Lecture Notes

Introduction to Support Vector Machines

Raj Bridgelall, Ph.D.

Overview

A support vector machine (SVM) is a non-probabilistic ***binary*** linear classifier. The non-probabilistic aspect is its key strength. This aspect is in contrast with probabilistic classifiers such as the Naïve Bayes. That is, an SVM separates data across a decision boundary (plane) determined by only a small subset of the data (feature vectors). The data subset that supports the decision boundary are aptly called the ***support vectors***. The remaining feature vectors of the dataset do not have any influence in determining the position of the decision boundary in the feature space. In contrast with SVMs, probabilistic classifiers develop a model that best explains the data by considering all of the data versus just a small subset. Subsequently, probabilistic classifiers likely require more computing resources.

The binary and linear aspects, however, are two SVM limitations. Recent advances using a "Kernel Trick" have addressed the linearity restriction on the decision boundary. However, the inability to classify data into more than two classes is still an area of ongoing research. Methods so far involve creating multiple SVMs that compare data objects among themselves in a variety of ways, such as one-versus-all (OVA) or all-versus-all (AVA) (Bhavsar and Ganatra 2012). The latter is also called one-versus-one (OVO). For *k* classes, OVA requires training *k* classifiers so that each class discriminates against the remaining *k*-1 classes. AVA requires k(k-1)/2 classifiers because each class is discriminated against every other class, for all possible pairings. After constructing the number of required binary classifiers for either method, a new object is classified based on the comparison that provides the largest discriminant value.

The SVM separates all data objects in a feature space into two classes. The data objects must have features $\{x_1 \ldots x_n\}$ and a class label, y_i. SVM treats each data object as a point in feature space such that the object belongs to one class or the other. Specifically, a data object (characterized by its feature vector) either belongs to a class, in which case the class label is $y_i = 1$, or it does not belong to the class (implying that it belongs to the other class) in which case the class label is $y_i = -1$. Therefore, the definition for the data is

$$\text{Data} = \left\{ (x_i, y_i) \mid x_i \in \Re^p, y_i \in (-1,+1) \right\}_{i=1}^{n} \tag{1}$$

where *p* is the dimension of the feature vector and *n* is the number of data points.

During training, the SVM classifier finds a decision boundary in the feature space that separates the data objects into the two classes. The optimization problem is to find the decision boundary (a linear hyperplane) that has the maximum separation (margin) between the two classes. The margin of a hyperplane is the distance between parallel equidistant hyperplanes on either side of the hyperplane such that the gap is void of data objects. The optimization during training finds a hyperplane that has the maximum margin. The SVM then uses that hyperplane to predict the class of a new data object once presented with its feature vector.

Hyperplane Definition

In geometry, a hyperplane is a subspace that has one dimension fewer than its ambient space. The hyperplane separates the space into two parts. A classifier is linear when it uses a hyperplane in multidimensional space to separate data. From geometry, the general equation for a hyperplane is

$$\mathbf{w} \bullet \mathbf{x} = 0 \tag{2}$$

For example, in a 2D space the equation of a hyperplane is a line where

$$y = ax + b \tag{3}$$

Rewriting the equation for the line yields the standard form for the hyperplane, which is

$$y - ax - b = 0 \tag{4}$$

The equivalent vector notation is

$$\mathbf{w} = \begin{pmatrix} -b \\ -a \\ 1 \end{pmatrix} \quad \text{and} \quad \mathbf{x} = \begin{pmatrix} 1 \\ x \\ y \end{pmatrix} \tag{5}$$

where $w_0 = -b$, $w_1 = -a$, and $w_2 = 1$. Taking the dot product of **w** and **x** and setting that equal to zero then produces the equation of a line in standard form. Given that the vectors are column vectors, the dot product is equivalent to the matrix operation

$$\mathbf{w}^T \mathbf{x} = 0 \tag{6}$$

The vector and matrix forms are easier to work with when using matrix algebra within numerical packages. Some notations explicitly write out the $w_0 = -b$ component of the vector, in which case the dimension index starts at 1 instead of 0.

The **w** vector is always normal to the hyperplane (Dawkins 2017). Its unit vector **u** is

$$\mathbf{u} = \frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{7}$$

The vector **w** must be perpendicular (normal) to the hyperplane H0 because the dot product is zero by definition, when the cosine of the angle between **x** and **w** is zero. The cosine is zero when the angle is 90 degrees. For the 2D example,

$$\mathbf{u} = \frac{\mathbf{w}}{\|\mathbf{w}\|} = \left(\frac{w_1}{\|\mathbf{w}\|}, \frac{w_2}{\|\mathbf{w}\|} \right) \tag{8}$$

This unit **w** vector is important in finding the distance of any point (feature) from the hyperplane by projecting that point to a normal vector of the hyperplane. For example, vector **p** in Figure 1 is the projection of point A onto the plane of the **w** vector. Hence the distance from point A to the hyperplane is the same as the length of **p** or ||**p**||. The projection of vector **a** onto the plane of **w** is **p** where

$$\mathbf{p} = (\mathbf{u} \bullet \mathbf{a})\mathbf{u} \tag{9}$$

The dot product produces a scalar, which is the magnitude (length) of the vector such that

$$\mathbf{u} \bullet \mathbf{a} = \sum_i u_i a_i \tag{10}$$

and the direction of the vector is **u**.

