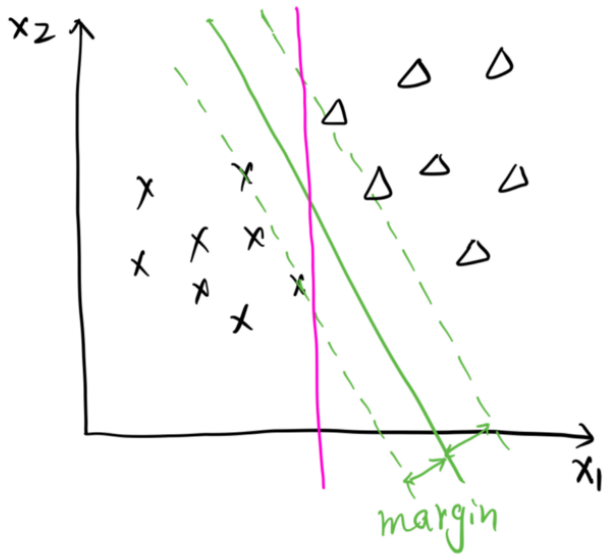
Loss Function(Part III): Support Vector Machine


Shuyu Luo Oct 16, 2018 · 8 min read

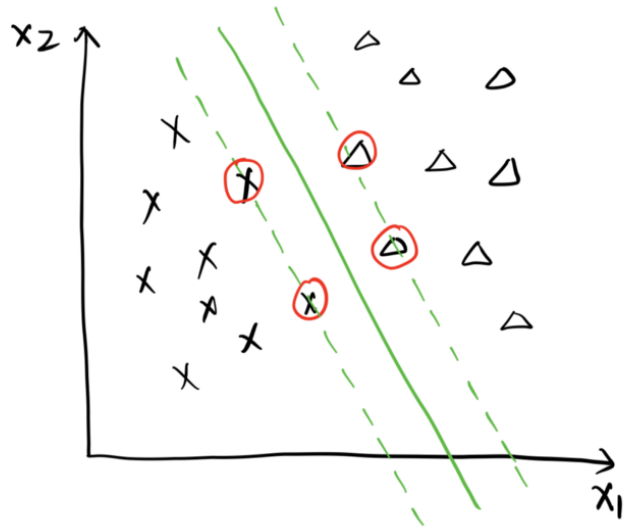
Continuing this journey, I have discussed the loss function and optimization process of linear regression at Part I, logistic regression at part II, and this time, we are heading to Support Vector Machine.

Linear SVM

Let's start from Linear SVM that is known as SVM without kernels. Looking at the scatter plot by two features X1, X2 as below. We actually separate two classes in many different ways, the pink line and green line are two of them. SVM ends up choosing the green line as the decision boundary, because how SVM classify samples is to find the decision boundary with the largest margin that is the largest distance from a sample who is closest to decision boundary. That's why Linear SVM is also called **Large Margin Classifier**.

