On Convex optimization and Support Vector Machines

Arkin Gupta, Andrew Gates

July 8, 2018

Abstract

Support vector machines (SVMs) are an extremely powerful machine learning tool to solve various classification problems. Not only are they less prone to over-fitting due to large margins, but they are also easy to optimize due to their convex nature. In this paper we will review both soft and hard margin formulations of linear SVMs. First, we discuss how to solve soft-margin SVMs via dual formulation, and justify how the dual problem will in-fact give the optimal solution of primal form. Then, we discuss kernel tricks to solve non-linear classification using convex optimization. Finally, we perform classification on real-world data using both non-linear and linear SVMs using the algorithms devised prior.

1 Background

Given the data $\{\mathbf{x}_i, y_i\}_{i=1}^n$ is linearly separable, we want to find a hyperplane which separates this data: $\{\mathbf{x} : \beta^T\mathbf{x} + \beta_0 = \beta^T(\mathbf{x} - \mathbf{x}_0) = 0\}$.

1.1 Hard margin

In the case of linearly separable data, we use a hard margin (no tolerance for error in classification) and maximize the width of the margin defined by the hyperplane. First we define our classification rule as:

$$y_i(\beta^T x + \beta_0) \geq 1 \quad \forall i \tag{1}$$

where $y_i = sign(\beta^T x_i + \beta_0)$. Now, the width of the margin is given by

$$\frac{1 - \beta_0 + 1 - \beta_0}{\|\beta\|} = \frac{2}{\|\beta\|} \tag{2}$$

We want to maximize this width, which is equivalent to the following (convex) optimization problem:

$$\underset{\beta,\beta_0}{\operatorname{argmin}} \frac{1}{2}\|\beta\|^2 \quad \text{s.t.} \quad y_i(\beta^T x + \beta_0) \geq 1 \tag{3}$$

However, this is not the case very often, which is when we use a soft margin, adding tolerance to errors in classification.

1.2 Soft margin

Here we modify the classification rule and add slack variables:

$$y_i(\beta^T x + \beta_0) \geq 1 - \xi_i \quad \forall i \tag{4}$$

with the additional constraint that $\xi_i \geq 0 \quad \forall i$. Now the objective function is the following subject to the constraint above:

$$\underset{\beta}{\operatorname{argmin}} \frac{1}{2}\|\beta\|^2 + C\sum_i \xi_i \tag{5}$$

This concludes the required background.

2 Dual formation

In this section we will use Lagrange multipliers, slater's conditions and strong duality to give the dual formations of soft-margin SVMs and justify that the solution to the dual will give the optimal solution to the primal.

2.1 Duality for convex functions

It is a well known fact that given the following optimization problem:

$$min \quad f_0(x) \quad \text{s.t.} \quad f_i(x) \leq 0 \quad \text{for} \quad i = 1, ..., m \quad \text{and} \quad g_i(x) = 0 \quad \text{for} \quad i = 1, ..., p \tag{6}$$

we can define the Lagrangian in the following manner:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \tag{7}$$

and define the dual function as

$$g(\lambda, \nu) = \inf_x \quad L(x, \lambda, \nu) \tag{8}$$

Note that $g(\lambda, \nu) \leq f(x*)$. This follows trivially from the fact that $h_i(x) = 0$ and $f_i(x) \leq 0$. We know from slater's conditions that if the objective function is convex and constraint functions are affine, $f(x^*) - g(\lambda, \nu) = 0$. That is, strong duality holds and the solution for the dual is the solution for the primal. In this case, KKT conditions are necessary and sufficient to find an optimum. This is because (relaxed) Slater conditions are trivial, given tat in our case the constraints are affine. That is, we can simply maximize $g(\lambda, \nu)$ to solve for the minimum of $f_0(x)$.

2.2 Forming the dual function

Looking back at the primal optimization problem in (5), notice that the objective function is a norm, which is convex and the constraint function is affine. Therefore, we can simply solve the dual and arrive at the optimal solution for our primal solutions. Applying KKT conditions, we get the estimators. Recall that the Lagrangian is given as:

$$L(\beta_0, \beta, \lambda, \nu) = \frac{1}{2}\|\beta\|^2 + C\sum_i \xi_i + \sum_i \lambda_i(1 - y_i\beta^T x_i - \xi_i) - \sum_i \nu_i \xi_i \tag{9}$$