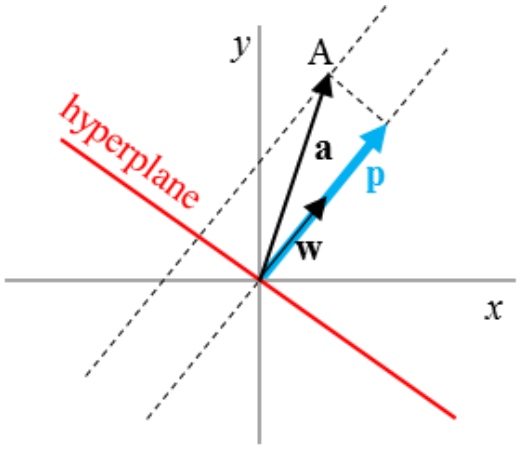

Figure 1: Projection of a vector to compute the distance to a hyperplane.

Margin of a Hyperplane

The margin of a hyperplane is twice the length (norm) of the projection of the nearest point such that

$$m = 2\|\mathbf{p}\| \tag{11}$$

The norm of a vector is the square root of the sum of squares of all the elements in the vector such that

$$\|\mathbf{p}\| = \sqrt{\sum_i p_i^2} \tag{12}$$

Optimal Hyperplane

As shown in Figure 2, the plane bounded by the margin along the hyperplane is void of data points. The optimal hyperplane is simply one that provides the largest margin *m*. As noted before, the support vectors are at the boundaries (H1 and H2) of the plane that the hyperplane equally bisects. Figure 2 indicates the support vectors with thick black borders. The data might be linearly or non-linearly separable. For linearly separable data, the problem amounts to maximizing the margin of a hyperplane H0 that separates the data. Handling non-linearly separable features require a transformation of the features to a higher dimension space by using kernel functions (covered later). The general solution uses linear Kernels for linearly separable feature spaces.

Constraints of Classification

Given a weight vector **w**, a parallel hyperplane with offset +δ to one side is

$$\mathbf{w} \bullet \mathbf{x} = +\delta \tag{13}$$

Hence, the equal and opposite offset to the other side of the hyperplane is

$$\mathbf{w} \bullet \mathbf{x} = -\delta \tag{14}$$

Given a feature vector $\mathbf{x}_i$, the assigned class must satisfy

$$\mathbf{w} \bullet \mathbf{x}_i \geq +\delta \tag{15}$$

for a class label of $y_i = 1$ and

$$\mathbf{w} \bullet \mathbf{x}_i \leq -\delta \tag{16}$$

for a class label of $y_i = -1$.

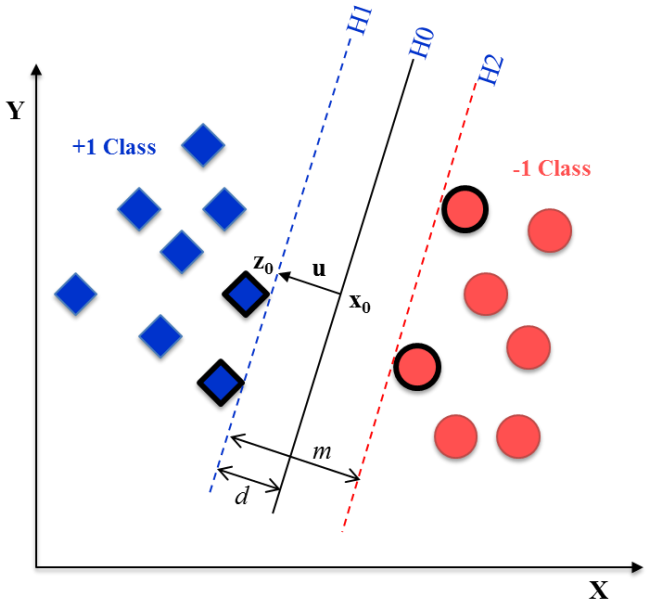

Figure 2: Maximizing the margin of a hyperplane.

The SVM will calculate the optimum **w** for any offset. Therefore, any non-zero offset will establish a margin that the optimization can maximize. For mathematical convenience, theoreticians set the offset to δ=1 to simplify the design of the classifier. This allows the two classification constraints to be combined into a single constraint. That is, setting δ=1, multiplying both sides of equation (15) by y_i, and assigning the class label value of +1 to the right yields

$$y_i(\mathbf{w} \bullet \mathbf{x}_i) \geq +1(+1) \Leftrightarrow y_i(\mathbf{w} \bullet \mathbf{x}_i) \geq 1 \tag{17}$$

Doing the same for equation (16) with its y_i label value at -1 yields

$$y_i(\mathbf{w} \bullet \mathbf{x}_i) \leq -1(-1) \Leftrightarrow y_i(\mathbf{w} \bullet \mathbf{x}_i) \geq 1 \tag{18}$$

Note that per rule for inequalities, multiplying by negative 1 flips the inequality sign. Hence, the "mathematical convenience" produced the single constraint

$$y_i(\mathbf{w} \bullet \mathbf{x}_i) \geq 1 \quad \forall \quad 1 \leq i \leq n \tag{19}$$

Equation (19) becomes the constraint in the optimization problem to ensure that the margin is void of data points. Training the SVM involves determining the hyperplane by solving for **w** that provides the maximum margin, given the data labels y_i and feature vectors $\mathbf{x}_i$.

