A Brief Exploration of the Mathematical Reasoning Behind Support Vector Machines

Brock Grassy

June 4, 2018

Abstract

Machine learning is increasingly becoming an integral part of today's society. While it is commonly thought of as an offshoot of computer science, it is also deeply rooted in fundamental concepts of mathematics and statistical theory; however, many introductory treatments of the subject neglect explanations of those topics that beget additional complexity. This paper attempts to provide such an examination at greater depth, through the lens of examination of Support Vector Machines, a specific type of machine learning model.

1 Introduction

Support Vector Machines (SVMs) were originally conceived of by Vladimir Vapnik and Alexey Chervonenkis in 1963 as a method of predicting what class data observations are classified into through the use of a separating hyperplane. The model was further refined, and in 1995 Vapnik and Corinna Cortes published a paper on the "modern" soft-margin, possibly nonlinear SVM [3]. Since then, further advances have been made in the literature; one of note is the concept of Bayesian Support Vector Machines, which allow for the use of tuning hyperparameters to optimize performance [8]. In this paper, I will discuss the fundamentals needed to understand how Support Vector Machines, and explain some of the underlying mathematical intuitions. I will first describe basic convex optimization, including the Karush-Kuhn-Tucker Conditions and the structure of general optimization problems. I will also describe statistical learning theory, and canonical problems and concepts that frequently come up when examining questions of statistical learning. After that point, I will have amassed enough theory to begin my description of the SVM model. I will first discuss the linear separating hyperplane problem, and then introduce the kernel trick and various transformations that the feature space can undergo. Finally, I will mention applications of SVMs to regression, and conclude with other potential lines of inquiry I could have pursued.

2 Definitions and Background

In order to understand statistical learning models, it is necessary to know some basics of statistical theory and optimization. I will describe some of those details here.

2.1 Convex Optimization

On a basic level, *optimization* aims to find values of a function that correspond to either maxima or minima. We define an *optimization problem* as a problem in the following form:

$$\begin{aligned} &\text{minimize} && f_0(x) \\ &\text{subject to} && f_i(x) \leq b_i, \ldots, m. \end{aligned} \tag{1}$$

We call f_0 the *objective* function and f_i the *constraint functions*, where b_i are the *bounds*. For this paper, we will be particularly concerned with *convex optimization*. The goal of convex optimization is to maximize and minimize convex functions over convex sets.

Definition 2.1. A set S is a *convex set* if for any $x_1, x_2 \in S$ and $t \in [0,1]$ implies that $tx_1 + (1-t)x_2 \in S$; that is, every line segment connecting points in S is fully contained in S.

Definition 2.2. A function f is convex on a convex set S if for all $x_1, x_2 \in S$ and $t \in [0,1]$, we have $f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$.

Intuitively, if a function is convex, the line segment between any two points on the graph of the function must sit above the part of the graph in between those points. When these restrictions are placed upon our functions and domains, we are able to make more headway on our problems than would typically be the case. We will encounter *quadratic programming problems* in this paper, which are optimization problems in the form

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ &\text{subject to} && A\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{2}$$

where $\mathbf{x}$ and $\mathbf{c}$ are n-dimensional vectors, Q is a symmetric $n \times n$ matrix, A is a $m \times n$ matrix, and $\mathbf{b}$ is a m-dimensional vector. The matrix notation amounts to a re-expression of systems of equations; fundamentally, we're trying to find $\mathbf{x}$ that minimizes a quadratic equation subject to a series of linear, or affine, constraints.

Now suppose we have an optimization problem as described in the start of the chapter, with added constraints $h_i(x) = 0, i = 1, 2, \ldots, p$.

Definition 2.3. The *Lagrangian* of the problem is given by

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

where the λ_i and ν_i terms are the *Lagrange multipliers* of the respective constraint functions. The vectors λ and ν are called the dual vectors.

We can define the *Lagrange dual function* by the infimum of the Lagrangian; that is, we have if f_0 is defined over some domain $\mathcal{D}$, we define

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu).$$

We allow g to take on the value $-\infty$ if our dual function is unbounded below. The dual function proves to be extremely useful when attempting to provide lower bounds on the optimal value we wish to find. Suppose $\lambda \succeq 0$ (we define $\mathbf{x} \succeq a$ to mean each element of $\mathbf{x}$

is greater than or equal to a) and x is a point in the domain of f_0 that satisfies all of the constraints. We thus have $f_i(x) \leq 0$, and $h_i(x) = 0$. Thus we have

$$f_0(x) + \sum_{i=1}^{n} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \leq f_0(x).$$

Since this holds for all x in the domain, taking the infimum we get that $g(\lambda, \nu) \leq f_0(x)$ for all x.

We have found some lower bound for $f_0(x)$ (that may be $-\infty$). In the case where $f_0(x) > -\infty$, we thus know that there exists some greatest lower bound. From this, we find another optimization problem, which we call the *Lagrange dual problem*:

$$\begin{aligned} &\text{maximize} && g(\lambda, \nu) \\ &\text{subject to} && \lambda \succeq 0. \end{aligned} \tag{3}$$

Henceforth, we will refer to this optimization problem as the *dual problem*, and the original optimization problem defined in (1) with the added equality constraint functions as the *primal problem*. Suppose we found that d^* is the maximum value of the dual function, and p^* is the minimum value of the primal function. We have that $d^* \leq p^*$ from our earlier claims about the lower bound the Lagrangian applies. We define $p^* - d^*$ to be the *dual gap*. If the dual gap is zero, we say that the optimization problem exhibits *strong duality*, otherwise it exhibits *weak duality*.

