Kernel Methods and SVMs Extension

The purpose of this document is to review material covered in Machine Learning 1 - Supervised Learning regarding support vector machines (SVMs). This document also provides a general overview of some extensions to that which were described in the course, including non-binary classification and support vector regression.

We will introduce the concept of SVMs using the simplest case for application. Consider a scenario where we have data that must be classified into two different groups. If the data are *linearly separable*, or in other words, can be separated completely into their groups by a dividing hyperplane, then our goal is to find the equation of the hyperplane that best divides the groups.

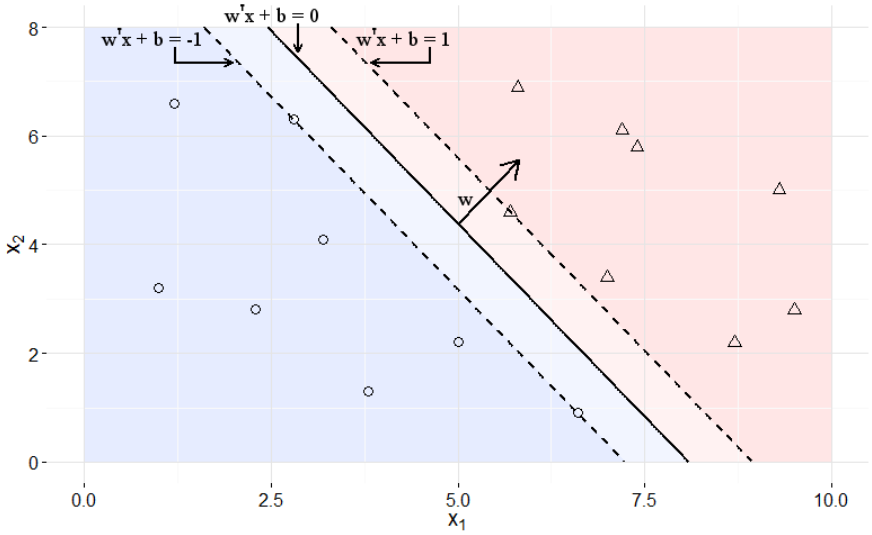
To be more formal with the problem description, we label the classes for each of the data points x_i as being -1 or 1, i.e. $y_i \in \{ -1, 1 \}$. Our hyperplane function has the equation $w^T x + b$ and is defined such that for all points that have a class $y_i = -1$,

$$w^T x_i + b \leq -1$$

and for points with a class $y_i = 1$,

$$w^T x_i + b \geq 1.$$

In our training data, we should have no points in between the hyperplanes $w^T x + b = -1$ and $w^T x + b = 1$, a region called the 'margin'. The dividing plane is the function $w^T x + b = 0$ and we classify new points by their sign: $\hat{y}_i = sign(w^T x + b)$.



(Note that w does not look perpendicular due to difference in x and y-axis scaling)

There are many choices of our parameter vector w that allow us to separate the data, but some are clearly better than others. Ideally, we want to select parameters for the hyperplane that maximize the size of the margin.

Consider two points that lie on opposite margins, x_+ and x_-, that are as close as possible to one another. In this case, the vector connecting these two lines will be perpendicular to the hyperplanes defining the margin.

$$w^T x_+ + b = 1 \text{ and } w^T x_- + b = -1,$$

Subtracting the two equations generates $w^T(x_+ - x_-) = 2$. Since the vectors w and $x_+ - x_-$ are parallel, $||w||*||x_+ - x_-|| = 2$, where $||v||$ is the magnitude/length of a vector v. Dividing $||w||$ on both sides gives the distance between the hyperplanes $||x_+ - x_-||$ which is equal to $2 / ||w||$ From here, we observe that maximizing the size of the margin is equivalent to finding the minimum $||w||$ that maintains the relationship $y_i(w^T x_i + b) \geq 1$ for all points in the training data. (Recall that $w^T x_i + b \geq 1$ when $y_i = 1$ and $w^T x_i + b \leq -1$ when $y_i = -1$.)

