Support Vector Machine Study Summary (I): Intuition

(From http://billyinn.wordpress.com/2014/10/28/support-vector-machine-study-summary-i-intuition/)

1, Introduction

Fine to skip this section if willing.

As generally known, SVM may be the most powerful and widespread classifier nowadays. However, due to the complicated mathematics inside and some convient libraries like LIBSVM, most of us including me just use it as a blackbox tool without knowing how it works.

I am just a beginner in statistical machine learning, and use SVM as a omnipotent classifier. Whenever I encounter a classifying problem, the first thing I do is to throw the data into SVM and see how the performance is. But the curiosity makes me want to make clear of what the SVM exactly is.

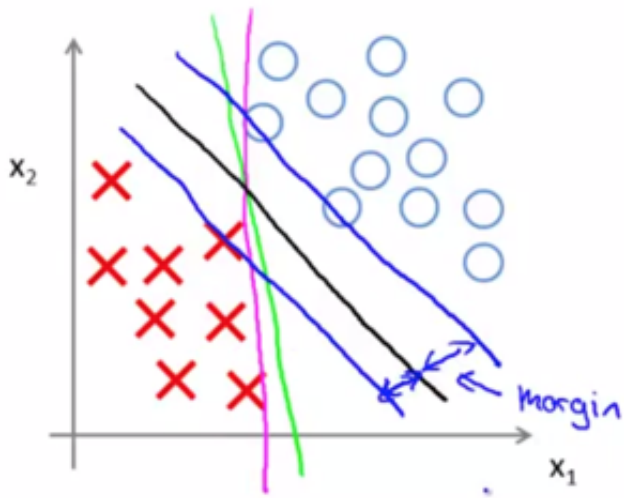
So, I plan to write a series of study summaries of SVM. The preliminary plan is to write three summaries including intuition, mathematical deduction and implementation. Also I want to write an advanced topic about parallelizing SVM on GPU hardware and achieveing better performance if possible.

2, Linear SVM

Intuition

I've read several books' chapters and articles about SVM. All of them first introduce the *Linear SVM*. For it's easy to understand the essence of SVM, I also start with it.

What is *Linear SVM*? It's not hard to understand, just as the two dimensional picture illustrated (the pic is captured from Andrew Ng's Machine Learing Courses), we can seperate the two classes with a linear boundary. And the *Linear SVM* can give the seperating line when the classes are separable by a linear boundary, otherwise the *Linear SVM* fails.



However when the classes are seperable by a linear boundary, there are infinite feasible linear boundaries. How *Linear SVM* find the desired boundary and what metric it use to determine it? On one side, we can establish a metric considering all the examples and try to optimize it; on the other side, it's also reasonable to find a linear boundary which makes the closest example as far as possible. Just as the picture above illstruated, the black line is better than the green and the purple line from our intuition. And in my opinion, it's the basic difference between logistic regression and SVM. Logistic regression use the first metric and SVM use the second. As a result, SVM classifier is also called as *large margin classifier*.

Definition

Now, it's time to introduce some basic definition to generalize the case to the high dimension which is not that easy to image.

Supposing that there are N training examples $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where $x_i \in \mathcal{X} = \mathrm{R}^n, y \in \mathcal{Y} = \{+1, -1\}, i = 1, 2, \dots, N$. x_i is the ith feature vector; y_i is the x_i's label. If $y_i = +1$, x_i is a positive example; otherwise, x_i is a negative example.

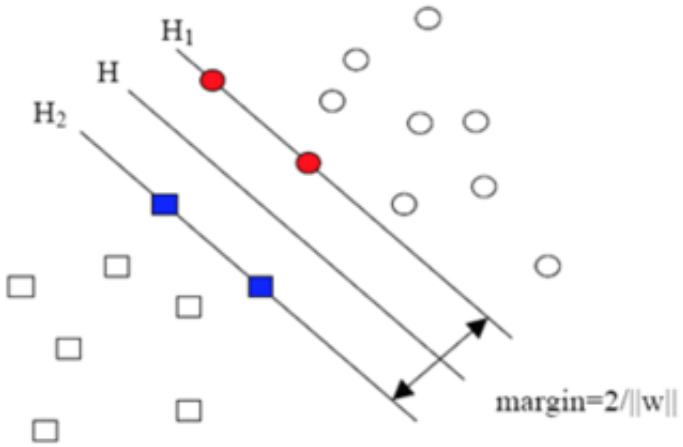
When the dimension is larger than 2, the boundary would not be a line any more. We now call it as *hyperplane* in general. The *hyperplane* can be represented as $w * x + b = 0$. And then here is the judge function $g(x) = w \cdot x + b$ and the classify function $f(x) = sgn(w \cdot x + b)$. If you are still not familiar with the notation *sgn*, see more details on this page.