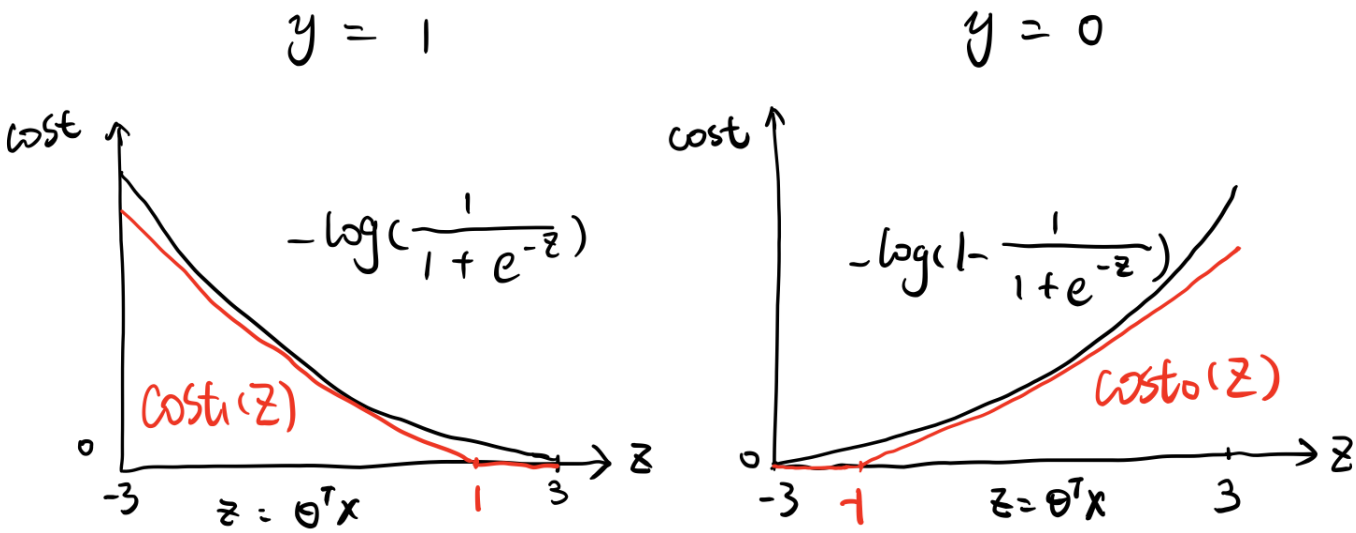


Who are the support vectors? Support vector is a sample that is incorrectly classified or a sample close to a boundary. Looking at the plot below. The samples with red circles are exactly decision boundary. In SVM, only support vectors has an effective impact on model training, that is saying removing non support vector has no effect on the model at all. Why? We will figure it out from its cost function.



The loss function of SVM is very similar to that of Logistic Regression. Looking at it by $y = 1$ and $y = 0$ separately in below plot, the black line is the cost function of Logistic Regression, and the red line is for SVM. Please note that the X axis here is the raw model output, $\theta^T x$. Remember putting

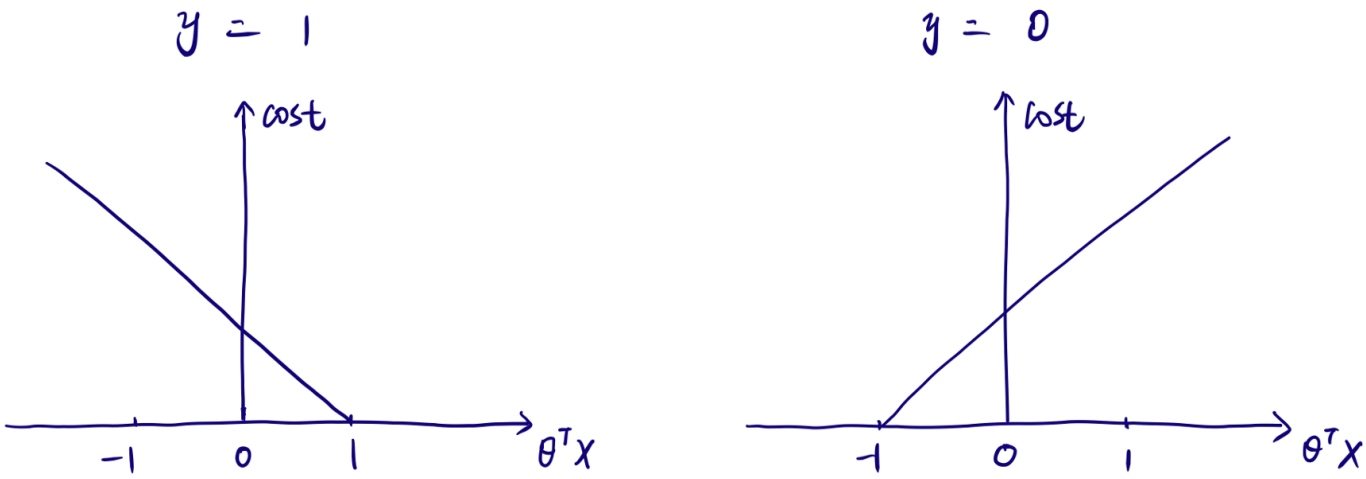
the raw model output into Sigmoid Function gives us the Logistic Regression's hypothesis. What is the hypothesis for SVM? It's simple and straightforward. When $\theta^T x \geq 0$, predict 1, otherwise, predict 0.