We approach solving the problem by noting that minimizing $||w||$ is equivalent to minimizing $\frac{1}{2}||w||^2$, converting the problem into a quadratic programming optimization problem. The Lagrange multipliers α transform our optimization problem into one of maximizing the output of

$$w(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

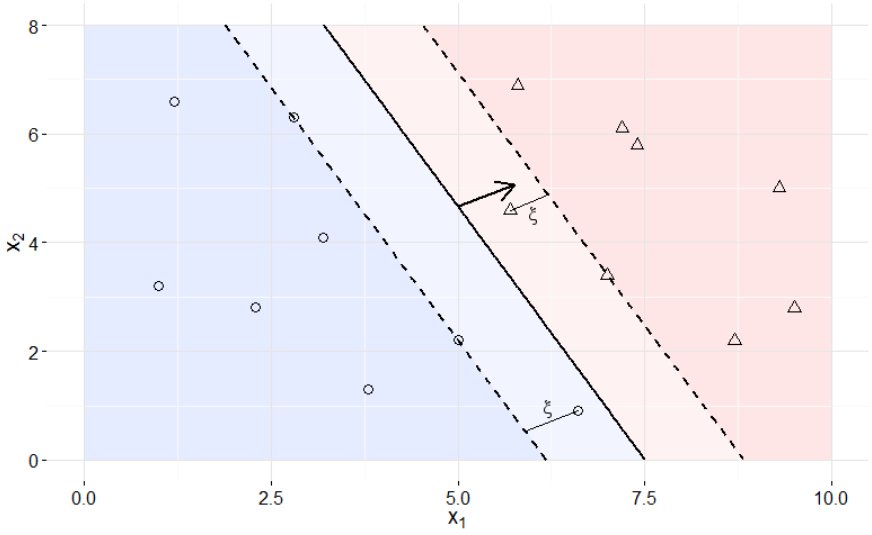
while satisfying the constraints that all $0 \leq \alpha_i$ and $\sum_i \alpha_i y_i = 0$. To provide some context for interpreting this, think of the multipliers α as weights on data points. From the constraint $\sum_i \alpha_i y_i = 0$, the sum of the weights on the points categorized as $y_i = -1$ should be equal to those categorized as $y_i = 1$. As for w(α), the second term controls the summed weights in the first term from getting too large. The second term takes into account the categories of each pair of points ($y_i y_j = 1$ if they are in the same class, -1 if they differ) and a measure of similarity (evoked by $x_i^T x_j$).

When we obtain the optimal Lagrange multipliers, it turns out that most of the weights α_i are equal to zero. The points that have non-zero weight are the only points that contribute to the calculation of w, and all in fact fall on the margin, satisfying $y_i(w^T x_i + b) = 1$. These points are the *support vectors* for the model. We obtain the parameter values for our dividing hyperplane from $w = \sum_i \alpha_i y_i x_i$ and $b = w^T x_i - y_i$ for some point that lies on the margin.

The above describes the general process for computing SVMs for linearly separable data, but real-life datasets do not normally allow themselves to be divided so easily. Here, we discuss two ways to deal with non-linearly separable datasets and move beyond hard-margin SVMs. If we have data that is mostly linearly separable, we can consider using soft-margin SVMs, relaxing the criteria that all points are correctly classified. If we have data that is separable in a nonlinear fashion, we can consider using *kernel functions* to be able to capture a nonlinear dividing curve between classes. Typically, we make considerations of both kernel function and value of soft-margin parameter to perform classification tasks.

In a soft-margin SVM, we do not require the data to be completely linearly separable and allow for some points to be classified incorrectly. We provide for each point a non-negative slack variable ξ_i that illustrates to what degree each point is misclassified: $y_i(w^T x_i + b) \geq 1 - \xi_i$. If a point is classified correctly on its side of the margin, then $\xi_i = 0$. If a point gets placed within the margin or in the wrong-classed

region, then ξ_i takes on positive value proportional to the point's distance from its desired marginal hyperplane.