Now, setting $\frac{\partial L}{\partial \beta} = 0$ we arrive at

$$\beta = \sum_i \lambda_i y_i x_i \tag{10}$$

and setting $\frac{\partial L}{\partial \lambda} = 0$, $\frac{\partial L}{\partial \nu} = 0$ we get

$$\sum_i \lambda_i y_i = 0 \tag{11}$$

$$0 \leq \lambda_i \leq C \tag{12}$$

Combining these two and plugging back into (8) we have

$$g(\lambda, \nu) = \sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j x_i^T x_j \tag{13}$$

So finally, the dual problem is formulated as

$$\operatorname*{argmax}_{\lambda,\nu} g(\lambda, \nu) \tag{14}$$

subject to (11) and (12). Note that (11) is essentially the same as complementary slackness. Note that follows that we can calculate b (using a non-zero λ_i). Of course, we should average over all such points.

$$\beta_0 = y_i - \sum_i (\lambda_i y_i x_i) x_i \tag{15}$$

2.3 Classification rule

Now that we have a closed form for β, we can classify a new data point, z as:

$$sign(\sum_i \lambda_i y_i x_i z + \beta_0) \tag{16}$$

Note that we only have to evaluate a dot product to make classification decisions.

2.4 Discussion

The Lagrange multipliers λ_i can be thought of as giving weights to certain points. It makes sense that λ_i is non-zero if the point is a support vector. This is because let's say if two points a_i and a_j are not related, their dot product is 0, and they do not add to $g(\lambda, \nu)$ in (13). In the case where a_i and a_j are in the same direction (similar features), but are of the same class, if λ_i and λ_j are given positive values the term $\lambda_i \lambda_j y_i y_j x_i^T x_j$ will be positive, making g small. Lastly, if a_i and a_j are in the same direction (similar features) but have opposite classes, by giving λ_i and λ_j a positive value the term $\lambda_i \lambda_j y_i y_j x_i^T x_j$ is negative, adding to g as required. This concludes the duality section.

3 The Kernel trick

Converting the original primal optimization problem to a dual gives us two benefits: 1) Dual variables are easier to deal with, and our hyperplane parameters have a nice closed form only involving dot products. The latter proves to be very useful.

3.1 Non-linearly separable data

Very often we come across data which is not linearly separable, and one way we can make it linearly separable is to project the data points onto some higher dimensional space, and then find an appropriate SVM. Let's define a function

$$\phi(x) : R^d \to R^n \tag{17}$$

where d is the number of features and n is the dimension of the space onto which the data becomes linearly separable. However, note that it would be very computationally heavy to project every data point onto this new space and then find a hyperplane in $n-1$ dimensions. This is where kernel functions come into play.

3.2 The kernel function

The kernel function K, is defined as

$$K(x, y) = \phi(x)\phi(y) \tag{18}$$

The Gram matrix of a kernel function, $\mathbf{G}$ is defined as: $G_{ij} = K(i, j)$. A valid Kernel function is (1) symmetric and (2) as a positive semidefinite Gram matrix. Now we could simply apply this Kernel function to the dot product in the function g and in calculating $\beta z + \beta_0$, as opposed to projecting every x_i into the new vector space given by ϕ and proceeding from there. We have g and the updated classification rule:

$$g(\lambda, \nu) = \sum_i \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{K}(x_i^T x_j) \tag{19}$$

$$sign(\sum_i \lambda_i y_i \mathbf{K}(x_i, z) + \beta_0) \tag{20}$$

The kernel function gives us a way to compute the dot product of two vectors in a high dimensional space without actually requiring the high dimensional projections of those two vectors. This implies that we don't need to calculate projections, or store these new high dimensional features, but only compute the dot products with kernels.

4 Maintaining an optimal SVM with loss of data points

4.1 Intuition

It is not hard to see that the SVM can be generated using only a few crucial data points (support vectors). But it is important to note that these crucial points won't be chosen until the SVM is actually trained. That is, given an arbitrary dataset, it is not justified to remove data points before training the support vector machine.

4.2 Hard margin

In the case of hard margin, we can throw away every data point except those with corresponding non-zero Lagrange multipliers, $\alpha_i \geq 0$.

4.3 Soft margin

Here, we want to keep data points which are misclassified in addition the support vectors. That is, all points with non-zero Lagrange multipliers. $\alpha_i \geq 0$ and all x for which $y_i(\beta^T x + \beta_0) \geq 1 + \xi_i$ and $\xi_i > 1$.