Distance Between Hyperplanes

As noted before, the vector **w** is perpendicular (normal) to the hyperplane H0. Therefore, its unit vector **u** = **w**/||**w**|| must be perpendicular to H0 with magnitude 1. The vector $d\mathbf{u}$ is the perpendicular vector between hyperplane H0 and a parallel hyperplane separation H1 some distance d away. Let $\boldsymbol{x}_0$ be the base coordinate of the $d\mathbf{u}$ vector on the hyperplane, and $\boldsymbol{z}_0$ be the tip coordinate, which must line on the hyperplane H1. Therefore, the distance vector is

$$\mathbf{z}_0 = \mathbf{x}_0 + d\mathbf{u} \tag{20}$$

The fact that $\boldsymbol{z}_0$ is on H1 means that

$$\mathbf{w} \bullet \mathbf{z}_0 = +1 \tag{21}$$

Substituting $\boldsymbol{z}_0$ from Equation (20) yields

$$\mathbf{w} \bullet (\mathbf{x}_0 + d\mathbf{u}) = +1 \tag{22}$$

Substituting **u** from Equation (7) yields

$$\mathbf{w} \bullet \left(\mathbf{x}_0 + d \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) = +1 \tag{23}$$

Expanding Equation (23) yields

$$\mathbf{w} \bullet \mathbf{x}_0 + d \frac{\mathbf{w} \bullet \mathbf{w}}{\|\mathbf{w}\|} = +1 \tag{24}$$

Given that

$$\mathbf{w} \bullet \mathbf{w} = \|\mathbf{w}\|^2 \tag{25}$$

Equation (24) becomes

$$\mathbf{w} \bullet \mathbf{x}_0 + d\|\mathbf{w}\| = +1 \tag{26}$$

Hence,

$$\mathbf{w} \bullet \mathbf{x}_0 = 1 - d\|\mathbf{w}\| \tag{27}$$

The fact that $\boldsymbol{x}_0$ is on H0 means that

$$\mathbf{w} \bullet \mathbf{x}_0 = 0 \tag{28}$$

Substituting Equation (28) into Equation (27) yields

$$0 = 1 - d\|\mathbf{w}\| \tag{29}$$

Solving for d yields

$$d = \frac{1}{\|\mathbf{w}\|} \tag{30}$$

Therefore, the margin, which is the distance between H1 and another hyperplane H2 that is equidistant from H0 on the opposite side must be 2/||**w**||. Hence the margin is

$$m = \frac{2}{\|\mathbf{w}\|} \tag{31}$$

Note again that using the class labels of +1 and -1 as the decision boundaries for vectors above and below H1, respectively, allowed for a simple derivation of the margin in terms of only the length of the **w** vector. From Equation (31) the margin maximization goal is equivalent to minimizing the length of the **w** vector.

The SVM problem now becomes one of minimizing the norm of **w** subject to the constraint of Equation (19) that keeps feature vectors outside the margin. That is,

$$\begin{aligned} &\min_{f(w)} \|\mathbf{w}\| \\ &s.t. \qquad g: y_i(\mathbf{w} \bullet \mathbf{x}_i + w_0) \geq 1, \quad 1 \leq i \leq n. \end{aligned} \tag{32}$$

The optimization process of finding the best **w** can be visualized in 2D space as moving around a rectangle by changing its orientation and position to find a spot where it is the widest it can be with no points within it. Once the solution produces the optimal vector **w**, the hyperplane becomes defined and the classifier becomes

$$\mathbf{x} \mapsto \text{signum}(\mathbf{w} \bullet \mathbf{x}) \tag{33}$$

That is, given a feature vector **x**, the class label is the sign of the dot product of the **w** vector and the presented feature vector **x**. The signum function produces -1 if its argument is negative (less than zero), 0 if its argument is equal to 0, and +1 if its argument is positive (greater than zero). Note that the comparison is with zero, not +1 or -1. That is, the hyperplane is **w**•**x** = 0, thus either side of it will be either less than 0 or greater than zero.

The dot product is a measure of similarity. Intuitively, the SVM produces a larger absolute value for vectors with a direction that approaches the perpendicular to the hyperplane (the direction of **w**) and smaller absolute values for vectors that approach the parallel of the hyperplane. That is, the latter vectors will be closer to the hyperplane, which is the decision boundary, making the class decision less certain when there is noise or variations in the features.

SVM Theory

Quadratic Primal Form

Practical implementations of SVM recast the minimization problem of Equation (32) as a ***quadratic*** programming problem with constraints. In particular, the problem becomes

$$\begin{aligned} &\min_{f(w)} \frac{1}{2}\|\mathbf{w}\|^2 \\ &s.t. \qquad g: y_i(\mathbf{w} \bullet \mathbf{x}_i + w_0) - 1 = 0, \quad 1 \leq i \leq n \end{aligned} \tag{34}$$

Note that the squaring of the length of the **w** vector produces a parabola with a single minima. The ½ factor is for the convenience of cancellation when taking the partial derivative with respect to **w**. This is called the ***primal*** problem. It requires learning a large number of parameters in the **w** feature space.

Lagrangian Dual Form

The ***dual*** problem results in some beneficial properties that will aid in the computation, including the use of Kernel functions to solve non-linearly separable data. The dual problem requires learning only the number of support vectors, which can be much fewer than the number of feature space dimensions. The dual problem is found by constructing the Lagrange (see the appendix), by combining both the objective function and the equality constraint function. The objective function of the Lagrange formulation is

$$f \equiv \frac{1}{2}\|\mathbf{w}\|^2 \tag{35}$$

and the constraint function is

$$g \equiv y_i\left(\mathbf{w} \bullet \mathbf{x}_i + w_0\right) - 1 = 0 \tag{36}$$

Therefore, the Lagrange formulation is

$$L(\alpha, \mathbf{w}, w_0) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i \left[y_i\left(\mathbf{w} \bullet \mathbf{x}_i + w_0\right) - 1\right], \quad \alpha_i \geq 0 \quad \forall i \tag{37}$$

where the α_i parameters are the Lagrange multipliers. Note that the Lagrange is a function of only the **w** and the α_i parameters because the feature vectors $\mathbf{x}_i$ and labels y_i are known from the data. However, when examining the gradient of the Lagrange as a function of **w**, the α_i parameters are considered fixed. Expanding the summation yields

$$L(\alpha, \mathbf{w}, w_0) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1} \alpha_i y_i\left(\mathbf{w} \bullet \mathbf{x}_i + w_0\right) + \sum_{i=1} \alpha_i, \quad \alpha_i \geq 0 \quad \forall i \tag{38}$$

