Lab 3 SVM (Support Vector Machines)

Introduction

SVM was introduced around 50 years ago, they have evolved over time and have also been adapted to various other problems like regression, outlier analysis, and ranking.

In this lab manual, I'll review linear classifier at first with hard math, and then show the essential parts of SVM and develop a strong intuition of the working algorithms while skipping as much of the math as possible. In the end, some useful libraries and resources are introduced to get you started.

Maximum Margin Classifier

Before talking about the details of SVM, let's start with *linear classifier*. Now we have a set of **n** dimensions vector ***x***, and there two classes noted as ***y***, and ***y*** can be 1 or -1 for convenience. A linear classifier is to find a hyperplane, who has the form of

$$w^T x + b = 0$$

so that it can separate these vectors into two sides, in one of them all the vectors are 1, and in the other side, all the vectors are -1. Let $f(x) = w^T x + b = 0$, obviously, if $f(x) = 0$, means x is on this hyperplane. WLOG, for all the points satisfies $f(x) < 0$, the corresponding y equals to -1, meanwhile, $f(x) > 0$, indicates that y is 1. As we can see from Fig. 1,

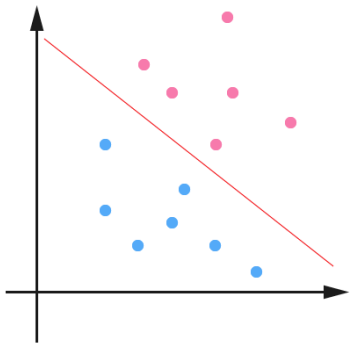

Fig. 1

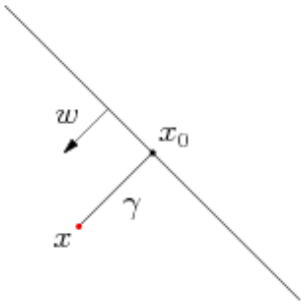

Fig. 2

In fact, when $|f(x)|$ is very small, little vibration will change the result. Ideally, we hope every point can have $|f(x)|$ big enough to distinguish itself.

Therefore, we first define *functional margin* as,

$$\hat{\gamma} = y(w^T x + b) = yf(x)$$

multiply y is to make sure functional margin is non-negative. And the distance between point and hyperplane γ is *geometrical margin* as shown in Fig. 2. Now we want the derive the relationship between *functional margin* and *geometrical margin*. Suppose we have a point x, its projection on hyperplane is x_0, and as w is a vector perpendicular to hyperplane. We have,

$$x = x_0 + \gamma \frac{w}{||w||}$$

because x_0 is on hyperplane, so $f(x_0)$=0, substitute into hyperplane function, we have,

$$\gamma = \frac{w^T x + b}{||w||} = \frac{f(x)}{||w||}$$

like $\hat{\gamma}$, we also add a y to make sure its non-negativity. So, we have *geometrical margin* as,

$$\tilde{\gamma} = y\gamma = \frac{\hat{\gamma}}{||w||}$$

We can see that the only difference of functional margin and geometrical margin is a constant $||w||$. From previous analysis, a larger margin can guarantee a higher confidence of classification. Considering if we multiply any constant to (w, b), the hyperplane won't change, but $f(x)$ will change, and leads to the change of $\hat{\gamma}$, which is the situation we don't want to see. Therefore, we use geometrical margin as the target function of maximum margin classifier, and we assign the minimal geometrical margin of all the point as the margin of the dataset. We want this margin as big as possible, so that the classifier has better performance.

$$\max \tilde{\gamma}$$

and according to the definition of margin, this maximum also needs to satisfy the condition of

$$y_i(w^T x_i + b) = \hat{\gamma}_i > \hat{\gamma}, \qquad i = 1, \dots, n$$

As we discussed, $\hat{\gamma}$ will change with $||w||$, which is irrelevant to the hyperplane we find. So, we let $\hat{\gamma} = 1$, and the objective function becomes,

$$\max \frac{1}{||w||}, s.t. y_i(w^T x_i + b) = \hat{\gamma}_i > \hat{\gamma}, i = 1, \dots, n$$

Solve this problem, we can find a classifier which can maximize the margin. As shown in Fig. 3, the red line is the *optimal hyperplane,* and its distance between the blue line (or the pink one) is $\hat{\gamma}$.