Then it's time to introduce the metric we use to measure the performance of this hyperplane. In general, the distance between the example and the hyperplane, namely $|w \cdot x + b|$, can represent the certainty of the prediction of the classifier. Notice that whether the sign of $w \cdot x + b$ is equal to the sign of y can determine the correctness of the prediction, so we can use $y(w \cdot x + b)$ can represent the correctness and certainty of the classification which we call it as *functional margin*. It seems really nice to be the measurement.

> **Definition**(Functional Margin)
> Given the training examples set T and hyperplane (w, b), define the functional margin of example (x_i, y_i) as $\hat{\gamma}_i = y_i(w \cdot x_i + b)$, and define the functional margin of all examples in T as $\hat{\gamma} = \min\limits_{i=1,\dots,N} \hat{\gamma}_i$.

However, the *functional margin* is not enough. If we change w and b in scale, our *functional margin* will also change with exactly the same hyperplane. This fact tells us that we need to give some constraints to *w*. In practice, we normalize that $||w|| = 1$ so that the margin can be determined which we call it as *geometric margin*, where $||w||$ is w's norm. See more details about norm on this page.

> **Definition**(Geometric Margin)
> Given the training examples set T and hyperplane (w, b), define the geometric margin of example (x_i, y_i) as $\gamma_i = y_i(\frac{w}{||w||} \cdot x_i + \frac{b}{||w||})$, and define the geometric margin of all examples in T as $\gamma = \min\limits_{i=1,\dots,N} \gamma_i$.



From the picture above(from the posts of SVM on blogjava.net), we can understand the concept of geometric margin more clearly. H is the hyperplane, H1 and H2 are parallel to H and go through the cloest examples, and the distance between H1 and H1 is the geometric margin.

One thing should be told is that the fact, the wider the margin the better the hyperplane, is not only from our intuition but also can be proved theoretically. There is a relationship between the times of predicting failure and geometric margin: $TimesOfFailure \leq (\frac{2R}{\gamma})^2$, where $R = \max\limits_{i=1,\dots,N} \|x_i\|$. Now, we're sure that the training objective is to maximize the geometric margin.

Objective

We now know the objective from our intuition, but we also need to state our objective more formally and make it more clear:

$$\begin{aligned} &\max_{w,b} \gamma \\ &s.t. \quad y_i\left(\frac{w}{\|w\|} \cdot x_i + \frac{b}{\|w\|}\right) \geq \gamma, \quad i = 1, 2, \dots, N \end{aligned}$$

Note that $\gamma = \frac{\hat{\gamma}}{\|w\|}$, the geometric margin γ is inversely proportional to $\|w\|$. Therefore, instead of maximizing γ, we would rather minimize $\|w\|$ with $\hat{\gamma}$ fixed to 1. Why we can do this? Actually, the value of functional margin make no difference to the solution our problem, because we can change w and b in scale to change $\hat{\gamma}$ at the same time. Then it produces an equivalent objective.

To make the objective easier to handle in mathematics, we need to transform the objective a little bit. Obviously,$\min\|w\|$ is equivalent to $\min \frac{1}{2}\|w\|^2$. As a result, the objective transformed to the form:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 \\ &s.t. \quad y_i(w \cdot x_i + b) - 1 \geq 0 \quad i = 1, 2, \dots, N \end{aligned}$$

This a problem called *convex quadratic programming* in mathematics. I will explore deeper into it in my next summary.

Here I don't want to talk too much about the mathematical deduction here. But the problem has some

really nice properties:

> **Theorem**(Existence and Uniqueness of the Maximum-Margin Hyperplane)
> If the training example set T is linear separable, the maximum-margin hyperplane which can separate the examples in the set correctly exists and is unique.

Now, the objective is clear enough. Just skip how to reach the objective a little while, and first consider the non-separable cases.

3, Slack Variables

In practice, the classes are often not fortunate enough to be linear separable completely, especially there are just some outliers in the examples. And then, all we mention above don't work any more. How we can extend the linear situation to the non-linear situation? Maybe we can just drop those outliers and the performance may even not be affected. However, the machines cannot recognize these outliers themselves, so we introduce a slack variable $\xi_i \geq 0$ to each example (x_i, y_i).

Then the constraint condition change to the form:

> $$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad i = 1, 2, \dots, N$$

The introduction of slack variables can help us get larger margin and smoother boundary at the expense of the accurate prediction of some examples. When we want to drop some examples, we need to pay costs for them. So we add a term to the objective function:

> $$\frac{1}{2} \|w\| + C \sum_{i=1}^{N} \xi_i$$

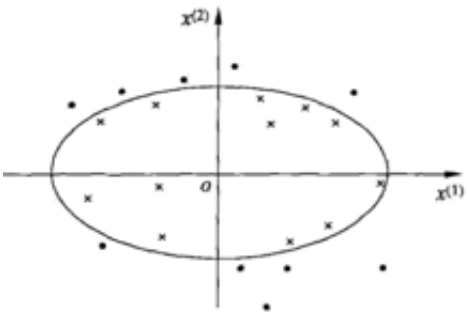
C is the cost parameter. The larger C will be, the more cost we will pay for dropping the examples. When C becomes infinite large, we go back to the situation of linear separable cases.