The solution for the Lagrange is at the point where the gradient, as a function of **w**, is zero. That is,

$$\nabla L(\mathbf{w}, w_0) = 0 \tag{39}$$

This requires taking the partial derivatives and setting them equal to zero, then solving the resulting simultaneous equations. In this formulation, there is only one variable **w**. Differentiating with respect to **w** yields

$$\frac{\partial}{\partial w} L(\mathbf{w}, w_0) = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \tag{40}$$

Which implies that the solution is

$$\mathbf{w} = \sum_{i=1} \alpha_i y_i \mathbf{x}_i, \quad \alpha_i \geq 0 \quad \forall i \tag{41}$$

Taking the partial derivative with respect to the bias component w_0 yields

$$\frac{\partial}{\partial w_0} L(\mathbf{w}, w_0) = \sum_{i=1} \alpha_i y_i = 0 \tag{42}$$

Hence the constraints are

$$\sum_{i=1} \alpha_i y_i = 0, \quad \alpha_i \geq 0 \tag{43}$$

This solution is also given by Representer Theorem (Schölkopf, Herbrich and Smola 2001). The key feature of Equation (41) is that $\alpha_i = 0$ zero for many of the vectors, except the support vectors. Unlike other methods such as artificial neural networks, there are no local minima in the optimization problem.

Given the solution for **w**, the classifier is found by substituting **w** into Equation (33), yielding

$$\mathbf{x} \mapsto \text{signum}\left(\sum_{i=1} \alpha_i y_i (\mathbf{x}_i \bullet \mathbf{x}) \right) \tag{44}$$

This Lagrange reformulation of the optimization problem produces an equivalent classifier that is more efficient because it operates directly on the known support vectors $\mathbf{x}_i$ and their associated known class. As described later, the dot product also facilitates the direct use of Kernel functions that can transform non-linear feature spaces to other (higher dimensional) spaces that are amenable to data separation by linear hyperplanes.

Slack Variables

Data may not be completely separable in some cases. That is, some points will lie within the margin or they may be incorrectly classified. Therefore, SVM implementations allow for some slackness in the classification by introducing an adjustable cost penalty *C*. The design is such that if $C = 0$, all errors are allowed. If $C \to \infty$ then no errors are allowed. SVM users generally set *C* to some finite number between 0 and infinity, with $C > 0$. The classifier with slackness is

$$\begin{aligned} \mathbf{w} \bullet \mathbf{x}_i &\geq +1 - \varepsilon_i \quad \text{for } y_i = +1 \\ \mathbf{w} \bullet \mathbf{x}_i &\leq -1 + \varepsilon_i \quad \text{for } y_i = -1 \end{aligned} \tag{45}$$

The cost function to optimize then becomes

$$J = \|\mathbf{w}\|^2 + C \sum_i \varepsilon_i^k \tag{46}$$

C determines the contributions of the slack variables relative to the influence of $\|\mathbf{w}\|$. The power *k* is an integer, normally set to $k = 1$ by SVM implementations.

Lagrangian Trick

Using the Lagrangian duality, the optimization is reformulated as

$$\begin{aligned} \max_{\alpha} \; & L(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i=1}\sum_{j=1}\alpha_i\alpha_j y_i y_j \mathbf{x}_i \bullet \mathbf{x}_j \\ s.t. \quad & \alpha_i \geq 0 \\ & \sum_{i=1} \alpha_i y_i = 0 \end{aligned} \tag{47}$$

The derivation is found by substituting **w** from Equation (41) and the constraint from Equation (43) into the Lagrangian Equation (38). The substitutions yield

$$L(\alpha) = \sum_i \alpha_i + \frac{1}{2}\left\|\sum_i \alpha_i y_i \mathbf{x}_i\right\|^2 - \sum_i \alpha_i y_i \left(\sum_i \alpha_i y_i \mathbf{x}_i \bullet \mathbf{x}_i\right), \quad \alpha_i \geq 0 \quad \forall i \tag{48}$$

The equality to zero constraint eliminated some terms. The last remaining term is

$$\sum_i \alpha_i y_i \left(\sum_i \alpha_i y_i \mathbf{x}_i \bullet \mathbf{x}_i\right) = \sum_i \alpha_i y_i \mathbf{x}_i \left(\sum_j \alpha_j y_j \mathbf{x}_j\right) = \|\mathbf{V}\|^2 \tag{49}$$

The arbitrary assignment to vector **V** is made to simplify the algebra. That is

$$V = \sum_i \left(\alpha_i y_i \mathbf{x}_i\right) \tag{50}$$

and

$$\|V\| = \sqrt{\sum_i \left(\alpha_i y_i \mathbf{x}_i\right)^2} \tag{51}$$

Note that squaring removes the square-root. Subsequently,

$$L(\alpha, x, y) = \sum_i \alpha_i + \frac{1}{2}\|V\|^2 - \|V\|^2 = \sum_i \alpha_i - \frac{1}{2}\|V\|^2 \tag{52}$$

Expanding the second term (the norm-squared vector) yields

$$\|V\|^2 = \left\|\sum_i \alpha_i y_i \mathbf{x}_i\right\|^2 = \left\{\sum_i \alpha_i y_i \mathbf{x}_i\right\}^T \left\{\sum_j \alpha_j y_j \mathbf{x}_j\right\} = \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j = \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \bullet \mathbf{x}_j \tag{53}$$