Our optimization problem now has to balance the size of the errors we make: our goal is to minimize $\frac{1}{2}||w||^2 + C\sum_i \xi_i$, where C is a regularization parameter that tells us the weight we want to put on misclassification errors. With smaller values of C, we punish errors less, thus increasing the size of the margin. Larger values of C result in narrower margins; the limit of C as it tends towards infinity is that any misclassification error is punished to an extent that we effectively have our original hard-margin SVM. When we convert the optimization problem into the form maximizing the output of

$$w(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j ,$$

the constraint that $\sum_i \alpha_i y_i = 0$ remains the same, while the other constraint now has an upper bound $0 \le \alpha_i \le C$. With the soft-margin SVM, our support vectors (points that have weight $0 < \alpha_i$) include not just points on the marginal hyperplanes, but also those points that are within the margin or are misclassified.

For data that is separable, but not linearly, we can use a kernel function to capture a nonlinear dividing curve. The kernel function should capture some aspect of similarity in our data; it also

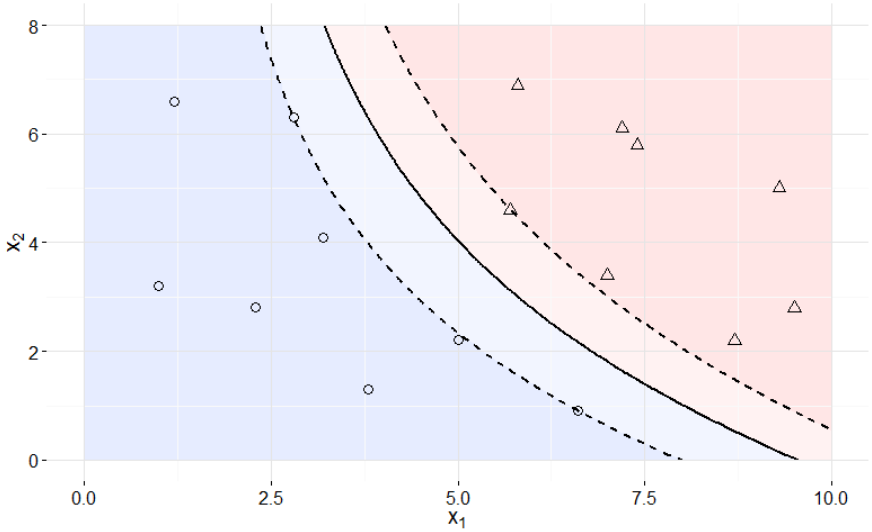
represents domain knowledge regarding the structure of the data. In general, we can write the function we want to maximize as

$$w(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j).$$

In our original, linear SVM, our kernel function was $k(x_i,x_j) = x_i^T x_j$ and suggested a dividing hyperplane. The kernel function $k(x_i,x_j) = (x_i^T x_j)^2$ generates a dividing hypersphere, while $k(x_i,x_j) = (x_i^T x_j + c)^d$ is the general form for polynomial kernels. With kernel functions, we can project the data into a transformed space where a dividing hyperplane can be found, but when plotted in the original feature space ends up being a non-linear dividing curve. In order to compute the class of a new instance, we now utilize the sign of the output

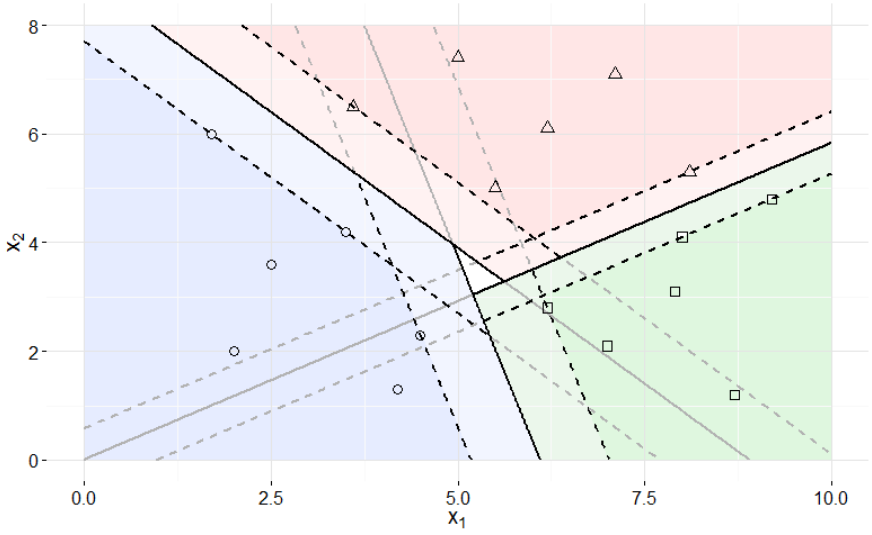
$$\sum_i \alpha_i y_i k(x_i, x) + b.$$

Since most of the weights are equal to zero, this is still a fairly quick computation compared to the linear case.