2.1.1 The Karush-Kuhn-Tucker Conditions

When examining an optimization problem, it often is useful to check to see if we see whether it exhibits strong duality prior to going through the work and calculating the maxima using the Lagrange dual function. There are many ways to do so, but checking if the problem satisfies the Karush-Kuhn-Tucker (KKT) conditions proves to be among the easiest ways to do so. The conditions were first published by William Karush in 1939 [6], and were later independently discovered and published by Harold Kuhn and Albert Tucker in 1951 [7]; the three of them now receive joint credit.

Suppose we have a convex optimization problem where the functions $f_0, f_1, \ldots, f_n$ are all differentiable, and the problem exhibits strong duality. Therefore, we have there exist x^*, λ^*, ν^* so that x^* maximizes f_0 and minimizes $L(x^*, \lambda^*, \nu^*)$ over x, and that those two values coincide. Thus, we have that $\nabla X(x^*, \lambda^*, \nu^*) = 0$; writing out our expression for the Lagrangian, we get

$$\nabla f_0(x^*) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i \nabla h_i(x^*) = 0.$$

Now, suppose that the f_i functions are both convex and differentiable, and that x^*, λ^*, and

ν^* satisfy the following constraints:

$$\begin{aligned} f_i(x^*) &\leq 0, \quad i = 1, \ldots, m \\ h_i(x^*) &= 0, \quad i = 1, \ldots, p \\ \lambda_i^* &\geq 0, \quad i = 1, \ldots, m \\ \lambda_i^* f_i(x^*) &= 0, \quad i = 1, \ldots, m \\ \nabla f_0(x^*) + \sum_{i=1}^{n} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(x^*) &= 0. \end{aligned}$$

We then have that x^* is a solution of the primal problem and (λ^*, ν^*) is a solution of the dual problem. The first three constraints are ones we stated must be true of the primal and dual problem for any properly formatted convex optimization problem, and the last constraint follows from the minimality constraint on the Lagrangian. The fourth constraint requires a little bit more work, but is still rather trivial. We know that the dual gap must be zero. Thus, we have

$$\sum_{i=1}^{n} \lambda_i^* f_i(x^*) + \sum_{i=1}^{p} \nu_i^* h_i(x^*) = 0.$$

However, all of the equality constraint functions are zero. Thus, this reduces to

$$\sum_{i=1}^{n} \lambda_i^* f_i(x^*) = 0$$

which acts as the final KKT condition. We will use these conditions to verify that solutions we find are maximal, and make claims about what possible solutions may look like. For more information on the KKT conditions and general convex optimization problems, Boyd and Vandenberghe's *Convex Optimization* [2] is a good resource.

2.2 Supervised Learning

The problem that we wish to examine in this paper is the problem of *supervised learning*. The basic task of supervised learning is to use the values of input variables to predict the values of output variables. An archetypal example of this method is to predict housing prices based on various factors such as location, square footage, or number of bedrooms. Furthermore, there are two different types of tasks which supervised learning can take on: *classification* and *regression*. Classification problems aim to predict discrete classes the input observations fall into, while regression problems predict 1ontinuous variables.

In terms of notation, we denote input variables by X_j, where the subscript indicates which input variable we are examining. If we have multiple input vectors holding p values, we can concatenate the vectors into the $N \times p$ input matrix $\mathbf{X}$, where the jth column of j is X_j. We call each row of $\mathbf{X}$ an *observation*, and the ith row is represented by x_i^T. Finally, we will denote continuous output variables by the $1 \times p$ output vector Y. Occasionally my notation will slightly change, but I will make sure to specify what observations or input vectors I am talking about.

2.3 Statistical Models as a Function

Let $X \in \mathbb{R}^p$ be an input vector, and let $Y \in \mathbb{R}$ be the output value. We can think of the supervised learning question as finding a function, f, such that

$$Y = f(X) + \epsilon \tag{4}$$

provides a good approximation. Here ϵ represents an error term that varies with X, which we aim to make small by our choice of f. We evaluate the "fit" of our approximation by using a *training set* of observations. We can denote this training set by the set of tuples (x_i, y_i), where x_i is the ith set of input variables and y_i is the corresponding value. Our eventual goal would be to use this function to predict values given a set of just the input variables, for which we want to find the corresponding outputs.

We require a heuristic to determine the goodness of the fit of f to our training set. Supposing x_i is a vector, meaning we have a single input variable, we can define the *residual sum of squares*

$$RSS(f) = \sum_{i=1}^{N} (y_i - f(x_i))^2. \tag{5}$$

to be the sum of squares of the Euclidean distance ℓ^2 between the actual and predicted values. This is one of the canonical loss functions that we use in machine learning. We can also define the *mean squared error* by

$$MSE(f) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2, \tag{6}$$

which is the residual sum of squares divided by degrees of freedom. Finally, we can define the *root mean squared error* by

$$MSE(f) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2}. \tag{7}$$

The advantage of RMSE as a metric is that its units are more interpretable; because we are taking the square root of the sum of squares, it ultimately has the same units as the original y_i, while the other two metrics have the units squared.