The expression is now,

$$L(\alpha, x, y) = \sum_{i=1} \alpha_i - \frac{1}{2}\sum_{i=1}\sum_{j=1}\alpha_i\alpha_j y_i y_j \mathbf{x}_i \bullet \mathbf{x}_j, \quad \alpha_i \geq 0 \quad \forall i \tag{54}$$

The optimization problem now becomes

$$\begin{aligned} \max_{\alpha} \; & L(\alpha) = \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i=1} \sum_{j=1} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \bullet \mathbf{x}_j \right) \\ s.t. \quad & \alpha_i \geq 0 \\ & \sum_i \alpha_i y_i = 0 \end{aligned} \tag{55}$$

This is called the **dual** problem. It has become one of maximization. Rather than solving for **w** under the classification constraints, the problem becomes one of learning the α_i parameters given the features and their classifications. The non-zero parameters are those of the support vectors. The dot product of the feature vectors also facilitate easy computation and can leverage the MAC operation of DSPs.

Inclusion of the slack variables requires forming the Lagrangian to include both the new objective, Equation (46), and new constraint functions, Equation (45). Setting the partial derivatives of the modified Lagrangian to zero, solving for **w** and w_0, and substituting them back into the Lagrangian yields (as was done before with details left as an exercise), the general SVM solution becomes

$$\begin{aligned} \max_{\alpha} \; & L(\alpha) = \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i=1} \sum_{j=1} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \bullet \mathbf{x}_j \right) \\ s.t. \quad & 0 \leq \alpha_i \leq C \\ & \sum_i \alpha_i y_i = 0 \end{aligned} \tag{56}$$

Note that the only change is now in the constraint of the Lagrange multipliers. The constraint now has an upper bound of *C*. The generalization is that setting C = ∞ results in the original solution. Setting C closer to zero results in a slack margin to classify more of the data at the expense of misclassification. SVM users generally determine *C* heuristically or by cross validation (covered later).

The Kernel Trick

A transformation function Φ(**x**) maps the input vector to a higher dimensional (new vector) by operating on one or more combinations of the lower dimension features. For SVMs, the transform does not introduce new variables for the higher dimensions. Instead, it introduces new parameters that operate on the lower dimension variables of the feature space to produce the dimensions of the higher feature space. For example, the exponential transform creates infinite feature dimensions because the exponential is an infinite series that operates on one dimension *z* such that

$$\exp(z) = \sum_{k=0}^{\infty} \frac{z^k}{k!} \tag{57}$$

Subsequently, a kernel function *K*(**x**) takes a dot product of the higher dimension feature vectors such that

$$K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \Phi\left(\mathbf{x}_i\right) \bullet \Phi\left(\mathbf{x}_j\right) \tag{58}$$

The dot product of the transformed vectors produces a scalar value. The ***trick*** part is that the kernel does not actually compute the right side of Equation (58) by first transforming each feature vector and then taking their dot products. Rather, the kernel effectively achieves the same result by operating only on the vectors in their ambient space, without doing any transforms into a higher dimension space. This "trick" saves computing time and lowers memory requirements.

Worked Example

Let $\Phi(\mathbf{x})$ be a transform that maps from 2D to 3D space such that

$$\Phi(x_1, x_2) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) \tag{59}$$

The new hyperplane will be

$$\mathbf{w} \bullet \mathbf{x} = w_0 + w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 = 0 \tag{60}$$

which is an ellipse. Figure 3a is a plot of the transformation function in 3D space with $w_0 = 0$ and $\mathbf{w} = [1, 1, 1]$. Figure 3b plots some possible hyperplanes in the 2D space. The possible hyperplanes are projections of the contour lines of the 3D transform function onto the 2D space. The contour lines are at the intersections of the function with linear 2D hyperplanes. The projected hyperplane in 2D space forms a non-linear decision boundary to separate the data in the original 2D feature space.

The following example illustrates the derivation of a kernel function for the polynomial mapping of a vector from 2D to 3D space. Let $\Phi(\mathbf{r})$ be the mapping function of vector $\mathbf{r}$ such that

$$\Phi(\mathbf{r}) = \left(r_1^2, \sqrt{2}r_1r_2, r_2^2\right) \tag{61}$$

The dot product of the two transforms yield a scalar such that

$$\Phi(r_1, r_2) \bullet \Phi(x_1, x_2) = r_1^2x_1^2 + 2r_1r_2x_1x_2 + r_2^2x_2^2 \tag{62}$$

This is identical to

$$\Phi(r_1, r_2) \bullet \Phi(x_1, x_2) = (r_1x_1)^2 + 2(r_1x_1)(r_2x_2) + (r_2x_2)^2 = (r_1x_1 + r_2x_2)^2 = (\mathbf{r} \bullet \mathbf{x})^2 \tag{63}$$

Hence, the kernel function is

$$K(\mathbf{r}, \mathbf{x}) = \Phi(\mathbf{r}) \bullet \Phi(\mathbf{x}) = (\mathbf{r} \bullet \mathbf{x})^2 \tag{64}$$

This outcome shows that the kernel function produces a scalar by involving only a dot product of the feature vectors, which is equivalent to computing a dot product of transformed feature vectors in a higher dimensional space. Intuitively, vectors transformed into a higher feature space by using some combination of the lower dimensions will be similar to their lower dimension counterparts. The dot product is a measure of similarity in both the ambient space of the Kernel operation and in the higher dimension space. This is the reason that the kernel trick works.