It is important to note that the natural task for SVMs lies in binary classification. For classification tasks involving more than two groups, a common strategy is to use multiple binary classifiers to decide on a single-best class for new instances. For example, we may create one classifier for each class in a one-versus-all fashion then, for new points, classify them based on the classifier function that produces the largest value. Alternatively, we can set up classifiers for all

pairwise comparisons and select the class that ‘wins’ the most pairwise matchups for new points.

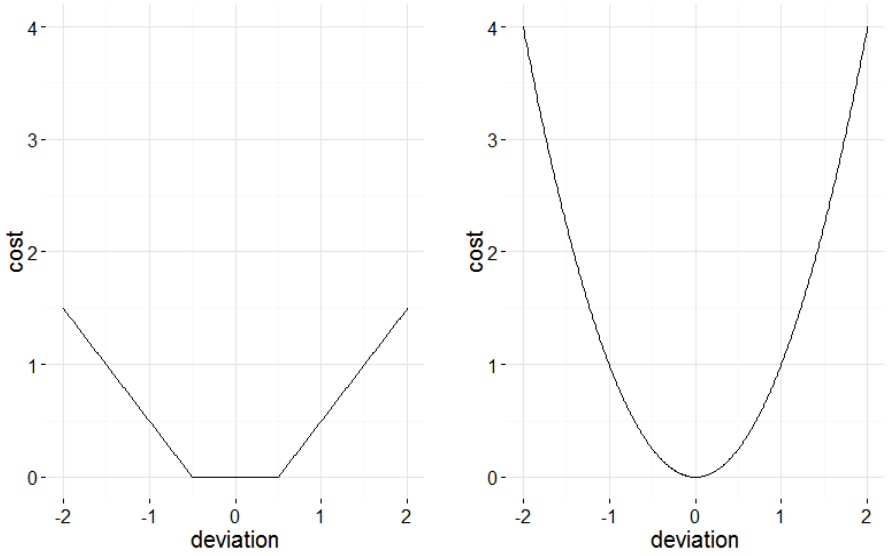


(Figure depicts pairwise matchups approach. Gray lines indicate where a binary classifier has no effect. Note central area where no class has dominance.)

We can also extend SVMs to regression tasks, or support vector regression (SVR). As with SVMs, we project data in an SVR task using a kernel function so that they can be fit by a hyperplane. Instead of dividing the data into classes, however, the hyperplane now provides an estimate for the data’s output value. In addition, the margin and error are treated differently. A parameter ε is specified such that small deviations from the regression hyperplane do not contribute to error costs, i.e. when we attempt to minimize

$$\frac{1}{2}||w||^2 + C\sum_i \xi_i,$$

$\xi_i = 0$ when a point lies within the margin. Non-zero slack variable values are instead the (linear) distance beyond the ε-region that a point lies. Compare this to the quadratic error function that is found in standard linear regression tasks, where all deviations from the estimate count against the function’s fit, but errors are penalized by the quadratic difference from the estimate.



Essentially, however, SVR operates in much the same way as SVM does. For each point in the training data, we instead have two slack variables, ξ_i and ξ_i^*, one for positive deviations and one for negative deviations from the regression hyperplane. This results in two Lagrangian multipliers associated with each point, $0 \leq \alpha_i, \alpha_i^* \leq C$, and a respecified constraint on weight values $\sum_i (\alpha_i - \alpha_i^*) = 0$. When solved, the regression function takes the form

$$\sum_i (\alpha_i - \alpha_i^*)\, k(\mathbf{x}_i^T \mathbf{x}) + b.$$

As before, most of the weights take a value of zero, and for points with non-zero weights, at most one of α_i, α_i^* will be non-zero.