SVM Hypothesis:

$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x >= 0 \\ 0 & \text{otherwise} \end{cases}$$

Then back to loss function plot, aka. **Hinge Loss**, when the actual is 1 (left plot as below), if $\theta^T x \geq 1$, no cost at all, if $\theta^T x < 1$, the cost increases as the value of $\theta^T x$ decreases. Wait! When $\theta^T x \geq 0$, we already predict 1, which is the correct prediction. Why does the cost start to increase from 1 instead of 0? Yes, SVM gives some punishment to both incorrect predictions and those close to decision boundary ($0 < \theta^T x < 1$), that's how we call them support vectors. When data points are just right on the margin, $\theta^T x = 1$, when data points are between decision boundary and margin, $0 < \theta^T x < 1$. I will explain why some data points appear inside of margin later. As for why removing non-support vectors won't affect model performance, we are able to answer it now. Remember model fitting process is to minimize the cost function. Since there is no cost for non-support vectors at all, the total value of cost function won't be changed by adding or removing them.



Let's write the formula for SVM's cost function:

$$Cost(h_\theta(x), y) = \begin{cases} max(0, 1 - \theta^T x) & \text{if } y = 1 \\ max(0, 1 + \theta^T x) & \text{if } y = 0 \end{cases}$$

$$J(\theta) = \sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T(x^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T(x^{(i)})$$

$$J(\theta) = \sum_{i=1}^{m} y^{(i)} max(0, 1 - \theta^T x) + (1 - y^{(i)}) max(0, 1 + \theta^T x)$$

$$m = number\ of\ samples$$

We can also add regularization to SVM. For example, adding L2 regularized term to SVM, the cost function changed to:

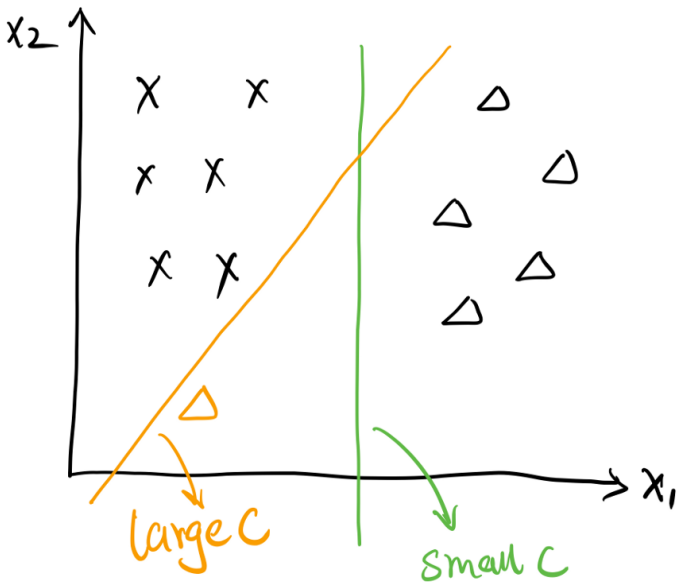
$$J(\theta) = C[\sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T(x^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T(x^{(i)})] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

