Ch. 9. Support Vector Machines

Support Vector Machines (SVM) have been shown to perform well in a variety of settings, and are often considered one of the best “out of the box” classifiers.
The support vector machine is a generalization of a simple and intuitive classifier called the **maximal margin classifier**. Though it is elegant and simple, we will see that this classifier can be applied only to linearly separable data sets. The **support vector classifier** is an extension of the maximal margin classifier that can be applied in a broader range of cases.
The **support vector machine** is a further extension of the support vector classifier that can accommodate non-linear class boundaries.
Support Vector Machines are intended for the binary classification setting in which there are two classes; the approach can be extended to more than two classes.

What is an Hyperplane?

In a p-dimensional space, a hyperplane is a flat affine (does not pass through the origin) subspace of dimension p - 1.
The mathematical definition of a hyperplane in two dimensions is:
$\square_0 + \square_1 X_1 + \square_2 X_2 = 0$
When we say that it defines the hyperplane, we mean that any $X = (X,X)^T$ for which the equation holds is a point on the hyperplane. Note that the equation is a line since indeed in two dimensions a hyperplane is a line.

We can extend the equation of a hyperplane to the p-dimensional setting:
$\square_0 + \square_1 X_1 + \dots + \square_p X_p = 0$

Now suppose that $X = (X_1, \dots, X_p)^T$ does not satisfy the equation; then:

$\square_0 + \square_1 X_1 + \dots + \square_p X_p > 0$ tells us that X lies on one side of the hyperplane
$\square_0 + \square_1 X_1 + \dots + \square_p X_p < 0$ tells us that X lies on the other side of the hyperplane
So we can think of the hyperplane as dividing the p-dimensional space into two halves.
One can easily determine on which side of the hyperplane a point lies by simply calculating the sign of the left side of the equation of the hyperplane.

Classification Using a Separating Hyperplane

Suppose that we have a n x p data matrix X that consists of n training observations in p-dimensional space: $x_1 = (x_{11} \dots x_{1p})^T , \dots , (x_{n1} \dots x_{np})^T$ and that these observations fall into two classes -- that is, $y_1, \dots, y_n \in \{-1, 1\}$ where -1 represents one class and 1 the other class.

We also have a test observation, a p-vector of observed features $x^* = (x_1^* \dots x_p^*)^T$.

Our goal is to develop a classifier based on the training data that will correctly classify the test observation using its feature measurements.

Suppose that it’s possible to construct a hyperplane that separates the training observations perfectly according to their class labels.
Then a separating hyperplane has the property that:
$\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p > 0$ if $y_i = 1$
and
$\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p < 0$ if $y_i = -1$
equivalently, a separating hyperplane has the property that
$y_i(\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p) > 0$ for all $i = 1, \ldots, n$.

If a separating hyperplane exists, we can use it to construct a very natural classifier: a test observation is assigned a class depending on which side of the hyperplane it is located.

That is we classify the test observation x^* based on the sign of $f(x^*) = \beta_0 + \beta_1 x^*_1 + \ldots + \beta_p x^*_p$.
If $f(x^*)$ is positive → we assign class 1 to x^*; if $f(x^*)$ is negative → we assign class -1 to x^*.

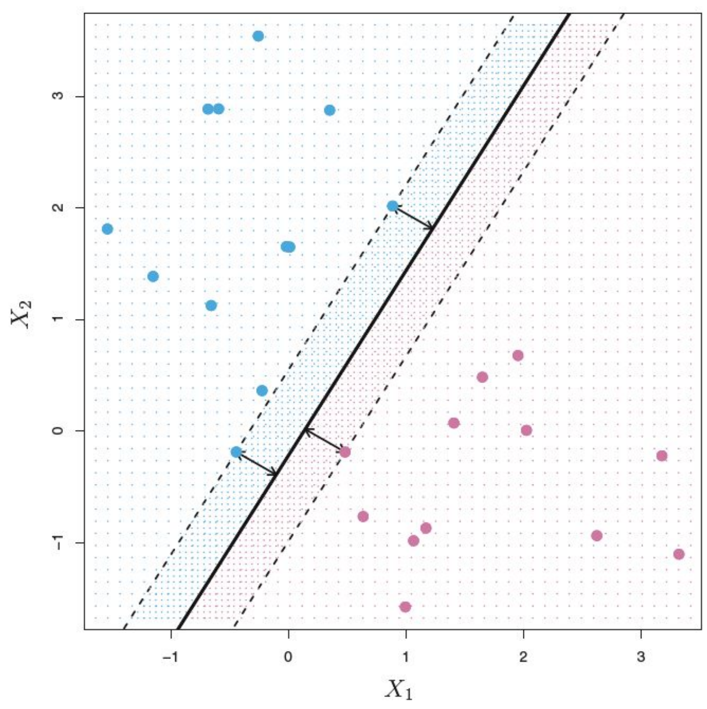
We can also make use of the magnitude of $f(x^*)$. If $f(x^*)$ is far from zero, then x^* lies far from the hyperplane, and so we can be confident about our class assignment for x^*. On the other hand, if $f(x^*)$ is close to zero, then x^* is located near the hyperplane, and so we are less certain about the class assignment.
Not surprisingly, a classifier that is based on a linear hyperplane leads to a linear decision boundary.