2.4 Bias and Variance

Suppose we have a statistical model described by $Y = f(X) + \epsilon$. We want to find some function $\hat{f}$ that approximates f as well as possible. Let $\mathbf{X}$ be the input variables for our training set, and $\mathbf{y}$ be the corresponding output variables. Let's define the *expected prediction error* of $\hat{f}$ at x_0 to be

$$EPE(x_0) = E[(\mathbf{y} - \hat{f}(x_0))^2 \mid X = x_0]. \tag{8}$$

This is the expected squared error of the function, given that the input variable is equal to x_0. We can decompose this to

$$\sigma^2 + \left[f(x_0) - \frac{1}{k} \sum_{\ell=1}^{k} f(x_{(\ell)}) \right]^2 + \frac{\sigma^2}{k}. \tag{9}$$

Observe that there are three terms in this expression. We call σ^2 the *irreducible error*. This is the result of the ϵ term in the model; as is implied by the name, no matter what we do we cannot eliminate this term from the error.

We call the sum of the other two terms the *mean squared error*, as it is expected value of the square of the error of our approximation over our training set. The second term, or *squared bias* is a rough measure of the rigidity and applicability of our model. If the bias is high, that means that the model will be simpler. If it is too high, the model may *underfit* test results and may not provide a good estimation. If the bias is too low, the model will *overfit*. The model will fit the test data extremely well, but may not generalize to other similarly generated training data.

Finally, the last term is the *variance*. As the complexity of the model increases, bias typically increases and variance decreases. We call this relationship the *bias-variance tradeoff*, which is something that needs to be kept in mind when selecting models. In practice, we want to find a model that provides a good balance between training set and test set performance. For further information on this subject, Hastie, Tibrishani, and Friedman's *The Elements of Statistical Learning* [4] provides a good treatise on the matter.

2.5 Case Study: Linear Regression

Linear regression assumes that the predictor function f is linear in nature; that is, we can write f as a linear combination of the input vectors. Let $\beta \in \mathbb{R}^{p+1}$ be a vector with $p+1$ real values, and let $X^T = (X_1, X_2, \ldots, X_p)$ be the vector of input variables. We can write our function f as

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j. \tag{10}$$

Linear regression aims to pick β such that the residual sum of squares, as defined in section 2.1, is minimized. Observe that if we append 1 to the start of X (e.g. $X_0 = 0$), we now have

$$f(X) = \sum_{j=0}^{p} X_j \beta_j \tag{11}$$

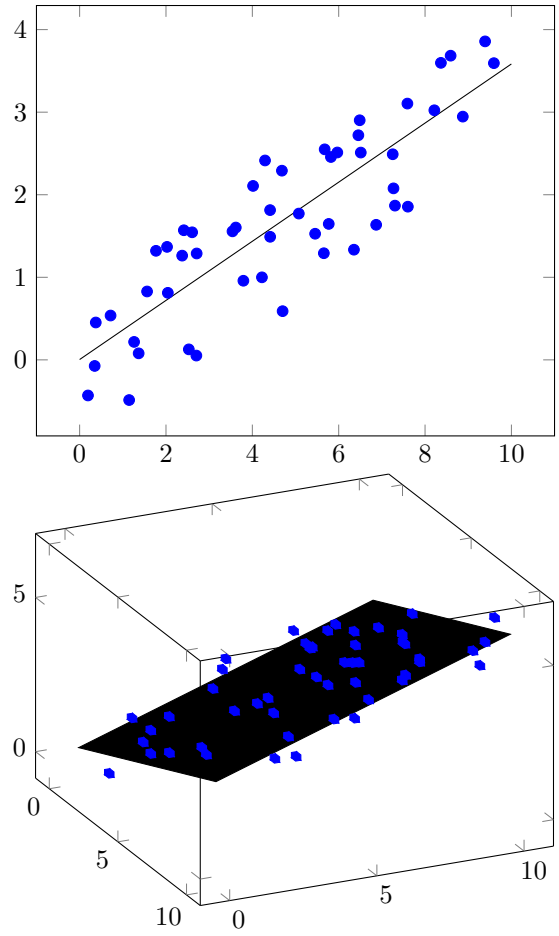
Now, suppose that we have the $N \times p + 1$ matrix $\mathbf{X}$, which holds N observations of the $p+1$ input vectors in X, and the N-vector $\mathbf{y}$ that holds the values of the observations in the training data. We have that $\mathbf{X}\beta$ results in a N-vector $\hat{\mathbf{y}}$, where the ith element $\hat{\mathbf{y}}_i = \sum_{j=0}^{p} \mathbf{X}_{ij}\beta_j$. Thus, $\hat{\mathbf{y}}$ holds the predicted values of f on each observation.

The solution of the least squares problem can be found using linear algebra and examining the projection of the output vector onto the feature space, and ultimately proves to be

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

2.5.1 Benefits and Downsides of Linear Regression

As described above, linear regression proves to be a simple way to construct a predictor based on our test data. As implied by the name, linear regression constructs a linear fit. Below are examples of the type of prediction that linear regression would result in.



