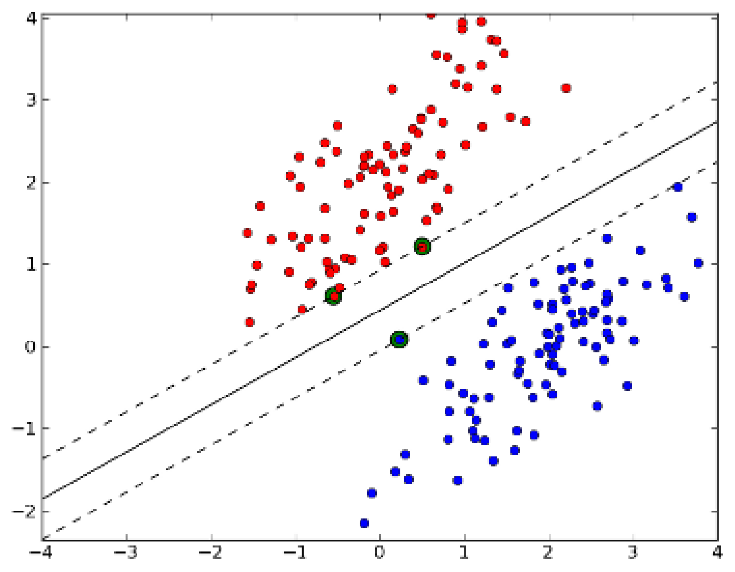
Understanding the kernel trick.

Harish Kandan Follow
Aug 30, 2017 · 4 min read

I've observed that just like me, a lot of us who try to learn about support vector machines find it difficult to comprehend the brilliance of kernels. It took me quite some time and a lot of resources but I've finally crossed the river and I intend to help you folks do that too. It may seem confusing in the middle but with a little bit of patience and persistence, I'm pretty sure that you'd have a good idea about it by the end. So well, let's get started now!

For those of you unfamiliar with SVM, here's a brief introduction.
In data classification problems, SVM can be used to it provides the maximum separating margin for a linearly separable dataset. That is, of all possible decision boundaries that could be chosen to separate the dataset for classification, it chooses the decision boundary which is the most distant from the points nearest to the said decision boundary from both classes. That ought to have been a little confusing so here's a figure to help you get an intuition.



There are two classes here, red and blue. The line in the middle is the decision boundary. The highlighted points are the support vectors. The distance between all of these points and the line is the maximum possible among all possible decision boundaries, thus making this the most optimal one.

Isn’t that good now? It is, except for the fact that this is applicable; in it’s raw form; only for linearly separable data. What if it isn’t? Enter kernels. The idea is that our data, which isn’t linearly separable in our ’n’ dimensional space may be linearly separable in a higher dimensional space. To understand how kernels work, some math would be necessary so brace yourselves!

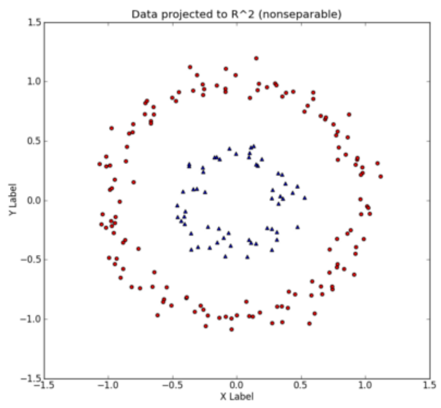
Lets say the decision boundary. i.e the hyperplane separating the classes have the weights(Co-efficients) given by vector w. This is what we need to find. We try to maximize the distance from this w vector to the nearest points(support vectors) so this now becomes our constraint. Saving you a bit of math which goes behind this, I’ll have to ask you here to take it for a given that to reach the solution, we solve the following

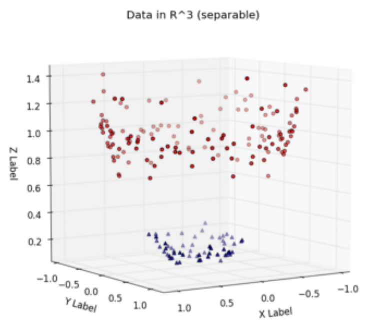
$$\text{minimize:}\quad W(\alpha) = -\sum_{i=1}^{\ell}\alpha_i + \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} y_i y_j \alpha_i \alpha_j \mathbf{x}_i \mathbf{x}_j$$

$$\text{subject to:}\quad \sum_{\substack{i=1 \\ 0\le\alpha_i\le C}}^{\ell} y_i\alpha_i = 0 \qquad (4)$$

I’ll give you a brief explanation of the scary-looking-yet-simple-natured equation. l is the number of data points in our training data. y’s denote the outputs of the data points, which for our convenience, we express as +1 or -1. x is the feature vector in each training example. Alpha now…… Well, alpha is the Lagrangian constant. To give a brief overview about this, Lagrangian multipliers are used to include the constraints for solving a minimization or maximization problems, thus enabling us to not worry about them while reaching our solution. Anyway, while minimizing for W(alpha) [Weight vector as a function of alpha] we see the term x*x(transpose). That is to say that we do not exactly need the exact data points, but only their inner products to compute our decision boundary. Big deal, right? So what if this is the case? How does this help?
What it implies is that if we want transform our existing data into a higher dimensional data, which in many cases help us classify better(see the image below for an example) we need not compute the exact transformation of our data, we just need the inner product of our

data in that higher dimensional space. This works like a charm in datasets which aren't linearly separable!





It's a lot easier to get the inner product in a higher dimensional space than the actual points in a higher dimensional space. Polynomial kernels by simply using exponents of 'd' to map our data into a 'd' dimensional space can be effective for our solution. Gaussian kernels, mathematically, maps our data into an infinite dimension space(yes I did NOT misspell it). These were just out of the many kernels available. In this way, we get our solution from a higher dimensional space without even visiting it.

There is, of course, a lot more math and logic involved than what I've explained here, but I hope I have been able to give you a decent intuition about the kernel trick. Happy learning!