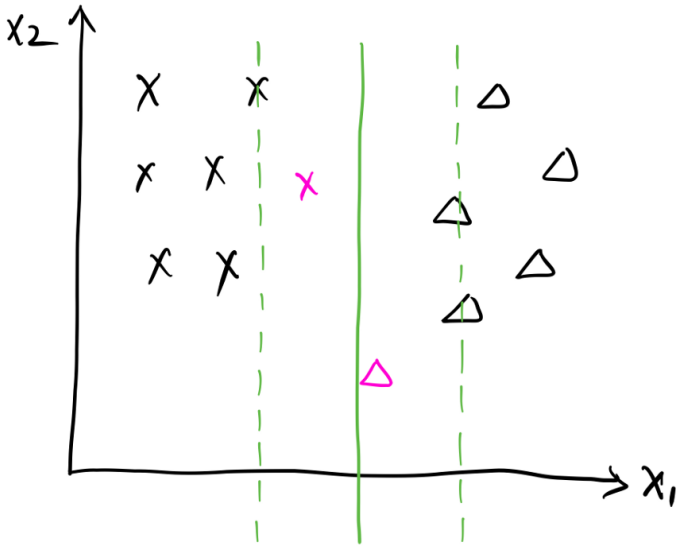
$$m = number\ of\ samples, \quad n = number\ of\ features$$

Different from Logistic Regression using **λ** as the parameter in front of regularized term to control the weight of regularization, correspondingly, SVM uses **C** in front of fit term. Intuitively, the fit term emphasizes fit the model very well by finding optimal coefficients, and the regularized term controls the complexity of the model by constraining the large value of coefficients. There is a trade-off between fitting the model well on training dataset and the complexity of the model that may lead to overfitting, which can be adjusted by tweaking the value of λ or C. Both λ and C prioritize how

much we care about optimize fit term and regularized term. Placing at different places of cost function, C actually plays a role similar to $1/\lambda$.

With a very large value of C (similar to no regularization), this large margin classifier will be very sensitive to outliers. For example, in the plot on the left as below, the ideal decision boundary should be like green line, by adding the orange orange triangle (outlier), with a vey big C, the decision boundary will shift to the orange line to satisfy the the rule of large margin. On the other hand, C also plays a role to adjust the width of margin which enables margin violation. See the plot below on the right. When C is small, the margin is wider shown as green line. The pink data points have violated the margin. It is especially useful when dealing with non-separable dataset. So This is how regularization impact the choice of decision boundary that make the algorithm work for non-linearly separable dataset with tolerance of data points who are misclassified or have margin violation.





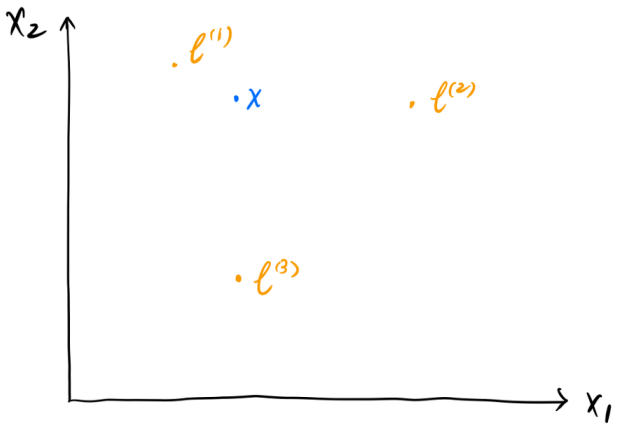
Non-Linear SVM

When decision boundary is not linear, the structure of hypothesis and cost function stay the same. Firstly, let's take a look.

$$Hypothesis: \quad h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T f >= 0 \\ 0 & \text{otherwise} \end{cases}$$

$$Cost\ Function: \quad J(\theta) = C[\sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T(f^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T(f^{(i)})]$$

You may have noticed that non-linear SVM's hypothesis and cost function are almost the same as linear SVM, except 'x' is replaced by 'f' here. f is the function of x, and I will discuss how to find the f next. Let's tart from the very first beginning. Assume that we have one sample (see the plot below) with two features x1, x2. I randomly put a few points ($l^{(1)}$, $l^{(2)}$, $l^{(3)}$) around x, and called them landmarks. I would like to see how close x is to these landmarks respectively, which is noted as f1 = Similarity(x, $l^{(1)}$) or k(x, $l^{(1)}$), f2 = Similarity(x, $l^{(2)}$) or k(x, $l^{(2)}$), f3 = Similarity(x, $l^{(3)}$) or k(x, $l^{(3)}$).