I generated the two datasets by constructing linear relationships between the input variables and the output variable, the first of which by $y = \frac{1}{3}x$ and the second by $z = \frac{1}{3}x + \frac{1}{4}y$. The x-values were generated from a random uniform univariate distribution between 1 and 10. Additionally, I added on a randomly distributed error term to the output variable, so that the distribution wasn't immediately linear. As can be seen, the approximation works very well for input data that falls in a linear distribution. However, when our data is in a nonlinear distribution, this clearly would not work as well. As a result, the scope and applicability of this method is rather limited. Framing this within our earlier discussion of bias and variance, linear regression has lower bias and higher variance for a nonlinearly distributed dataset. It is thus necessary to develop more complex models that allow for greater range.

3 Support Vector Classifiers

This now brings me to the main purpose of my paper: discussing the mathematics behind Support Vector Machines, which I will refer to as SVMs. SVMs were initially conceived as a classification algorithm, which aims to separate data into different classes by using a *separating hyperplane*.

Definition 3.1. A *hyperplane* in an n-dimensional ambient space is a subspace of dimension $n-1$ of that ambient space.

For the purpose of this paper we will only consider affine subspaces of vector spaces. We define an *affine subspace* of a vector field V to be subspaces of the form $A = \{a + w : w \in V\}$. Intuitively, we can think of this as a vector space with a possible translation away from the origin; while vector spaces require the zero vector to be in them, there is no such constraint on general affine subspaces. This means that if we were to take $\mathbb{R}^2$ as our vector space, the affine subspaces of $\mathbb{R}^2$ are all possible lines in the plane. Likewise, the affine subspaces of $\mathbb{R}^3$ are all planes in $\mathbb{R}^2$. We can provide an alternative definition of a hyperplane as the set

$$\{x : x^T\beta + \beta_0 = 0\}$$

where $\|\beta\| = 1$ and $x, \beta, \beta_0 \in \mathbb{R}^n$.

As a consequence of the affine nature of our hyperplanes, we can say that our hyperplanes separate the ambient space into two parts. The geometric intuition remains clear when examining lines splitting $\mathbb{R}^2$ and planes splitting $\mathbb{R}^3$. Consequentially, we can define a classification function from this hyperplane

$$G(x) = \text{sign}[x^T\beta + \beta_0]$$

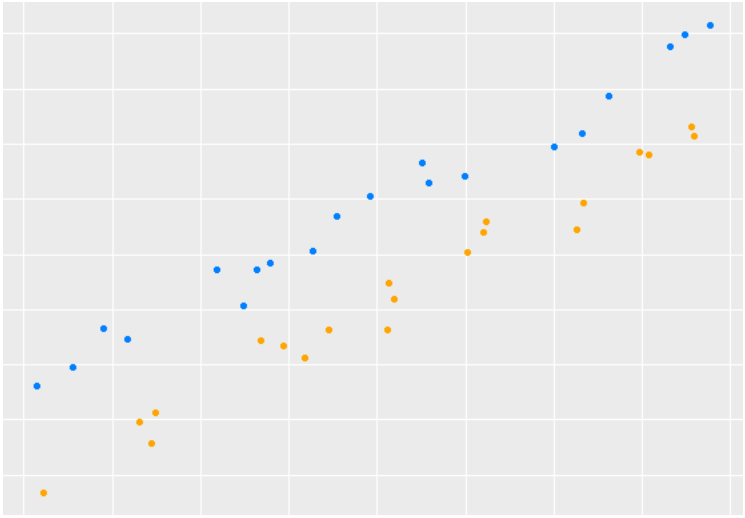
which classifies observations by whether they are above or below the hyperplane, where the sign function returns 1 for non-negative numbers and -1 for negative numbers. Let (x_i, y_i) be the set of ordered pairs of observations and their corresponding outputs. We define the loss function

$$D(\beta, \beta_0) = -\sum_{i \in \mathcal{M}} y_i(x_i^T\beta + \beta_0),$$

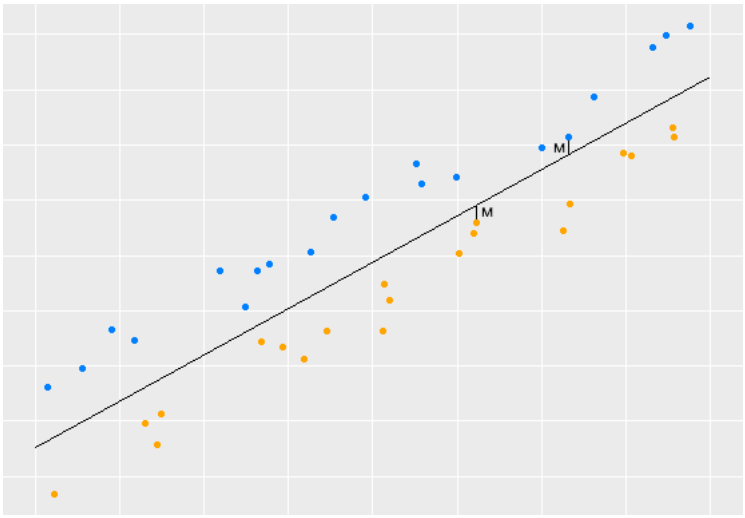
where $\mathcal{M}$ is the set of all indices of misclassified points. Observe that if $y_i = 1$ and it is misclassified, this means that $x_i^T\beta + \beta_0 < 0$. We get the opposite result for $y_i = -1$. Thus, multiplying this quantity by y_i and summing over all i gives us a negative value, so our loss function is non-negative. Additionally, its terms are all proportional to their distance from the hyperplane. It is intuitively clear how this loss function allows us to find a hyperplane that provides a "better" fit. There are now two main cases for our data that we must examine.