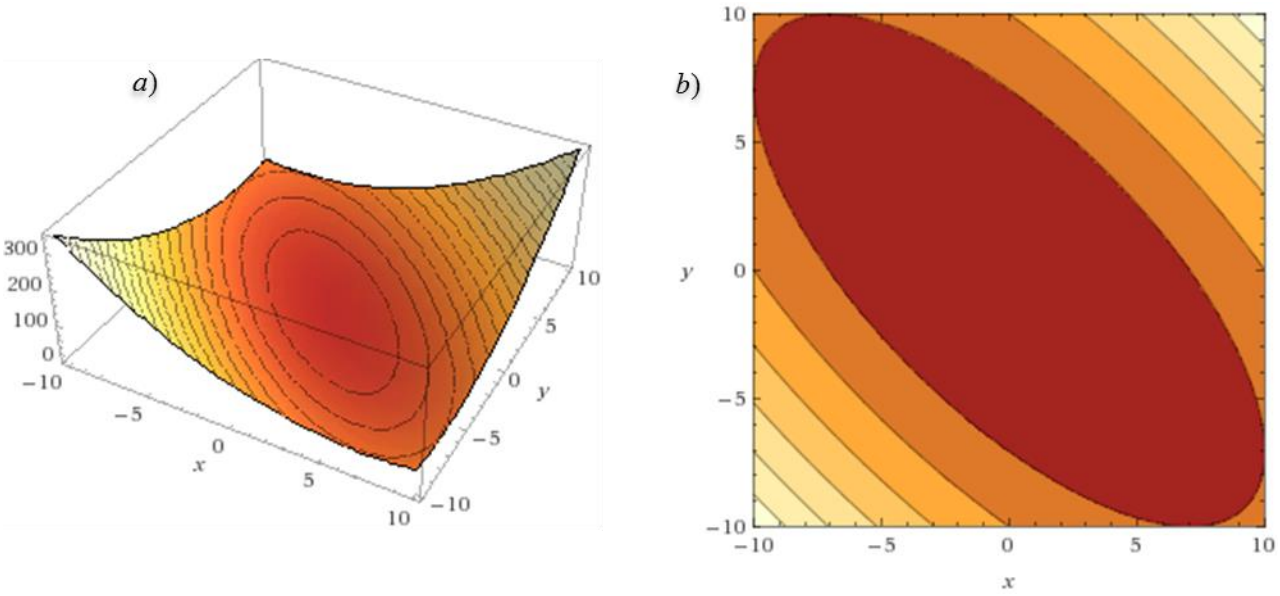

Figure 3: a) Function in 3D space and b) hyperplanes in 2D space.

Theoreticians have worked out many different types of kernels. Some provide better results for one application relative to another. Therefore, it is important to evaluate different kernel functions and their parameters when attempting classification problems with SVMs. Selecting the Kernel function and tuning their parameters may involve techniques such as K-Fold Cross Validation.

The following are a few types of Kernels:

Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|}{2\sigma^2}\right) \tag{65}$$

Polynomial

$$K(\mathbf{u},\mathbf{v}) = (\gamma \cdot \mathbf{u} \bullet \mathbf{v} + r)^d \tag{66}$$

Radial basis function (RBF)

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\gamma \cdot \|\mathbf{u}-\mathbf{v}\|^2\right) \tag{67}$$

After applying the Kernel the training problem becomes

$$\begin{aligned} &\max_{L(\alpha)} \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &s.t. \quad \alpha_i \geq 0, \quad i = 1,\ldots,m \\ &\quad \sum_i \alpha_i y_i = 0 \end{aligned} \tag{68}$$

and the classification becomes

$$\mathbf{x} \mapsto \text{signum}\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right) \tag{69}$$

That is, both training and classification must use the same kernel function.

Evaluating Classifier Performance

Classifiers often report a set of performance parameters such as accuracy, precision, and recall (Powers 2007). Let the following variables be defined as:

TP = True Positive, *TN* = True Negative
FP = False Positive (false alarm, Type I error)
FN = False Negative (a miss, Type II error)
P = Number of Real Positives in the data
N = Number of Real Negatives in the data

Confusion Matrix

A **confusion matrix** is a table that summarizes the performance of the classifier as follows:

n=100	Predicted Negative	Predicted Positive	
Actual Negative	TN = 40	FP = 10	TN + FP = N = 50
Actual Positive	FN = 5	TP = 45	FN + TP = P = 50
	TN + FN = 45	FP + TP = 55	

A good classifier will have large numbers in the left diagonal of the matrix.

The **accuracy** of a classifier is

$$a = \frac{TP + TN}{P + N} = \frac{TP + TN}{(TP + FN) + (TN + FP)} \tag{70}$$

The **error rate** of misclassification rate is 1 - *a*. The **specificity** is 1 - *FP*, and it is equivalent to the true negative rate.

The **precision** *p* of classification for each class is a measure of its prediction accuracy for that class. Hence, for the positive class, it is the number of correct positives divided by the number of all positives predicted for that class, even if some of them are false positives. That is, *p* is

$$p = \frac{TP}{TP + FP} \tag{71}$$

Precision, therefore, provides a relative comparison of the classifier performance among the different classes. That is, the classifier be an unbalanced performer by doing better at prediction for one class versus another class.

The **recall** *r* is the number of correct positive results divided by the number of positive results that should have been returned.

$$r = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{72}$$

The **F_1 score** is the harmonic mean of precision and recall such that

$$F_1 = 2 \times \frac{p \cdot r}{p + r} \tag{73}$$

It is interpreted as a weighted average of the precision and recall (best is 1 and worst is 0). Note that F_1 is 1 if $p = r = 1$ and zero if either p or r is zero.