So this is called **Kernel Function**, and it's exact 'f' that you have seen from above formula. What is it inside of the Kernel Function? In other words, how should we describe x's proximity to landmarks? There are different types. **Gaussian Kernel** is one of the most popular ones. It's calculated with Euclidean Distance of two vectors and parameter σ that describes the smoothness of the function. Gaussian kernel provides a good intuition. If x ≈ $l^{(1)}$, f1 ≈ 1, if x is far from $l^{(1)}$, f1 ≈ 0. In Scikit-learn SVM package, Gaussian Kernel is mapped to 'rbf' , **Radial Basis Function Kernel**, the only difference is 'rbf' uses γ to represent Gaussian's $1/2\sigma^2$.

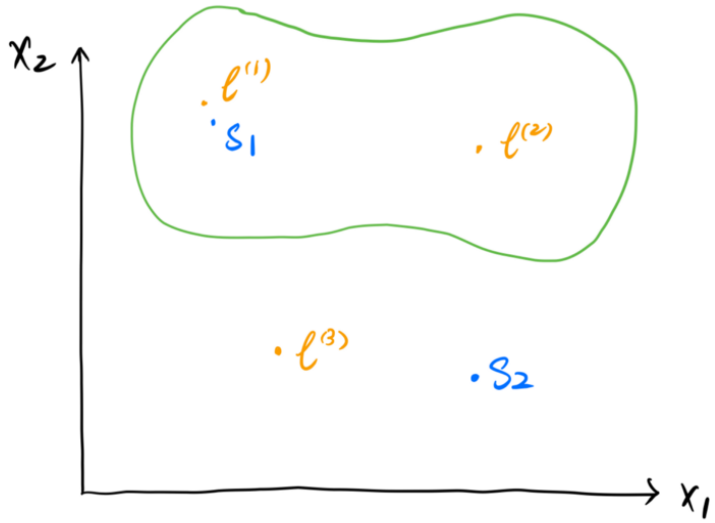
$$f_1 = Similarity(x, l^{(1)}) = exp(\frac{||x - l^{(1)}||^2}{2\sigma^2})$$

$$f_2 = Similarity(x, l^{(2)}) = exp(\frac{||x - l^{(2)}||^2}{2\sigma^2})$$

$$f_3 = Similarity(x, l^{(3)}) = exp(\frac{||x - l^{(3)}||^2}{2\sigma^2})$$

We can say that the position of sample x has been re-defined by those three kernels. That is saying, Non-Linear SVM computes new features f1, f2, f3, depending on the proximity to landmarks, instead of using x1, x2 as features any more, and that is decided by the chosen landmarks. This is where the raw model output $\theta^T f$ is coming from. Let's try a simple example. $\theta^T f = \theta 0 + \theta 1 f1 + \theta 2 f2 + \theta 3 f3$. Assign $\theta 0 = -0.5$, $\theta 1 = \theta 2 = 1$, $\theta 3 = 0$, so the $\theta^T f$ turns out to be -0.5 + f1 + f2. Looking at the first sample(S1) which

is very close to $l^{(1)}$ and far from $l^{(2)}$, $l^{(3)}$, with Gaussian kernel, we got f1 = 1, f2 = 0, f3 = 0, $\theta^T f = 0.5$. According to hypothesis mentioned before, predict 1. Sample 2(S2) is far from all of landmarks, we got f1 = f2 = f3 =0, $\theta^T f = -0.5 < 0$, predict 0. Based on current θs, it's easy to notice that any point near to $l^{(1)}$ or $l^{(2)}$ will be predicted as 1, otherwise 0. The green line demonstrates an approximate decision boundary as below.