3.1 Maximum Margin Classifier

The first case we can examine is when the two classes are perfectly separated; that is, we can construct a hyperplane such that all observations from one class are on one side, and all observations in the other class are on the other. One such perfectly separable dataset is like so:



Observe, however that this hyperplane is not unique; in fact, there are infinitely many hyperplanes that perfectly separate the classes. It is thus necessary to determine which one of of these we should choose for our classification function. We define the *margin* to be the largest number M such that every element is at least M distance away from the hyperplane. The *maximum margin hyperplane* for the previous dataset, as well as a representation of the margin, is shown below:



We want the margin to be larger, as intuitively it allows for greater test accuracy. We would be confidently able to predict values that are on either side of the margins. It proves to be better to have larger margins, as it intuitively increases the decision boundary to some degree; however, the true benefits of the large-margin characteristic are apparent when examining transformations of the hyperplane.

We can express the problem of finding the maximum margin hyperplane in terms of an optimization problem as follows:

$$\max_{\beta,\beta_0,\|\beta\|=1} M$$
$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M, i = 1, \cdots, N.$$

Observe that we still have a constraint on the norm of β. Because $\|\beta\| = 1$, we can rewrite our second constraint to $y_i(x_i^T\beta + \beta_0) \geq M\|\beta\|$. Suppose that we have β and β_0 that satisfy this inequality. Scaling both vectors by the same positive scalar retains that inequality. Thus, let's choose β so that $\|\beta\| = \frac{1}{M}$. We now have the optimization problem:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$
$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, \cdots, N.$$

Using optimization techniques, it proves to be possible and relatively simple to solve this problem. I will not prove the solution for the perfectly separable case, but will do so for the non-separable case that follows.

3.2 Classifying Overlapping Classes

Let's now consider the cases where no such perfectly separating hyperplane exists. It is clear that our initial method requires some modification to work properly. To do this, we allow for points to be on the wrong side of the margin, but add a penalty for each wrongly classified observation. We call these penalizing factors *slack variables*, and denote them by $\xi = (\xi_1, \xi_2, \ldots, \xi_N)$. We define these slack variables in terms of an optimization constraint like our previous case:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2$$
$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, i = 1, \cdots, N$$

where $\xi_i \geq 0$ and $\sum \xi_i$ is less than some constant. Recall that ξ_i is a measure of the amount that the variable falls on the wrong side of the margin. Bounding the sum of these ξ_i by some constant will thus create a bound on the number of misclassifications. I will not provide a proof for the maximizing hyperplane in this case, as it requires additional complex convex optimization machinery. Ultimately, the answer proves to follow a similar structure where the hyperplane is expressed as a combination of support vectors that depend on the misclassified points.

3.2.1 Deriving the Optimal Hyperplane

In the previous section, we defined the optimization problem for the general linear Support Vector Classifier. We are able to rewrite it like so:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i$$
$$\text{subject to } \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i.$$

In our previous section, we stated that $\sum \xi_i$ was bounded by some constant. We replace it by C in this definition. This is a quadratic convex optimization problem, so we know how to solve it. Using the method of Lagrange Multipliers with multipliers α_i and μ_i, we get the primal function

$$L_P = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N} \mu_i\xi_i. \tag{12}$$

We want to find the values of β, β_0 and xi_i. We differentiate with respect to each of these, to get minimizing values

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i, \tag{13}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i, \tag{14}$$

$$\alpha_i = C - \mu_i, \forall i, \tag{15}$$

We can substitute these minimum values into (12) to get our Lagrangian

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j x_i^T x_j^T.$$

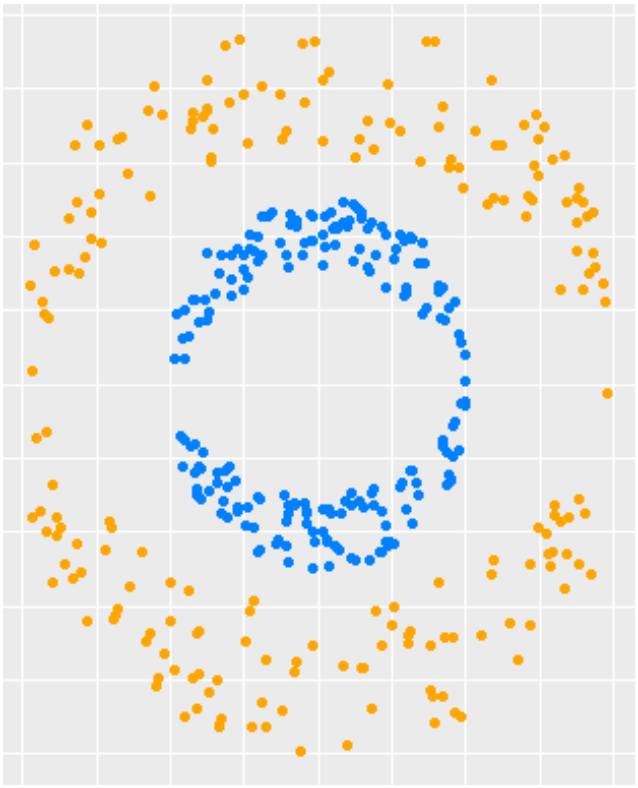
Maximizing this subject to $0 \leq \alpha_i \leq C$ and $\sum a_i y_i = 0$ gives us a solution. Furthermore, we can write out our additional KTT conditions for the problem:

