Kernel SVM (draft)

Abstract

Kernel method is a way that turns non-linear separable problem into linearly separable problem by mapping low dimensional feature space to a high dimensional feature space. It is used in many algorithms, including the kernel perceptron, support vector machines (SVM), Gaussian processes, principal components analysis (PCA), canonical correlation analysis, ridge regression, spectral clustering, linear adaptive filters and many others. The advantage of kernel method is the cheaper computational expense comparing to computing coordinates of the data explicitly. In following sections we will introduce how to use kernel tricks in SVM.

1. What is kernel SVM

2.1 Feature Space Mapping

2.2 How kernel tricks work

2.3 Dual Language formulation for kernels

2.4 common kernels

2.5 Mercer’s Theorem

2.6 Compare different kernels

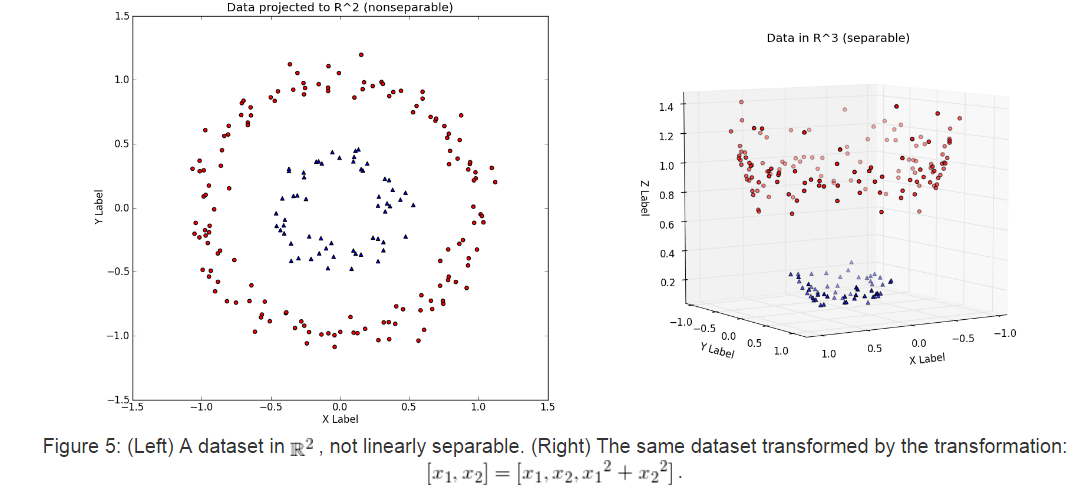
3. Reference

**1. What is kernel SVM**

Kernel SVM uses kernel trick to deal with non-linear separable data set using SVM. Basically, a kernel function http://latex.codecogs.com/gif.latex?\inline%20K(\vec%7bv%7d,\vec%7bw%7d) is a function http://latex.codecogs.com/gif.latex?\inline%20K:%20\mathbb%7bR%7d%5eN%20\times%20\mathbb%7bR%7d%5eN%20\rightarrow%20\mathbb%7bR%7d that that computes a **dot product** between http://latex.codecogs.com/gif.latex?\inline%20\vec%7bv%7d , http://latex.codecogs.com/gif.latex?\inline%20\vec%7bw%7d ,which also measures the **similarity** between vectors. [Why? for example, u⋅v=|u||v|cosθ, the dot product equals to 0 if cosθ is 0, i.e. u and v are orthogonal and not similar. The normalized dot product equals to 1 or -1 if θ is 0, which means u and v are similar. The can also extends to higher dimensions]. Because of **feature space mapping**, we are able to find a hyperplane that can separate non-linearly separable date. However, the computational complexity of feature space mapping is generally very high. The kernel method shows that we can use only the dot products in some feature space without knowing the explicit coordinates in F and even the mapping function. We will consider **Dual Language formulation** to show it.

**2.1 Feature Space Mapping**

Now we will use examples to see why mapping lower dimensional data to higher dimension can make data linearly separable. Consider a very simple two-class data

 Eric Kim <http://www.eric-kim.net>

Obviously we are unable to find a line to separate the two classes in the left figure. But we can imagine that drawing a circle in the gap of the two classes will be sufficient. Mathematically, the “radius” of the circle formed by blue dots and the red dots are different. [recall that a circle centered in (0,0) has the radius x^2+y^2 = r^2]. Here we can see that calculating x1^2+x2^2 will give different answer by red dots and blue dots. As the figure above on the right shows, mapping the two dimensional data into three dimension – adding the x1^2+x2^2 in the z direction, makes the data linearly separable with a x-y plane placed in two classes.