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{74}$$

ROC Curve

ROC stands for Receiver Operating Characteristic. It is often confused with the term region of convergence. It is a graphical plot of True Positive Rate (sensitivity) from the positive samples versus False Positive Rate (false alarm) from the negative samples, as a function of a parameter of the classifier (Fawcett 2006). Often, the parameter is the location of the decision boundary. The sensitivity is plotted on the y-axis and the false alarm is plotted on the x axis.

As an example, one can also think of the predictions of a receiver in terms of correctly predicting the transmission of a 0 (a positive) or 1 (not-zero or a negative). Hence, transmissions of a 0 that classifies as positive results in a TP, and if classified as a 1 (i.e. negative), that would be a FN. Similarly, transmissions of a 1 (negative) that classifies as a 1 is a TN, and if classified as a 0 (positive), that would be a FP.

Figure 4 provides an illustration for deriving the ROC, given the distributions of two classes that represent the probability of receiving zeros and ones as a function of a feature value. In this example, the feature value is the voltage level received. The feature is on the x-axis and the probability of reception is on the y-axis. The feature threshold T is set such that the classifier (receiver) predicts the transmission was a 0 for received voltages below T, and a 1 for voltages above. Some of the voltages that exceed the threshold (prediction is 1) are actually transmissions of 0, and hence the classification results in a false negative (FN) because the prediction is a negative for 0 when it was actually a zero being transmitted. Similarly, some of the voltages that fall below the threshold (prediction of 0) are actually transmissions of 1, and the classification is a false positive (FP) because a one is actually being transmitted.

The amount of overlap of the two distributions is proportional to the amount of noise in the transmission channel. A low noise channel will have a clear separation of the voltage distributions for a 1 and a zero. Hence, a decision threshold placed in the center of the gap will result in high TP and high TN.

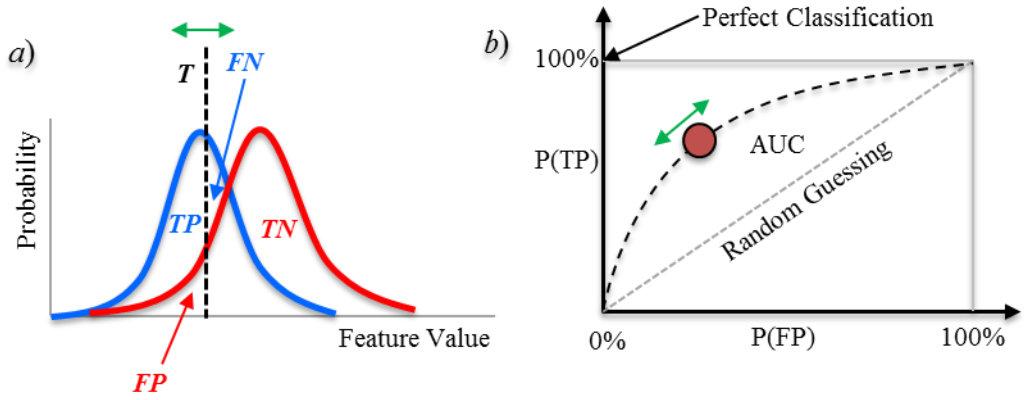

Figure 4: Determination of the ROC curve for a classifier.

The amount of overlap affects the shape of the curve. No overlap moves the curve towards the coordinate of perfect classification (0, 1). Full overlap moves the curve towards the diagonal. The performance points will fall on the diagonal for random guessing. Performance below the diagonal is worse than random guessing. Given some amount of overlap, moving the threshold moves the point along the ROC curve. The goal of classifier design is to maximize the area under the curve (AUC), and move the operating point towards the point of perfect classification.

Classifier Tuning and Validation

Several techniques are available for evaluating the prediction performance of classifiers, and subsequently to tune them by selecting the appropriate hyper-parameters such as the value of C, the Kernel, and the Kernel parameters. Conventional validation splits the dataset into a training and a validation subset. Typically the classifier trains on 70% of the data and the remaining 30% is used to validate its performance. This is also called **leave-p-out** cross-validation because it involves using *p* observations as the validation set, and the remaining data as the training set.

Another technique called **cross-validation** evaluates the potential generalized performance of a classifier on an independent dataset by segmenting the available dataset to create training and validation data subsets, and then rotating the evaluations among the subsets. The results of multiple evaluations are averaged to reduce the variability of the performance evaluation. Cross-validation is used when the data set is small.

K-fold cross-validation is a generalization that partitions the data into a validation set, and *k*-1 training sets. The partitions (subsets) are created by randomly selecting data objects and placing them into each subset such that they have approximately equal sizes. The *k*-1 training subsets are used to create *k*-1 SVMs for testing against the validation set. The average of a performance metric such as prediction accuracy is the output of each cross validation round. K-fold cross validation is when the process is repeated K times such that each partition (subset) is used exactly once. The various hyper-parameter selections are varied with each K-fold cross validation regime to tune the SVM design.

References

Bhavsar, Hetal, and Amit Ganatra. 2012. "Variations of Support Vector Machine classification Technique: A survey." *International Journal of Advanced Computer Research* 2 (6): 230-236.

Dawkins, Paul. 2017. *Paul's Online Math Notes.* Accessed September 1, 2017. http://tutorial.math.lamar.edu/Classes/CalcII/EqnsOfPlanes.aspx.

Fawcett, Tom. 2006. "An introduction to ROC analysis." *Pattern Recognition Letters* (Elsevier) 27 (8): 861-874. http://people.inf.elte.hu/kiss/11dwhdm/roc.pdf.

Powers, David M. W. 2007. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation.* SIE-07-001, School of Informatics and Engineering , Adelaide, Australia: Flinders University of South Australia, 24. http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf.