$$\begin{aligned} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] &= 0, \\ \mu_i\xi_i &= 0, \\ y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) &\geq 0. \end{aligned} \tag{16}$$

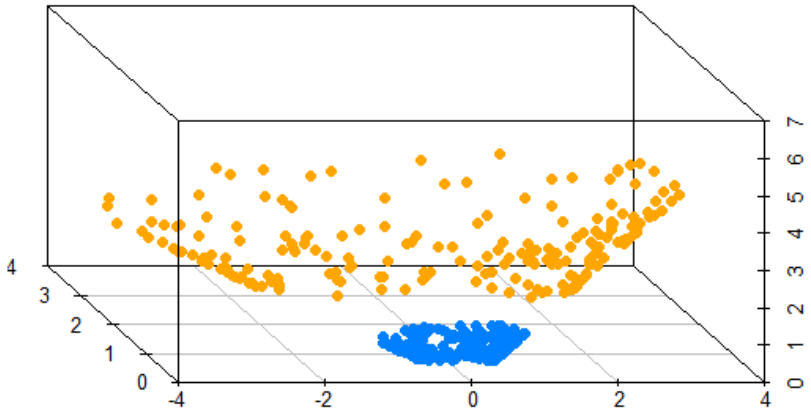
From this and (14), we get that α_i is zero when $y_i \neq 0$, and is only nonzero when all of the constraints are met. Recall that we found that $\beta = \sum_{i=1}^{N} \alpha_i y_i x_i$. Thus, β is the sum of vectors that are only nonzero when $y_i \neq 0$. We call these vectors *support vectors*, hence the name of the method.

3.3 Support Vector Machines

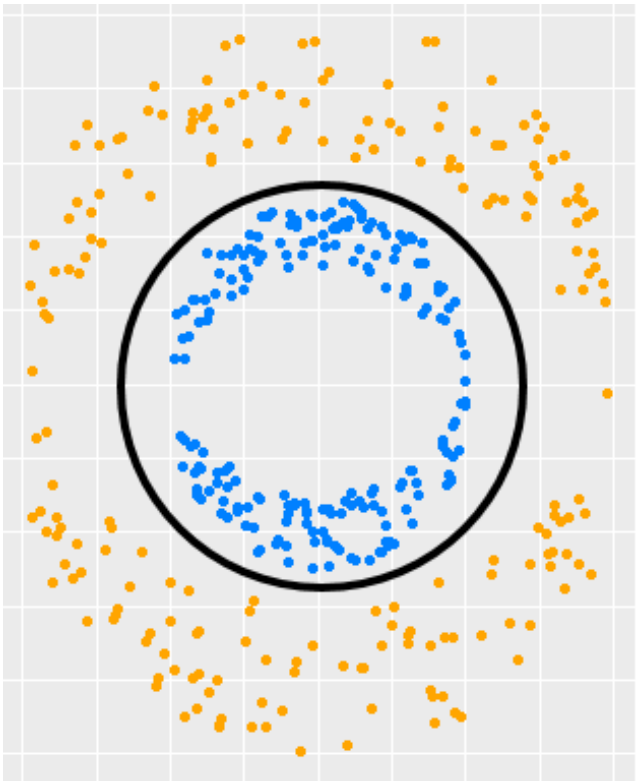
Up to this point, we have defined Support Vector Classifiers using linear decision boundaries in our feature space. As discussed in previous sections, however, we know this is frequently an inadequate strategy, as we rarely have data that actually follows that trend. As such, it seems that SVCs would not prove to be a very successful classification model. However, the strength of the method comes in its transformability. Consider the following graph of observations, which is the canonical example for this purpose:



As can be seen, there is no linear decision boundary that separates this or comes close to separating it. However, let's transform the points into $\mathbb{R}^3$, where the z-values of the x_i points are equal to $\|x_i\|$. Applying the transformation $\phi(x, y) = (x, y, x^2 + y^2)$, We end up with a diagram that looks roughly like this:



We can now construct a plane on the z-axis that perfectly separates the two classes. Now, transforming this back to the xy-plane, we get



which is a circular decision boundary. As can be seen, incorporating these transformations into our classifications allow for greater flexibility in our models, and ultimately better results.