However, for data with large dimension, it is impractical to map it to higher dimension. Consider the computational consequences of increasing the dimensionality from http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5eN to http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5eM (with http://latex.codecogs.com/gif.latex?\inline%20M%20%3E%20N ). If http://latex.codecogs.com/gif.latex?\inline%20M grows very quickly with respect to http://latex.codecogs.com/gif.latex?\inline%20N(e.g. http://latex.codecogs.com/gif.latex?\inline%20M%20\in%20O(2%5eN) ), then learning SVMs via dataset transformations will incur serious computational and memory problems!

Here is a concrete example: the Polynomial Kernel is a kernel often used with SVMs. For a dataset in http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5e2 , a two-degree polynomial kernel (implicitly) performs the transformation http://latex.codecogs.com/gif.latex?\inline%20%5bx_1,%20x_2%5d%20=%20%5b%7bx_1%7d%5e2,%20%7bx_2%7d%5e2,%20\sqrt%7b2%7d%20\cdot%20x_1%20\cdot%20x_2,%20\sqrt%7b2%20\cdot%20c%7d%20\cdot%20x_1,%20\sqrt%7b2%20\cdot%20c%7d%20\cdot%20x_2,%20c%5d . This transformation adds three additional dimensions http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5e2 -> http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5e5 . In general, a d-dimensional polynomial kernel maps from http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5eN to an http://latex.codecogs.com/gif.latex?\inline%20%7bN+d%20\choose%20d%7d -dimensional space [[6]](http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html#[6]). Thus, for datasets with large dimensionality, naively performing such a transformation will quickly become intractable.

**2.2 How kernel tricks work**

It turns out that the SVM has no need to explicitly work in the higher-dimensional space at training or testing time. One can show that during training, the optimization problem only uses the training examples to compute **pair-wise** dot products http://latex.codecogs.com/gif.latex?\inline%20\langle%20\vec%7bx_i%7d,%20\vec%7bx_j%7d%20\rangle , where http://latex.codecogs.com/gif.latex?\inline%20\vec%7bx_i%7d,%20\vec%7bx_j%7d%20\in%20\mathbb%7bR%7d%5eN .

Why is this significant? It turns out that there exist functions that, given two vectors v and w in http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5eN , implicitly computes the dot product between v and win a higher-dimensional

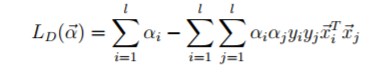
http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5eM  **without explicitly transforming v and w to**http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5eM. Such functions are

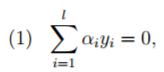
called **kernel** functions, http://latex.codecogs.com/gif.latex?\inline%20K(\vec%7bv%7d,%20\vec%7bw%7d) . The implications are:

1. By using a kernel http://latex.codecogs.com/gif.latex?\inline%20K(\vec%7bx_i%7d,%20\vec%7bx_j%7d) , we can implicitly transform datasets to a higher-dimensional http://latex.codecogs.com/gif.latex?\inline%20\mathbb%7bR%7d%5eM using no extra memory, and with a minimal effect on computation time.
   * The only effect on computation is the extra time required to compute http://latex.codecogs.com/gif.latex?\inline%20K(\vec%7bx_i%7d,%20\vec%7bx_j%7d) . Depending on http://latex.codecogs.com/gif.latex?\inline%20K , this can be minimal.
2. By virtue of (1), we can efficiently learn nonlinear decision boundaries for SVMs simply by **replacing all dot products in the SVM computation with**http://latex.codecogs.com/gif.latex?\inline%20K(\vec%7bx_i%7d,%20\vec%7bx_j%7d)**!**

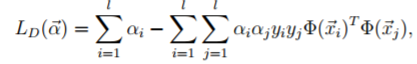
This is basically the idea of kernel trick. Next section explains why only dot product is necessary in a mathematical way.