The Maximal Margin Classifier

In general, if our data can be perfectly separated using a hyperplane, then there will be an infinite number of such hyperplanes.
We must have a reasonable way to decide which of the infinite possible separating hyperplane to use.
A natural choice is the **maximal margin hyperplane** (also known as the optimal separating hyperplane), which is the separating hyperplane that is farthest from the training observations. That is, we can compute the (perpendicular) **distance** from each training observation to a given separating hyperplane; the smallest such distance is the minimal distance from the observations to the hyperplane and is known as the margin.

The maximal margin hyperplane is the sep. hyperplane for which the **margin is largest** -- that is minimal distance is bigger.
We hope that a classifier that has a large margin on the training data will also have a large margin on the test data, and hence will classify the test observations correctly.
Although the maximal margin classifier is often successful, it can also lead to overfitting if p is large.

In a sense, the maximal margin hyperplane represents the mid-line of the widest “slab” that we can insert between the two classes.



Examining the figure, we see that three training obs are equidistant from the maximal margin hyperplane and lie along the dashed lines indicating the width of the margin. These three observations are known as **support vectors**, since they are vectors in p-dimensional space and they support the maximal margin hyperplane in the sense that if these points were moved slightly, then the maximal margin hyperplane would move as well. **Interestingly, the maximal margin hyperplane depends directly on the support vectors, but not on the other observations**.

Construction of the Maximal Margin Classifier

Briefly, the maximal margin hyperplane is the solution to the optimization problem:

maximize M
$\beta 0$, $\beta 1$, ...βp, M

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$, $y_i(\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p) \geq M \; \forall \; i = 1, \ldots, n$.

$y_i(\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p) \geq M \; \forall \; i = 1$ guarantees that each observation will be on the correct side of the hyperplane, provided that M is positive.

$\sum_{j=1}^{p} \beta_j^2 = 1$ add meaning to the previous equation since one can show that with this constraint, the perpendicular distance from the ith observation to the hyperplane is given by $y_i(\square_0 + \square_1 x_1 + \ldots + \square_p x_p)$. Therefore, the two constraints ensure that each observation is on the correct side of the hyperplane and at least at a distance M from the hyperplane.

Hence, M represents the margin of our hyperplane, and the optimization problem chooses $\square_0, \square_1, \ldots, \square_p$ to maximize M.

The Non-separable Case

In many cases no separating hyperplane exists, and so there is no maximal margin classifier.
In this case, the optimization problem mentioned above has no solution with M > 0.
We can extend the concept of a separating hyperplane in order to develop a hyperplane that almost separates the classes, using a so-called soft margin.
The generalization of the maximal margin classifier to the non-separable case in known as the **support vector classifier**.

Overview of the Support Vector Classifier

Even if a separating hyperplane does exist, there are instances in which a classifier based on a (perfectly) separating hyperplane might not be desirable. Such a classifier will perfectly classify all of the training observations; this can lead to sensitivity to individual observations.

A classifier based on a hyperplane that does not perfectly separate the two classes has

- Greater robustness to individual observations
- Better classification of most of the training observations

That is, it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.
The support vector classifier, sometimes called a soft margin classifier, does exactly this.
The margin is soft because it can be violated by some of the training observations.

An observation can not only be on the wrong side of the margin (inside the margin) but also on the wrong side of the hyperplane (misclassified). In fact, when there is no separating hyperplane, such a situation is inevitable.

Details of the Support Vector Classifier

The support vector classifier works in the same way as the maximal margin classifier. The new feature is that it is allowed to misclassify a few training observations.
It is the solution to the optimization problem:

maximize M

$\square 0, \square 1, \ldots \square p$, M

subject to $\sum_{j=1}^{p} \beta_j^2 = 1$, $y_i(\square_0 + \square_1 X_1 + \ldots + \square_p X_p) \geq M(1 - \epsilon_i)$

$\epsilon_i \geq 0$, $\sum_{i=1}^{n} \varepsilon_i \leq C$ where C is a **nonnegative tuning parameter**.