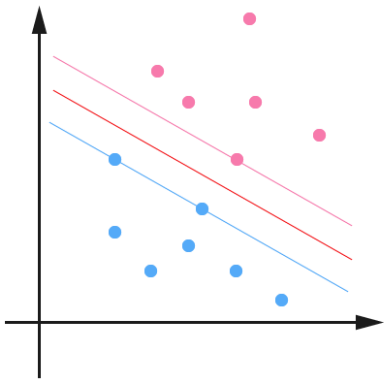

Fig. 3

Support Vector

Until now, we can bring out *support vector*. Let's see the other example in Fig. 4.

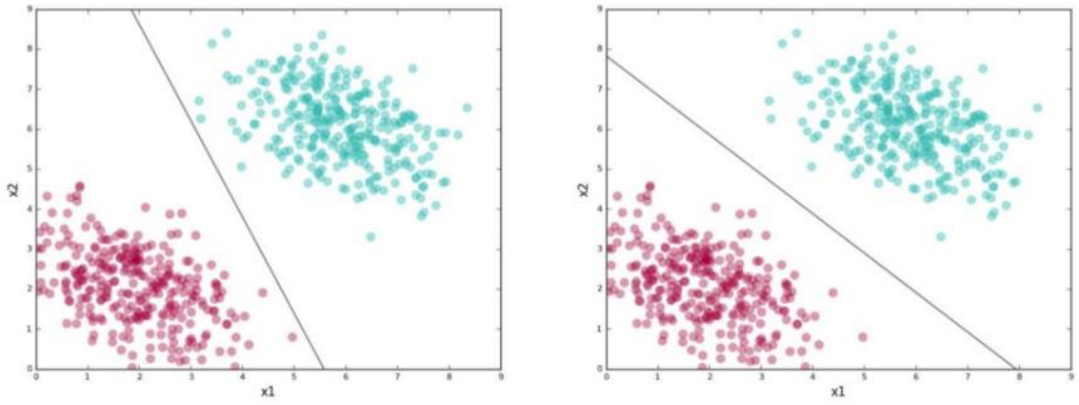

Fig. 4

In this case, it seems like we can find more than one line that can separate the red and green clusters. Remember that the worth of a classifier is not how well it separates the training data, but we want it to classify unseen data, which is called *test data*. Given that, we want to choose a good line (hyperplane) that captures the general pattern in the training data. In general, here is a simple version of what SVMs do without hard math:

1. Find planes that correctly classify the training data
2. Among all such planes, pick the one that has the greatest distance to the points closest to it

Recall what we have in last section, these closest points that identify this line are known as *support vectors*. SVMs give you a way to pick between many possible classifiers in a way that guarantees a higher chance of correctly labeling test data.

Outliers

We have looked at an easy case of perfectly linearly separable data in the last section. Real-world data is messy. Always we have a few instances that a linear classifier can't get right. Here is an example in Fig. 5.

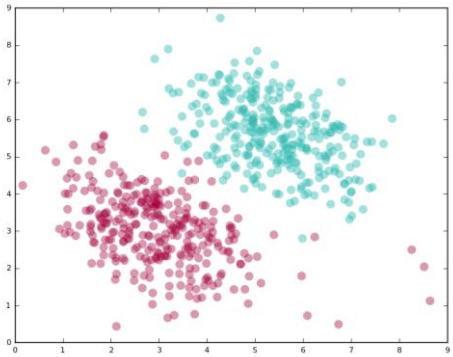

Fig. 5

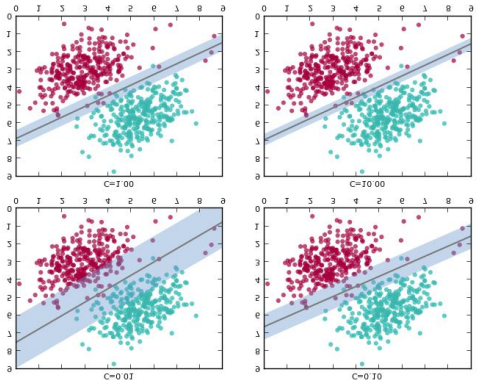

Fig. 6

SVMs solve these problems by adding a constant term, to specify how many errors you are willing to accept. Generally, the rule is

As the C(onstant) increases, the width of the margin will shrink.

A simulation can be seen in Fig. 6.