There is several things to be noted:

- In actual, only ontliers have the slack variables. Most examples' slack variables are equal to zero.
- The slack variables represent how far the example is away from the predicted class group.
- The cost parameter C is a constant and won't change with our optimization which will be discussed later.
- Though, the form of the objective seems to be changed, it can be solved with the similar method just like the linear separable cases.

4, Kernel Trick

However, sometimces the two classes cannot be separable thoroughly, just like the picture below. Fortunately, the kernel trick can solve the problem well and in a quite awesome way in my perspective.



Transform the Objective

Two parameters w, b determine the hyperplane, so the objective is to find w, b. In actual, if we know w, we can calculate b too. Why? Because $w \cdot x + b = 0$ and we already w, x, it's easy to get b.

b can be determined by w, which determine w? Obviously, the examples (x, y) (these are all what we have). And the hyperplane is linear so far, so we can state the relationship between w and x:

> $$w = \alpha_1 y_1 x_1 + \alpha_2 y_2 x_2 + \cdots + \alpha_N y_N x_N$$

Now, we can see α_i as normal parameters, which is called as *Lagrange Multipliers* in mathematics. I will explore deeper into it in my next summary, if interested, see more details on this page. Don't forget w, x here are all n-dimensional vectors here.

Actually, most α is zero, only those examples lie on the boundary have values other than zero. And the final hyperplane is determined by these examples and has nothing to do with other examples. These determining examples are called *Support Vectors*, and it's how *Support Vector Machine* comes from.

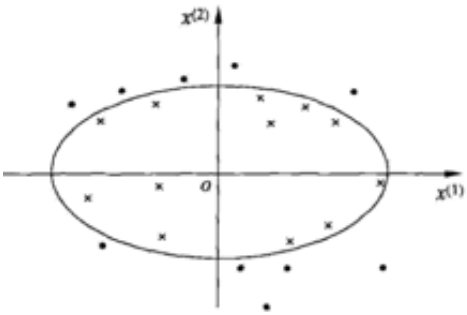
We can now have a more clear form:

> $$w = \sum_{i=1}^{N} a_i y_i x_i$$
> $$g(x) = \sum_{i=1}^{N} a_i y_i x_i \cdot x + b$$

Then there is no w in the $g(x)$. We can now introduce the concept of the kernel and generalize our problem to the non-linear-separable cases.

Kernel Trick Intuition

All we discuss before is about linear SVM and we definitely don't want to give up it all. If there are some methods that can transform the non-linear cases to the linear cases? Sure, it's why we introduce the kernel trick!



Just as the picture above, we cannot separate the two classes with the linear model $w_1x_1 + w_2x_2 + b = 0$ but we can set a mapping from the lower dimension to the higher dimension. We define a mapping from the origin space to the new space $z = ((x_1^2, x_2^2))^T$, then the points in the origin space are transformed to the corresponding points in the new space.

The ellipse in the origin space $w_1x_1^2 + w_2x_2^2 + b = 0$ now transform to the line in the new space $w_1z_1 + w_2z_2 + b = 0$. Wonderful, isn't it? Now we can seperate the classes with our linear model again! In mathematics, the origin space is called *Euclidean Space* and the new space is called *Hilbert Space*, something in algebra that I haven't figure out thoroughly.

But how to find such mappings? Unfortunately, there seems to be no systematic methods. We need to guess and try ourselves. However, don't worry about that, people have found several powerful kernel functions and these functions may be enough for us.

Kernel Function

Suppose that we have examples $x_i, i = 1, 2, \dots N$ with n dimensions in non-linear cases, we map them to x_i' with $2n$ dimensions so that they can be separated linearly.

Note that the mapping just involves with x and has nothing to do with y or α. We can represent the $g(x')$ in another form:

$$g(x') = \sum_{i=1}^{N} a_i y_i < x_i', x' > +b$$

where $< \cdot >$ means the dot product of two vectors, see more details about dot product on this page.

$g(x')$ is the high dimension linear function. If we substitute $< x_i', x' >$ with a function of x, it becomes exactly the lower dimension form:

$$g(x) = \sum_{i=1}^{N} a_i y_i K(x_i, x) + b$$

And with α, y_i and b unchanged. Then we can solve this problem with the linear model, and all we have to do is just substitute the dot product with the kernel function.

The last problem: which functions can be kernel functions? The problem is nothing mathematics. Actually, if a function satisfies the *mercer condition*, it can be a kernel function. I don't want to discuss it further, if interested, see more details on this page.

There all lots of kernel functions listed on the web pages and textbooks, but the choose of kernel functions still doesn't have general prinicles. In practice, the most often used kernel function is Gaussian kernel function:

> $$K(x, z) = exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$