3.3.1 Kernel Functions

Suppose we have a transformation function $h(x)$, that transforms our feature space. We can define the *kernel function*

$$K(x, x') = \langle h(x), h(x') \rangle$$

which computes inner products in the transformed space. This transformation is commonly referred to as the "kernel trick". Recall that we defined our standard SVC decision function by

$$\hat{G}(x) = \text{sign}[x^T \hat{\beta} + \hat{\beta}_0]$$

where $\hat{\beta}$ is in the form

$$\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i.$$

The Support Vector Machine defined by the transformation $h(x)$ is given by the decision function

$$\text{sign}[f(x)]$$

where

$$f(x) = \sum_{i=1}^{N} a_i y_i K(x, x_i) + \beta_0.$$

Basically, we still sum over combinations of the inputs; however, we transform those inputs with respect to the unknown input x that we wish to classify. There are three main kernel functions that are used in modern SVM literature:

$$d\text{th-Degree polynomial: } K(x, x') = (1 + \langle x, x' \rangle)^d,$$
$$\text{Radial basis: } K(x, x') = (-\lambda \|x - x'\|^2),$$
$$\text{Neural network/sigmoid: } K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2).$$

To further visualize how our kernels transform the feature space, let's consider a 2nd-Degree polynomial kernel with kernel function $K(x, x') = (1 + \langle x, x' \rangle)^2$. Suppose that x and x' are both vectors in $\mathbb{R}^2$, where $x = (x_1, x_2)$ and $x' = ({x_1}', {x_2}')$. Using the standard inner product for Euclidean spaces, we get

$$\begin{aligned} K(x, x') &= (1 + \langle x, x' \rangle)^2 \\ &= (1 + x_1 {x_1}' + x_2 {x_2}')^2 \\ &= 1 + x_1^2 ({x_1}')^2 + x_2^2 ({x_2}')^2 + 2x_1 {x_1}' + 2x_2 {x_2}' + 2x_1 {x_1}' x_2 {x_2}' \end{aligned}$$

We can further rewrite this expression as a dot product like so:

$$K(x, x') = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2) \cdot (1, ({x_1}')^2, ({x_2}')^2, \sqrt{2}x_1', \sqrt{2}x_2', \sqrt{2}x_1' x_2').$$

Observe, however, that this is in the form $\langle h(x), h(x') \rangle$ where

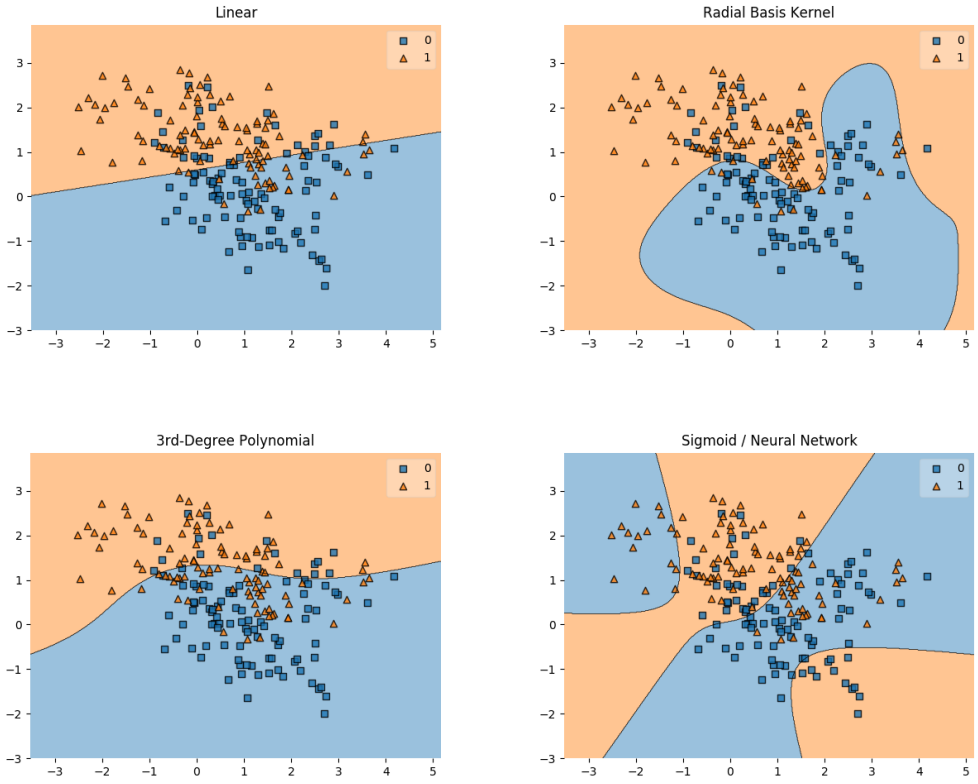
$$h(x) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2).$$

Thus, we can see that using a kernel allows us to transform our data from taking inner products in $\mathbb{R}^2$ to $\mathbb{R}^6$ without explicitly interacting with data in the transformed feature space. However, we cannot simply use any function as our kernel function. Let's define the kernel matrix $\mathbf{K}$ by $\mathbf{K}_{ij} = K(x_i, x_j)$, where x_i and x_j are the ith and jth input vectors. We now have a theorem that states

Theorem 3.1 (Mercer's Theorem)**.** A kernel matrix must be positive semidefinite. Furthermore, for any positive semidefinite matrix, there exists ϕ such that $K(x, y) = \langle \phi(x), \phi(y) \rangle$.