$\epsilon_i, \ldots, \epsilon_n$ are **slack variables** that allow individual observations to be on the wrong side of the margin or the hyperplane.

The slack variable ϵ_i tells us where the ith observation is located, relative to the hyperplane and relative to the margin:
$\epsilon_i = 0 \rightarrow x_i$ on the correct side of the margin
$\epsilon_i > 0 \rightarrow x_i$ on the wrong side of the margin → **violated the margin**
$\epsilon_i > 1 \rightarrow x_i$ on the wrong side of the **hyperplane → misclassified**

C bounds the sum of the ϵ_i's, and so it determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate. We can think of C as a budget for the amount that the margin can be violated by the n observations.
if C = 0 → no budget → no violations → maximal margin classifier
if C > 0 → no more than C observations can be on the wrong side of the hyperplane

In practice, C is treated as a tuning parameter that is generally chosen via cross-validation. As with the tuning parameters that we have seen throughout this book, C controls the bias-variance trade-off of the statistical learning technique.
C small → narrow margin, rarely violated → high fit, low bias, high variance
C larger → wider margin, allow for more violations → less hard fit, more bias, low variance

The optimization problem above has a very interesting property:
it turns out that only observation either on the margin or that violate the margin will affect the hyperplane, and hence the classifier.
Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as support vectors. These observations do affect the support vector classifier.

The fact that the support vector classifier's <u>decision rule is based only a potentially small subset of the training observations</u> (the support vectors) means that it is <u>quite robust to the behavior of observations that are far away from the hyperplane</u>.

Classification with Non-linear Decision Boundaries

In the case of the support vector classifier, we could address the problem of possibly non-linear boundaries between classes in a similar way we used for logistic regression, by enlarging the feature space using quadratic, cubic, and even higher-order polynomial functions of the predictors. For instance, rather than fitting a support vector classifier using p features $X_1, \ldots, X_p$, we could instead fit a support vector classifier using 2p features $X_1, X_1^2, \ldots, X_p, X_p^2$.

Why does this lead to a non-linear decision boundary? In the enlarged feature space, the decision boundary is in fact linear. But in the original feature space, the decision boundary is of the for m q(x) = 0, where q is a quadratic polynomial, and its solutions are generally non-linear.
It is not hard to see that there are many possible ways to enlarge the feature space, and that unless we are careful, we could end up with a huge number of features. Then computations would become unmanageable. The support vector machine, which we present next, allows us to enlarge the feature space used by the support vector classifier in a way that leads to efficient computations.

The Support Vector Machine

The support vector machine (SVM) is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using **kernels**. The main idea is to enlarge the feature space in order to accommodate a non-linear boundary between the classes. The kernel approach is simply an efficient computational trick for enacting this idea.

The solution to the support vector classifier problem involves only the inner products of the observations (as opposed to the observations themselves).

The inner product of two r-vevors a and b is defined as $<a,b> \; = \; \sum_{i=1}^{r} a_i b_i$. Thus the inner product of two observations x_i, $x_{i'}$ is given by $<x_i, x_{i'}> \; = \; \sum_{j=1}^{p} x_{ij}, x_{i'j}$.
It can be shown that

- The linear support vector classifier can be represented as
  $f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i <x, x_i>$
  where there are n parameters α_i, i = 1, …, n, one per training observation.
- To estimate the parameters α_1, …, α_n and □$_0$, all we need are the $\binom{n}{2}$ inner products $<x_i, x_{i'}>$ between all pairs of training observations,

Notice that in order to evaluate the function f(x), we need to compute the inner product between the new point x and each of the training points x_i.
However, it turns out that α_i is nonzero only for the support vectors in the solution -- that is, if a training observation is not a support vector, then α_i equals zero (constraint deactivated).
So if *S* is the collection of indices of these support points, we can rewrite any solution function as $f(x) = \beta_0 + \sum_{i \in S} \alpha_i <x, x_i>$ which typically involves far fewer terms.
To summarize, in representing the linear classifier f(x), and in computing its coefficients, **all we need are inner products**.

Now suppose that every time the inner product $<x_i, x_{i'}>$ appears in the representation or in a calculation of the solution for the support vector classifier, we replace it with a **generalization** of the inner product of the form

$K(x_i, x_{i'})$
Where K is some function that we will refer to as a kernel. A kernel is a function that quantifies the similarity of two observations.