**2.3 Dual Language formulation for kernels**

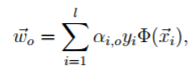
Recall that the dual problem for linear SVM is to maximize LD

Subject to , 

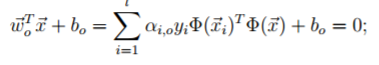
For kernal SVM, with x mapped to Φ(x), the equation becomes



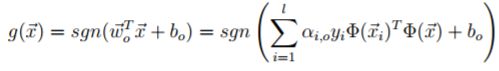
with the optimal weight



and the optimal plane

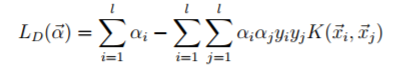


and the optimal decision function

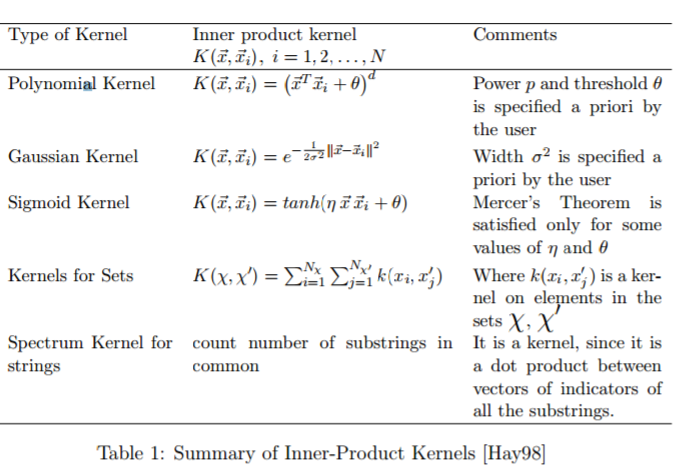


From above equations we can see, the weight vector of the optimal hyperplane in F can be represented only by data points. Note also the optimal hyperplane and decision function only depend on the mapped data through dot products in some feature space F. The explicit coordinates in F and even the mapping function Φ become unnecessary when we define a function K(xi , x) = Φ(xi) TΦ(x), the so called kernel function, which directly calculates the value of the dot product of the mapped data points in some feature space.

Thus, using Kernel function, we can write the equation of dual formulation as



**2.4 Commen Kernels**



The above table shows some common kernels. Although some kernels are domain specific there is in general no best choice. Since each kernel has some degree of variability in practise there is nothing else for it but to experiment with different kernels and adjust their parameters via model search to minimize the error on a test set. Generally, a low polynomial kernel or a Gaussian kernel have shown to be a good initial try and to outperform conventional classifiers

**2.5 Mercer’s Theorem**

Mercer’s Theorem determines a similarity function that can satisfy to be a kernel function. In mathematics, specifically functional analysis, Mercer's theorem is a representation of a symmetric positive-definite function on a square as a sum of a convergent sequence of product functions. Thus, the function must be symmetric and satisfy the inequalities that follow from the Cauchy-Schwarz inequality. From Mercer’s theorem, that a matrix is **a Gram Matrix,** if and only if it is positive and semi-definite, i.e. it is an inner product matrix in some space [CST00]. Hence, a Gram Matrix fuses all information necessary for the learning algorithm, the data points and the mapping function merged into the inner product.

Nevertheless, it is noteworthy, that Mercer’s theorem only tells us when a candidate kernel is an inner product kernel, and therefore admissible for use in support vector machines. However it tells nothing about how good such a function is. Consider for example a diagonal matrix, which of course satisfies Mercer’s conditions but is not quite good as a Gram Matrix since it represents orthogonal input data and therefore self-similarity dominates between-sample similarity.

**2.6 Compare different kernels**

In real life we will need to decide which kernel to use. The most common candidates are linear SVM, RBF kernel and polynomial with different degrees. The simplest way is to calculate the accuracy using all models with default setting, and then use cross validations to fine-tune optimal parameters.

Theoretically, this is a over-fitting vs. under-fitting problem. We define the simplification of bound as Test Error ≤ Training Error + Complexity of set of Models. Actually, a lot of bounds of this form have been proved (different measures of capacity). The complexity function is often called a regularizer. If you take a high capacity set of functions (explain a lot) you get low training error. But you might over-fit. If you take a very simple set of models, you have low complexity, but won’t get low training error.

Here is some advises on when to choose linear model and when to choose RBF kernel. First, you should use linear kernel when number of features is larger than number of observations. Second, you might use Gaussian kernel when number of observations is larger than number of features. Third, if number of observations is larger than 50,000 speed could be an issue when using Gaussian kernel; hence, one might want to use linear kernel

reference

<https://metacademy.org/graphs/concepts/kernel_svm#focus=svm_optimality_conditions&mode=learn>

<http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html>

<http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/slides/SupportVectorMachines.pdf>

http://www.cogsys.wiai.uni-bamberg.de/teaching/ss06/hs\_svm/slides/SVM\_Seminarbericht\_Hofmann.pdf