Thus, we require that any valid kernel function must have a positive semidefinite kernel matrix. Recall that a *positive semidefinite matrix* is a square $n \times n$ matrix M so that $z^T M z \geq 0$ for any nonzero column vector z with n elements. We now have a way to verify whether a given kernel function ϕ will provide a transformation that can be used for the kernel trick.

Below are examples of different kernels applied to a specific dataset:



As can be seen, the different kernel functions allow for different-shaped decision boundaries. In this case, the Radial Basis Kernel proves to be the best fit; the linear and polynomial boundaries are too rigid for our purposes. Consequentially, the RBF kernel is typically a good fit for a kernel function if we know there is no clear linear or polynomial separation within the classes.

We are able to define an optimization problem for the kernel problem. Let $\mathbf{K}$ be the matrix of the kernel function evaluated at two input vectors; that is, if we have input vectors x_j, $\mathbf{K}_{ij} = K(x_i, x_j)$. We can thus state that our optimization problem has the goal of minimizing

$$\sum_{i=1}^{N}(1 - y_i f(x_i)) + \frac{\lambda}{2}\alpha^T \mathbf{K}\alpha. \tag{17}$$

This is harder to solve, and beyond the scope of this paper; refer to [1] for a further exploration and explanation of the kernel trick.

4 Support Vector Regression

We can further generalize the Support Vector Machine model to work for questions of regression. Suppose we have a set of input and output pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

We can define a simple linear regression function analogous to the linear SVC by

$$\hat{y} = x^T\beta + \beta_0.$$

Instead of determining whether the prediction falls above or below the separating hyperplane, the values of the hyperplane themselves act as the predictor. We still aim to pick the hyperplane to maximize the margin, as we did when attempting to handle the issue of classification.

As we did for classifiers, we can also consider a soft margin separator. The goal of the soft margin for our initial SVM was to allow for misclassifications and penalize them accordingly, through the use of slack variables. For some observation (x, y), we add the constraint

$$|y - (x^T\beta + \beta_0)| < M + \xi$$

where M is the margin and ξ is our slack variable. Again, we bound $\sum_{k=1}^{i} \xi_i$ above by some constant C.

The structure of the optimization problem we need to solve is also extremely similar to that of the linear Support Vector Classifier. We end up wanting to minimize the function

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}\|\beta\|^2, \tag{18}$$

where we define

$$V_\epsilon(r) = \begin{cases} 0 & \text{if } |r| < \epsilon, \\ |r| - \epsilon, & \text{otherwise.} \end{cases} \tag{19}$$

V_ϵ acts as an error function that ignores errors that are smaller than ϵ. We are able to solve this in a similar manner to our original problem, and get that the optimal value for β can be expressed as the sum of support vectors. Like the classification problem, we are able to use kernels to transform the feature pace. In this case, the transformation comes by changing the error function and minimizing

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}\|\beta\|^2, \tag{20}$$

where V is some general error measure. We ultimately find that the optimal f for this is of the form

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{a}_i K(x, x_i),$$

which indicates how the kernel function manifests itself.

5 Conclusions

As I discussed, Support Vector Machines provide a flexible way to both classify and regress various datasets. While I briefly alluded to the practical applications of SVMs and how they would be used for real-life purposes, I did not describe any additional details or further lines of

inquiry in depth. While SVMs were originally designed and used as a two-class classification problem, it is possible to modify them to classify inputs between several other classes. In Chi-Wei Hsu and Chih-Jen Lin's *A Comparison of Methods for Multi-class Support Vector Machines* [5], they discuss various ways which SVMs can be used for multiclass classification problems. Their research found that two methods, the leave-one-out method and DAGSVM method, gave best results. The leave-one-out method constructs n SVMs if there are n classes. Each SVM has the same data, but is reclassified to be binary on whether the observation is in that given class or not. The DAGSVM method takes a different approach, constructing a DAG (directed acyclic graph) with SVMs as nodes, and follows a path corresponding to each result that eventually ends up with a final class being selected.

SVMs are also used for a variety of problems in the real world, such as face detection, handwriting recognition, and protein folding. Although there may be newer models that may provide higher performance on many problems, such as random forests and gradient boosting machines, SVMs still remain relevant in the upper echelon of machine learning models. Even if it eventually becomes outdated, the ubiquity of the kernel trick and fundamental concept of the model will still allow it to remain as a capable example of the power that machine learning has.

References

[1] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory - COLT 92* (1992).

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

[3] Corinna Cortes and Vladimir Vapnik. "Support-vector Networks". In: *Machine Learning* (1995), pp. 273–297.

[4] Trevor J. Hastie, Robert John Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer, 2009.

[5] Chi-Wei Hsu and Chih-Jen Lin. "A Comparison of Methods for Multi-class Support Vector Machines". In: *IEEE Transactions on Neural Networks* (2002), pp. 415–425.

[6] William Karush. "Minima of Functions of Several Variables with Inequalities as Side Constraints". PhD thesis. Chicago, Illinois: University of Chicago, 1939.

[7] Harold W. Kuhn and Albert W. Tucker. "Nonlinear Programming". In: *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability* (1951), pp. 481–492.

[8] Peter Sollich. "Bayesian Methods for Support Vector Machines: Evidence and Predictive Class Probabilities". In: *Machine Learning* (2002), pp. 21–52.