***Lagrange Duality (optional)**

However, objective function is mentioned, but how to get the optimal solution is not mentioned. Since it's a quite mathematical problem, I won't introduce in detail. Instead, I'll give you the essential equations of solving this problem.

Let's recall what the objective function looks like

$$\max \frac{1}{||w||}, \text{s.t.} \, y_i(w^T x_i + b) = \hat{\gamma}_i > \hat{\gamma}, i = 1, \dots, n$$

To make the problem easier, we change it to an equivalent form

$$\min \frac{1}{2}||w||^2, \text{s.t.} \, y_i(w^T x_i + b) = \hat{\gamma}_i > \hat{\gamma}, i = 1, \dots, n$$

Obviously, this is a convex optimization problem. We can use many existing *Quadratic Programming* library to solve it. But here we use Lagrange Duality to solve it which is more efficient. First, we construct Lagrange function with objective function and condition

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

And let

$$\theta(w, b) = \max_{\alpha_i > 0} L(w, b, a)$$

Easy to prove, if the condition is met, minimize $\frac{1}{2}||w||^2$ is equivalent to minimize $\theta(w, b)$. Therefore, the objective function becomes

$$\min_{w,b} \theta(w, b) = \min_{w,b} \max_{\alpha_i} L(w, b, a) = p$$

where p is the optimal result. Again, for the purpose of easing the problem, we

swap the min and max, and use d to represent the new optimal result. These two problems are not equivalent, and $d<p$, which at least give us a lower bound of p. The reason why we do this transformation you can refer to the textbook. To solve the new problem, we take partial derivative of L, and let them to 0.

$$\frac{\partial \mathrm{y}}{\partial \mathrm{x}} = 0 \rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

Substitute back to L, we have

$$L(w, b, \alpha) = \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

The problem becomes an α duality variable optimization

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j,$$

$$s.t. \alpha_i \geq 0, i = 1, \dots, n, and \sum_{i=1}^{n} \alpha_i y_i = 0$$

Before we move to next section, we take a look at the hyperplane function again, and remember what we have derived for w.

$$f(x) = \left(\sum_{i=1}^{n} \alpha_i y_i x_i\right) x + b = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + b$$

where $\langle x_i, x \rangle$ is inner product. This form tells us, to predict the new point (which side it belongs to), what we do is just do inner product with the training points.

Kernel

We have seen how SVM systematically handle perfectly linearly separable data. How about the cases where the data is not linearly separable? After all, most of the real-world data falls in this category.

Here is an example of non-linearly separable data shown in Fig. 7. What should we do about it? We try to project the data into a space where it is linearly separable and find a hyperplane in this space.

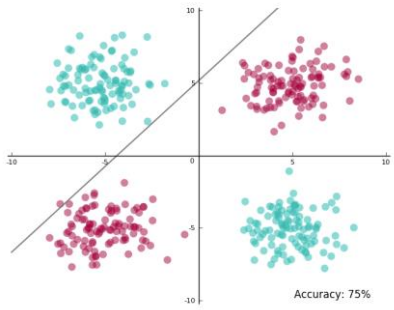

Fig. 7

We start with the dataset in Fig. 5, and project it into a 3-dimensional space where the new coordinates are:

$$X_1 = x_1^2, X_2 = x_2^2, X_3 = \sqrt{2}x_1x_2$$

Now even I didn't tell you how to separate the data points, you can tell it can be roughly separated with a plane.