5 Results

5.1 Data Generation

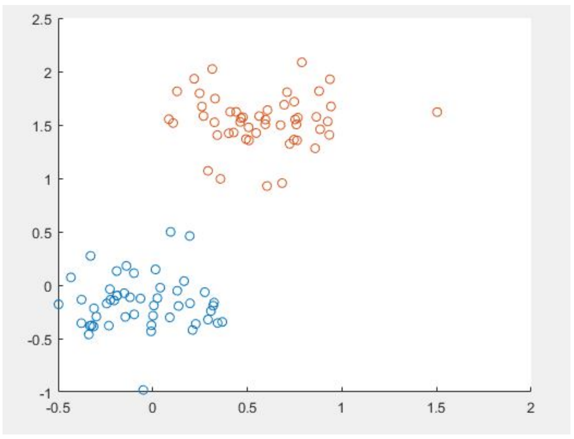
For all three problems we generated our own set of data. We attempted to use some datasets that we found on-line but creating our own allowed us to distribute the data precisely how we wanted for optimal testing. For both cases we used the randn() function to set a state so that our data would repeat every time we executed our program. This would allow us to test with various sets of data whenever we wanted to, but always allow us to repeat our testing on the same set of data when desired.

For the linear case we generated two sets of data X and Y, both 2-dimensions by creating a mean and standard deviation associated with X and Y. We then used the repmat() function to create two sets of data for X and Y based on the mean and standard deviation.

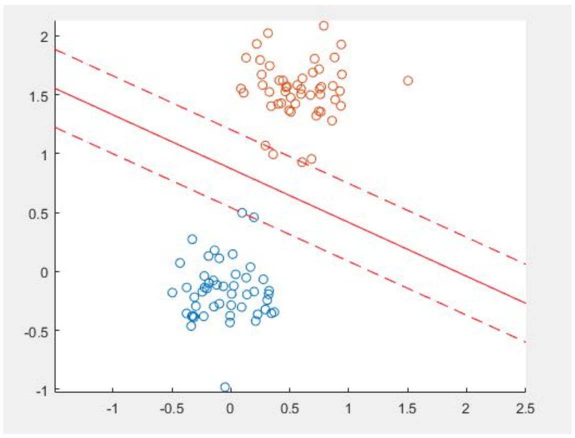
For the non-linear case we generated our data in a similar way, but this time we created a purely random set of data for X, and Y became either -1 or +1 depending on the class of the corresponding data points. We generated 500 points of data, 50 of which we used for training and the remaining 450 points we used for testing.

5.2 Linear Hard Margin SVM

For the linear hard margin example we used the data as shown below -



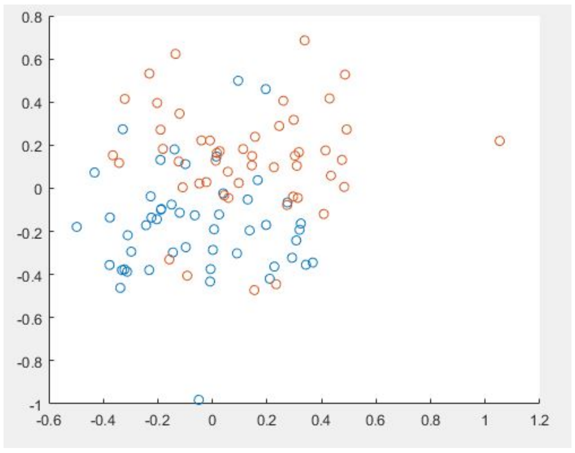
This data is linearly separable which allows a single hyperplane to cut the data into two distinct sets without any errors. If there was any data point that was too close to the other class of data points then this would be inseparable and a soft margin technique would have to be used. Once SVM is run on this set of the data the resulting hyperplane and support vectors are shown below -



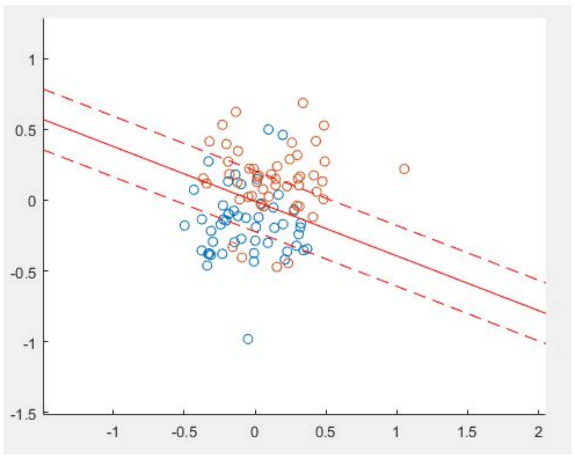
It can be seen here that the hyperplane cuts this data without error. There are roughly 4 support vectors in this case. Meaning that these are the only points that affect the SVM, all others can be ignored.