Schölkopf, Bernhard, Ralf Herbrich, and Alex J. Smola. 2001. "A Generalized Representer Theorem." *International Conference on Computational Learning Theory* (Springer) 2111: 416-426. doi:10.1007/3-540-44581-1_27.

Appendix

This appendix provides some background material on the various techniques used for the derivation of SVMs.

A. Optimization Problems with Constraints

A.1. Problem Definition

The standard form is:

minimize
x $f(x)$
subject to $g_i(x) = 0,\ i = 1, \ldots, p$
$h_i(x) \leq 0,\ i = 1, \ldots, m$

x is the optimization variable
$f(x)$ is the objective or cost function
$g_i(x)$ is one of p equality constraints
$h_i(x)$ is one of m inequality constraints

The goal is to find x^* for which the function f is at a minimum without violating any of the constraints. A condition to check is that all of the constraints can be true simultaneously, that is, there are no conflicts.
Another way to state the standard form is:

$$x^* = \inf\{ f(x) \mid g_i(x) = 0,\ i = 1, \ldots, p,\ h_i(x) \leq 0,\ i = 1, \ldots, m\}$$

inf is the **infimum**, which means the greatest lower bound. The dual is **supremum**, which is the lowest upper bound.

A.2. Lagrange Multipliers

The Wikipedia definition is "in mathematical optimization, the method of Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to **equality** constraints." Hence, this method does not work for inequality constraints but forms the basis for working with them.

The gradient of a function is projected as a vector field with the arrow pointing in the direction where the function is increasing. Figure 5 provides an example illustration. The arrows are perpendicular to a contour line. Lagrange found that the minimum of the cost function under the quality constraint function is when the gradients of both functions point in the same direction. That is

$$\nabla f(x,y) = \lambda \nabla g(x,y) \tag{75}$$

Lamda is the Lagrange multiplier that allows the gradients to have different lengths, even when they point in the same direction. In fact, this formula allows the gradients to be parallel so that both maxima and minima can be found.

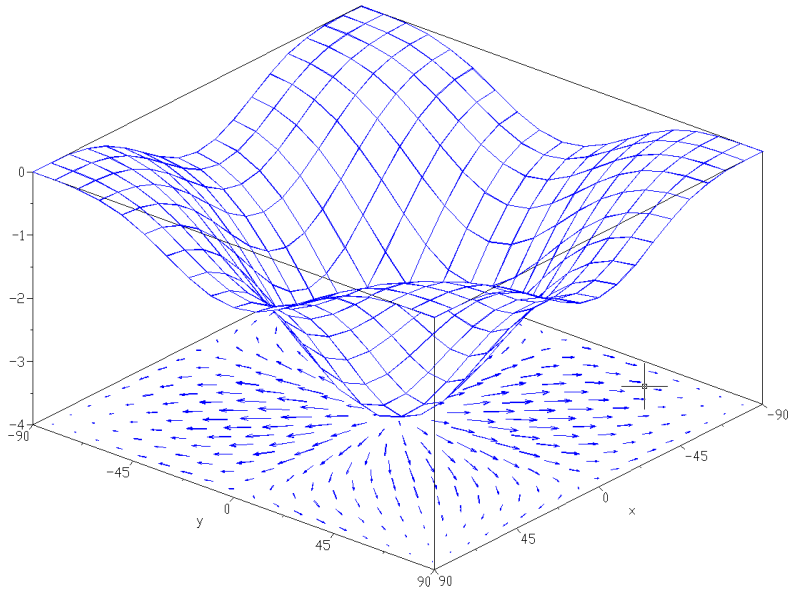

Figure 5: Projection of gradient vector fields (Source: Dr. D. Joyce, Clark University).

Rewriting Equation (75) and assigning it to a function (the Lagrangian) gives

$$\nabla f(x,y) - \lambda \nabla g(x,y) = 0 = \nabla L(x,y,\lambda) \tag{76}$$

Solving $\nabla L(x,y,\lambda) = 0$ means finding the points x, y, and λ where the gradient of f and g are parallel. The general form for the Lagrange to include multiple constraints is

$$L(x,\alpha) = f(x) + \sum_i \alpha_i g_i(x) \tag{77}$$

A.3. Dot Product

The dot product of two vectors **x** and **y** is

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x} \bullet \mathbf{y} = \|\mathbf{x}\| \times \|\mathbf{y}\| \times \cos(\theta) = \sum_i x_i y_i = \left| \mathbf{x}\mathbf{y}^T \right| = \alpha \tag{78}$$

which is a scalar. It is derived by taking the product of their magnitudes and the cosine of the angle between them. The vectors must have and equal number of elements.

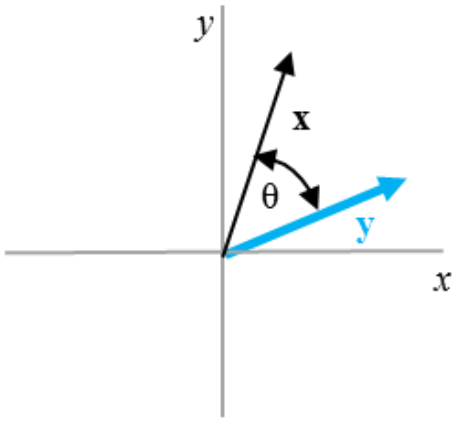

Figure 6: Dot product between two vectors.

The significance is that it is a measure of their similarity. If they are orthogonal, cos(90) = 0 and the dot product is 0. Alternatively, if the angle is 0, then the cos(0) = 1 and the product of their lengths become the measure of similarity.