obtain a quadratic form in α, that can be solved by a standard quadratic programming package. Taking the gradients and setting them to zero,

$$\nabla_{\mathbf{w}}\mathbf{L} = \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = 0 \tag{2.8}$$

$$\frac{\partial \mathbf{L}}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n = 0 \tag{2.9}$$

Substituting these in the Lagrangian, we get

$$\begin{aligned}\mathbf{L}(\alpha) &= \frac{1}{2}\sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^T \sum_{m=1}^{N} \alpha_m y_m \mathbf{x}_m - \sum_{n=1}^{N} \alpha_n (y_n (\sum_{m=1}^{N} \alpha_m y_m \mathbf{x}_m^T)\mathbf{x}_n + b) - 1) \\ &= \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m\end{aligned} \tag{2.10}$$

and the optimization objective become: maximize $\mathbf{L}(\alpha)$ w.r.t. α , subject to $\alpha_n \geq 0$ for all $n = 1, 2..N$. Note that the only variable here is α, since the other quantities are known (as training data).

The form above is a standard quadratic optimization problem, and can be solved by a quadratic programming package. Notice that $\mathbf{x}_n^T\mathbf{x}_m$ is the inner product of two feature vectors, but is a scalar (just one value). Hence, no matter how large the feature space is, the inner product will cause it to boil down to a single number in the computation. The outer summations are over the training points, not the features.

2.3 What does the result look like?

The KKT condition states that

$$\alpha_n(y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) = 0 \tag{2.11}$$

Hence, for all interior points, $\alpha_n = 0$. For points on the border of the margin, $y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1 = 0$. It is only for such points that is possible for α_n to be positive. So the solution is completely determined by the 'support vectors', the points that define the margin by virtue of being on the boundary between the margin and the rest of the plane. Those points 'support' the plane. Having found α, we find

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n \tag{2.12}$$

and we use $y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$ for any of the support vectors in order to find the value of b. This determines the plane completely.

2.4 The Kernel trick

At the beginning, we had made the assumption that our input features are linearly separable. What if they are not? We could perhaps transform the inputs to a high dimensional space where they *are* linearly separable. The naive way to compute the inner product required in the optimization objective would be to go to the high dimensional space, calculate the inner product by brute force, get a value, and return. The computation might get intensive when the number of dimensions is large. Are there ways to compute this inner product without actually visiting the high dimensional space? It turns out there are.

The problem is as such: we need to find a high dimensional feature space in which our original input data is linearly separable. After that, we need to find the inner product of our transformed inputs *in that high dimensional space* and feed it to the quadratic programmer package that will give us the values of α.

Consider the function K of two 2-D vectors

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= (1 + \mathbf{x}\mathbf{x}')^2 \\ &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + {x_1}^2 {x_1'}^2 + {x_2}^2 {x_2'}^2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_1' x_2 x_2' \end{aligned} \tag{2.13}$$

which is the inner product of the two vectors when transformed to the space $(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$. Such functions, that operates on the input space to generate the inner product of higher dimensional transformations of the inputs, are called kernels. Similarly, the kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||)^2) \tag{2.14}$$

can also be shown to have the same property. Take the 1-D case, for which

$$\begin{aligned} K(x, x') &= \exp(-(x - x)'^2) \\ &= \exp(-x^2)\exp(-x'^2)\exp(2xx') \\ &= \exp(-x^2)\exp(-x'^2)\sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!} \end{aligned} \tag{2.15}$$

which is again an inner product of two separable vectors that are functions of x and x'. Interestingly, in this case, the basis of the feature space in which the inner product is calculated in has infinite dimensions. But this does not matter, as long as the inner product exists, and we see how we can calculate that quantity without actually going to the infinite-dimension space.

How do we know if K is a valid kernel for which there actually exists a high dimensional space whose inner product is given by K? This is answered by *Mercer's condition*, which states that for K to be a valid kernel, the following two conditions must hold:

1. K is symmetric.
2. The matrix
$$\begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \dots & K(x_1,x_n) \\ K(x_2,x_1) & K(x_2,x_2) & \dots & K(x_2,x_n) \\ \dots & \dots & \dots & \dots \\ K(x_n,x_1) & \dots & \dots & K(x_n,x_n) \end{bmatrix}$$
is positive semi-definite.

Now we can rewrite our optimization objective as

$$\mathbf{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m K(\mathbf{x}_n, \mathbf{x}_m) \tag{2.16}$$

Nothing changes except for replacing the inner product. We have been able to generate the inner product for very large dimensional spaces without actually visiting those spaces, by functions applied to the input features directly.

2.5 Soft margins

What if we are willing to allow some amount of 'slack'? Instead of a hard margin, perhaps if we let some training points lie *inside* the margin, we could get a better generalization to test data. In such cases, we use a soft margin SVM. The optimization function is now

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{M} \xi_n$ such that $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0$ for all n for all $n = 1, 2..N$.

The Lagrange formulation is now
Minimize

$$\mathbf{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{M} \xi_n - \sum_{n=1}^{N} \alpha_n (y_n(\mathbf{w}^T - \mathbf{x}_n + b) - 1) - \sum_{n=1}^{M} \beta_n \xi_n \tag{2.17}$$

w.r.t. $\mathbf{w}$, b, and ξ and maximize w.r.t. each $\alpha_n \geq 0$, $\beta_n \geq 0$, where α_n and β_n are the Lagrangian multipliers. C acts as a regularization parameter. Notice that if $C \to \infty$, we get the hard margin SVM, because ξ would be 0 in order to minimize the expression.

If C is very small, it would be acceptable for the optimizer to violate the margin by large distances and do so many times.

Just like before, taking the gradients and setting them to zero,

$$\nabla_{\mathbf{w}}\mathbf{L} = \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = 0 \tag{2.18}$$

$$\frac{\partial \mathbf{L}}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n = 0 \tag{2.19}$$

$$\frac{\partial \mathbf{L}}{\partial \xi_n} = C - \alpha_n - \beta_n = 0 \tag{2.20}$$

If we substitute the third equation into the Lagrangian, we find that the ξ terms vanish. We end up with the same expression as before, with the only difference being an additional upper constraint on α.

$$\mathbf{L}(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m \tag{2.21}$$

The objective is to minimize $\mathbf{L}(\alpha)$ subject to $C \geq \alpha_n \geq 0$ for all $n = 1, 2..N$. This is again quite a simple objective for a quadratic programming package, and we can get back α easily.

Appendices

A A bit of historical trivia

Support vector machines are used in a wide variety of machine learning applications. Before deep learning came into vogue in 2012 after Alexnet's stunning performance in image classification, SVMs ruled the roost for a long time, from the 1990s. Other methods were seen as inferior in performance, and probably for good reason. SVMs just worked better, and had the very important advantage of being built on solid mathematical foundations. It is said that one of the firm believers in the deep learning paradigm, Yann LeCun, used to heckle people at conferences in those days when kernels and SVMs were the only methods taken very seriously, because deep learning was sidelined at that time by the larger community. Of course, after the deep learning revolution, LeCun and others had their 'I told you so' moment.

Nevertheless, SVMs pushed the boundaries of the field of pattern recognition over the years, and continue to be used in many applications. There are still many areas (eg: mass detection in mammograms) in which automated deep learning methods have not been very successful yet, and where SVMs still give excellent results. Especially when 'big data' is hard to find, SVMs are often the preferred choice of engineers.