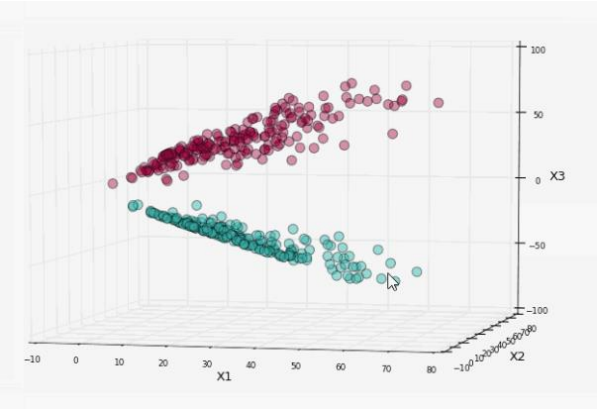

Fig. 8

Let's run SVM on it,

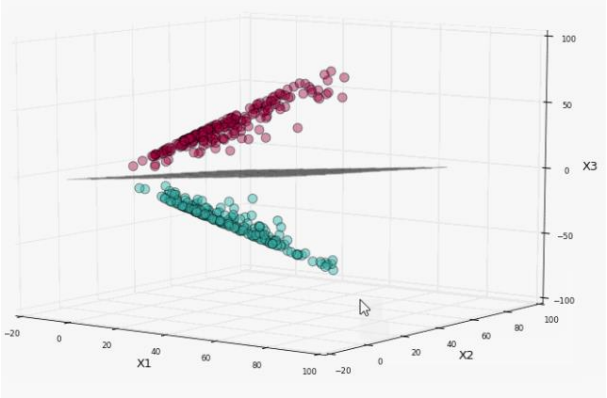

Fig. 9

Let's project the plane back to the original 2-dimensional space and see what the hyperplane becomes.

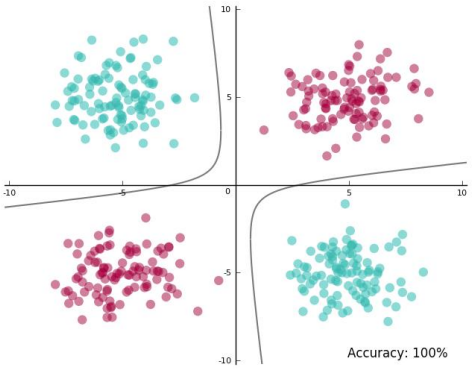

Fig. 10

Looks exactly what we want.

The first thing you may notice is that, in this example, the dataset is simple,

we can easily find the new basis based. What if for an arbitrary function, how can I know what space to project the data onto? Honestly, it's hard to know, however, what we know is data is more likely to be linearly separable when projected onto higher dimensions, thanks to *Cover's Theorem*. Sometimes we even project data onto infinite dimensions and that often works unexpectedly well.

The other thing you may ask is should we project the data first and then run the SVM. The answer is NO. Again, the given example is just a specific case. The fact is we will ask SVM to do the projection for us. Here it's time to bring out *kernel*.

A very surprising aspect of SVM is that in all of the mathematic machinery it uses, the projection doesn't show (mathematically called implicit). We just write all of it in inner product form like we mentioned in the end of last section. This gives us some hints that we don't need to provide the SVM with exact projections, instead we give it the inner product between all pairs of points in the projection space. And this is what kernel does.

Let's revisit the projection we did before, and see if we can come up with corresponding kernel. We will also calculate the number of computations (complexity) we need for the projection, and compare to inner product.

For a point I,

$$\overrightarrow{x_i} = (x_{i1}, x_{i2})$$

The corresponding projected point was

$$\overrightarrow{X_i} = (x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2})$$

The dot product in the new dimension is:

$$X_i X_j = X_{i1}X_{j1} + X_{i2}X_{j2} + X_{i3}X_{j3}$$

So that's 11 multiplications and 2 additions.

Now if we already know this kernel function will give me the same result:

$$K(x_i, x_j) = (x_i, x_j)^2$$

We need only 3 multiplications and 1 addition, which is faster than projection first. This little change will matter a lot when we encounter large dimension data. So that's one huge advantage of using kernels.

Some libraries

There are quite a few SVM libraries you could start with: libSVM, SVM-Light, SVMTorch. My recommendation is to start out and tried libSVM.

libSVM is available as a command line tool, but the download also bundles Python, Java, and MATLAB wrappers. As long as you have a file with your data in a format libSVM understands, you are good to go.

Exercise #1

Generate the training data yourselves as Fig. 7, and do SVM to find the hyperplane. In the report, list the details of how the data changes, and attach corresponding picture.

Exercise #2

Generate the training data yourselves as roughly locates around two concentric circles, and find the hyperplane using SVM. Same requirement for the report in #1.

Duality

$$a_1 x_1^2 + a_1 x_2^2 + a_3 x_1 x_2 + a_4 x_1 + a_5 x_2 + a_6 = 0$$

$$\sum_{i=1}^{5} a_i y_i + a_6 = 0$$

$$y_i(w^T x_i + b) = \hat{\gamma}_i > \hat{\gamma}$$

$$\min \frac{1}{2} ||w||^2$$

$$\max_{\alpha_i} \min_{w,b} L(w, b, a) = d$$