We have just went through the prediction part with certain features and coefficients that I manually chose. So, where are these landmarks coming from? How many landmarks do we need? Ok, it might surprise you that given m training samples, the location of landmarks is exactly the location of your m training samples.

$$Given\ (x^{(1)}, y^{(2)}), (x^{(1)}, y^{(2)}), \ldots\ldots (x^{(m)}, y^{(m)})$$

$$Choose\ l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots\ldots l^{(m)} = x^{(m)}$$

$$m\ =\ number\ of\ samples$$

That is saying Non-Linear SVM recreates the features by comparing each of your training sample with all other training samples. Thus the number of features for prediction created by landmarks is the the size of training samples. For a given sample, we have updated features as below:

$$Given\ the\ i^{th}\ sample\ x^{(i)}\ :$$

$$f_1^{(i)} = k(x^{(i)}, l^{(1)})$$

$$f_2^{(i)} = k(x^{(i)}, l^{(2)})$$

$$\ldots\ldots$$

$$f_i^{(i)} = k(x^{(i)}, l^{(i)})$$

$$\ldots\ldots$$

$$f_m^{(i)} = k(x^{(i)}, l^{(m)})$$

$$where\ x^{(i)} = l^{(i)},\ f_i^{(i)} = 1$$

Regarding to recreating features, this concept is like that when creating a polynomial regression to reach a non-linear effect, we can add some new features by making some transformations to existing features such as square it. For example, you have two features x1 and x2. To create polynomial regression, you created $\theta 0 + \theta 1x1 + \theta 2x2 + \theta 3x1^2 + \theta 4x1^2x2$, as so your features become $f1 = x1$, $f2 = x2$, $f3 = x1^2$, $f4 = x1^2x2$

Let's rewrite the hypothesis, cost function, and cost function with regularization.

$$Hypothesis: \quad h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T f >= 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\theta^T f = \theta_0 f_0 + \theta_1 f_1 + \ldots + \theta_m f_m$$

$$Cost\ Function: \quad J(\theta) = C[\sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T(f^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T(f^{(i)})]$$

$$Regularized\ Cost\ Function: \quad J(\theta) = C[\sum_{i=1}^{m} [y^{(i)} Cost_1(\theta^T(f^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T(f^{(i)})] + \frac{1}{2}\sum_{j=1}^{m} \theta_j^2$$

To achieve a good performance of model and prevent overfitting, besides picking a proper value of regularized term C, we can also adjust σ^2 from Gaussian Kernel to find the balance between bias and variance. Take a certain sample x and certain landmark l as an example, when σ^2 is very large, the output of kernel function f is close 1, as σ^2 getting smaller, f moves towards to 0. In other words, with a fixed distance between x and l, a big σ^2 regards it 'closer' which has higher bias and lower variance(underfitting),while a small σ^2 regards it 'further' which has lower bias and higher variance (overfitting).

Like Logistic Regression, SVM's cost function is convex as well. The most popular optimization algorithm for SVM is **Sequential Minimal Optimization** that can be implemented by 'libsvm' package in python**.** SMO solves a large quadratic programming(QP) problem by breaking them into a series of small QP problems that can be solved analytically to avoid time-consuming process to some degree. In terms of detailed calculations, It's pretty complicated and contains many numerical computing tricks that

makes computations much more efficient to handle very large training datasets.

In summary, if you have large amount of features, probably Linear SVM or Logistic Regression might be a choice. If you have small number of features (under 1000) and not too large size of training samples, SVM with Gaussian Kernel might work for you data well .

Machine Learning | Optimization | Support Vector Machine | Classification