5.3 Linear Soft Margin SVM

For the linear soft margin example we used the data as shown below -



This data is inseparable by a single hyperplane, which will result in some errors. This is the idea behind the soft margin case. With hard margin there are no errors allowed, but soft margin allows for errors to produce the best hyperplane to separate the datasets with as little classification error as possible. Running SVM on this set of data results in the hyperplane with support vectors as shown below -

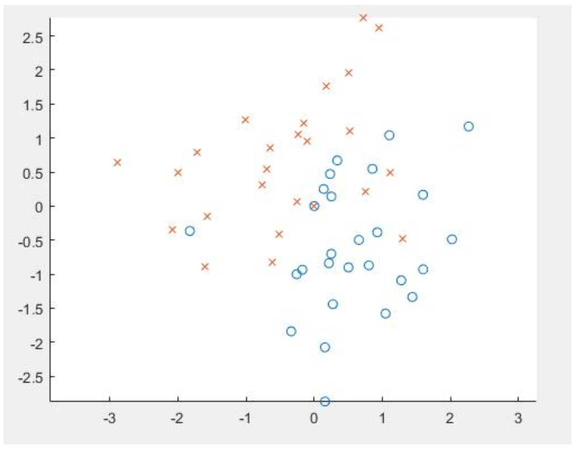


It's obvious that there are errors, which are expected. But this hyperplane cuts the data optimally which results in the smallest number of misclassified data. Similar to the hard margin case, all points that lie on the support vectors are the only points that affect the SVM, and all others can be ignored.

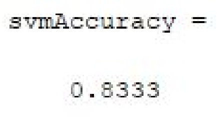
For both the Hard Margin and Soft Margin case we ran our program using the primal formulation of the SVM problem. We tested it using primal and dual but ended up using the primal in our final program. Both formulations produced the exact same results in terms of hyperplane and support vectors, which confirms our problem formulation earlier in the report that both formulations are empirically equivalent.

5.4 Non-Linear SVM

For the non-linear example we used the data as shown below -

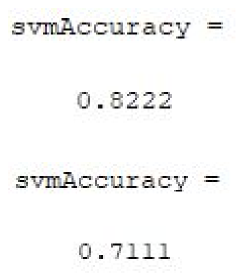


Non-linear SVM is similar to the soft margin SVM case. The data is inseparable just like the soft margin SVM case. The main difference though is that by using a kernel we can transform the non-linear data into a linearly separable set of data. Once this data has been transformed a hyperplane can be generated to cut this data which results in much higher accuracy than using a hyperplane on the original set of data. We used three different kernels to test our data, linear, gaussian and polynomial. Linear had the best accuracy, followed by gaussian, and lastly polynomial. The accuracy we achieved is shown below in this order -



```
svmAccuracy =

    0.8333
```



```
svmAccuracy =

    0.8222

svmAccuracy =

    0.7111
```

In our case linear was the most accurate, but for other data sets the best kernel was not linear. The best kernel for each situation is all dependent on how the data is distributed.

6 Discussion

Solving support vector machines can be a very computationally heavy problem when looking for a hyperplane using simply the primal optimization problem. Especially in the case of non-linearly separable data, applying a transformation function to every single training data would take exponentially more time that when solving the dual. As discussed above, solving the dual problem not only makes the optimization problem easier to deal with, but also helps us use the kernel function, the dot product of two data points in a higher dimensional space. This formation also gives us a closed form to compute the normal vector and bias of the separating hyperplane of the two polytopes formed by the data of the two classes. In terms of applying the SVM itself, if it's a data set that's linearly separable with absolutely no errors then a hard margin approach will work the best. If it's a set of data that's inseparable but still looks somewhat organized a soft margin approach will work well. If it's a set of data that is non-linear then the kernel trick comes in handy. Overall, the math behind support vector machines gets simplified quite a bit with only a few parameters to tune, making it an elegant solution for classification problems.

References

[1] Stephen Boyd and Lieven Vandenberghe *Convex Optimization*, Cambridge University Press, 2004.

[2] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx, September 2013.

[3] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs, Recent Advances in Learning and Control (a tribute to M. Vidyasagar), V. Blondel, S. Boyd, and H. Kimura, editors, pages 95-110, Lecture Notes in Control and Information Sciences, Springer, 2008. http://stanford.edu/ boyd/graph-dcp.html.