For instance, we could simply take $K(x_i, x_{i'})$ as the inner product of the two observations, which would just give us back the support vector classifier. The inner product is known as a **linear kernel** because the support vector classifier is linear in the features; the linear kernel essentially quantifies the similarity of a pair of observations using **Pearson** (standard) **correlation**.

An alternative option would be $K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij}, x_{i'j})^d$. This is known as a **polynomial kernel** of degree d, where d is a positive integer. Using such a kernel with d > 1, instead of the standard linear kernel leads to a much more flexible decision boundary.

When the support vector classifier is combined with a non-linear kernel, the resulting classifier is known as a **support vector machine**.

Another popular non-linear kernel is the **radial kernel**, which takes the form

$K(x_i, x_{i'}) = exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$ where γ is a positive constant.
How does the radial kernel work?
If a given test observation x^* is far from a training observation x_i in terms of euclidean distance, then $\sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$ will be large, and so $K(x_i, x_{i'})$ will be very tiny. This means that x_i will play virtually no role in $f(x^*)$.
This means that the radial kernel has very local behavior, in the sense that only nearby observations have an effect on the class label of a test observation.

What is the advantage of using a kernel rather than simply enlarging the feature space using functions of the original features?

Computational : with kernels, one need to only compute $K(x_i, x_{i'})$ for all $\binom{n}{2}$ distinct pairs i, i'. This can be done without explicitly working in the enlarged feature space.
This is important because in many applications the enlarged feature space is so large that computations are intractable.
For some kernels, such as the radial kernel, the feature space is implicit and infinite-dimensional, so we could never do the computations there anyway.

SMVs with More than Two Classes

The one-versus-one or all-pairs approach constructs $\binom{K}{2}$ SVMs, each of which compares a pair of classes. We classify a test observation using each of the $\binom{K}{2}$ classifiers, and we tally the number of times that the test observation is assigned to each of the K classes. The final classification is performed by assigning the test observation to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.

The one-versus-all approach fits K SVMs, each time comparing one of the K classes to the remaining K - 1 classes.
Let $\boldsymbol{\beta}_{0k}, \boldsymbol{\beta}_{1k}, \ldots, \boldsymbol{\beta}_{pk}$ the parameters that result from fitting an SVM comparing the kth class (coded as +1) to the others (coded as -1).
Let x^* denote a test observation.
We assign the observation to the class for which $\boldsymbol{\beta}_{0k} + \boldsymbol{\beta}_{1k}x_1^* + \ldots + \boldsymbol{\beta}_{pk}x_p^*$ is largest, as this amounts to a **high level of confidence** that the test obs belongs to the kth class rather than to any of the other classes.

Relationship to Logistic Regression

It turns out that one can rewrite the criterion for fitting the support vector classifier $f(X) = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1X_1 + \ldots + \boldsymbol{\beta}_pX_p$ as $\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{minimize}}\{ \sum_{i=1}^{n} max[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2 \}$

where λ is a nonnegative tuning parameter. When λ is large then the coefficients are small, more violations are tolerated and a low-variance but high-bias classifier will result.
Thus λ is analogous to the C tuning parameter of the old formulation.
$\lambda \sum_{j=1}^{p} \beta_j^2$ is the ridge penalty term, and plays a similar role in controlling the bias-variance trade-off for the support vector classifier.

Now the support vector classifier takes the Loss + Penalty form that we have seen repeatedly

$$\underset{\beta_0,\beta_1,\ldots,\beta_p}{\text{minimize}} \{ L(\mathbf{X},\mathbf{y},\boldsymbol{\beta}) + \lambda P(\boldsymbol{\beta})$$

The loss function $\sum_{i=1}^{n} max[0, 1 - y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip})]$ takes the name of **hinge loss**. It turns out that the hinge loss function is closely related to the loss function used in logistic regression.
An interesting characteristic of the support vector classifier is that only support vectors play a role in the classifier obtained; observations on the correct side of the margin do not affect it.
This is due to the fact that the loss function is exactly zero for obs for which $y_i(\square_0 + \square_1x_1 + \ldots + \square_px_p) \geq 1$.

In contrast the loss function for logistic regression is not exactly zero anywhere, but it is very small for observations that are far from the decision boundary. Due to the similarities between their loss functions, logistic regression and the support vector classifier often give very similar results. When the classes are well separated, SVMs tend to behave better than logistic regression; in more overlapping regimes, logistic regression is often preferred.

The choice of tuning parameter is very important and determines the extent to which the model underfits or overfits the data.

There is an extension of the SVM for regression called support vector regression. Support vector regression seeks coefficients that minimize a loss where only residuals larger in absolute value than some positive constant contribute to the loss function.