Support Vector Machines

Arnab Maity (modified by Justin Post)

The support vector machine (SVM) is a family of classification rules that contain both parametric (e.g., linear) and nonparametric (e.g., kernel based) methods. It is often considered as one of the ready-to-use classifiers. It can be viewed as a generalization of linear decision boundaries for classification. The method can also be applied to regression problems.

SVM produces nonlinear boundaries by constructing a linear boundary in a large, transformed version of the feature space. We will mainly discuss three approaches:

* maximal margin classifiers and the support vector classifiers
* support vector machines

People often loosely refer to any of these methods collectively as *support vector machines*.

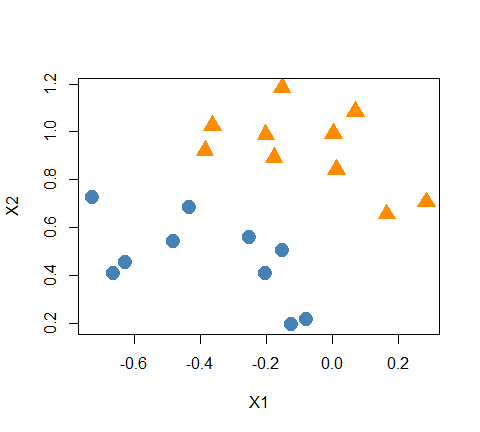
# Maximal Margin Classifiers & Support Vector Classifiers

The maximal margin classifier is a special case of the support vector classifier. It provides a useful starting point to understand SVMs generally.

## Maximal Margin Classifier

We know that LDA and logistic regression both estimate linear decision boundaries (using different approaches) in classification problems. SVMs are built to try to separate data into classes in an optimal way.

Consider a simulated dataset that contains two predictors, and , and observations come from one of two classes (blue circles and orange triangles).



Scatterplot between two simulated variables x1 and x2 where the data points are colored by our response, the class type. The classes are completely separable by a line between x1 and x2.

The two classes are well separated, and a straight line can be used for classification. This situation is called *linearly separable*. In the two predictor case that means:

There exists such that the line

perfectly separate the two classes.

Here, this line is not unique!

For SVM, we actually define our binary response by coding it as a -1 or 1 variable. With separable classes, this implies:

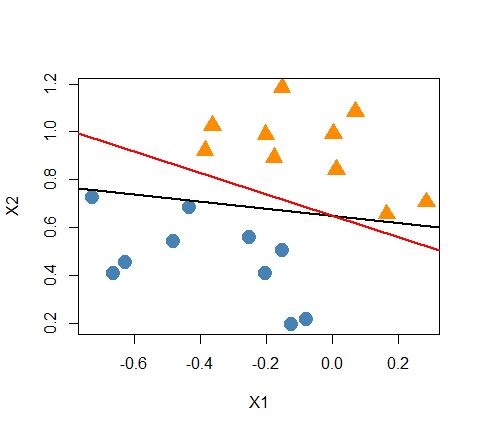
for a data point with predictor values , we have the relation

We can therefore define our by

* is called a **hyperplane** thus the name **separating hyperplane**.
* Note that
* Once we fit our terms, we can classify a new observation via

### Fitting the Hyperplane

When the problem is **linearly separable**, there are infinitely many such separating hyperplanes. Two are shown in the plot of our simulated data below.



Simulated two class data with two possible separating hyperplanes (lines) that perfectly separate the data.

How to determine the ‘optimal’ line in this case?

* Given any separating hyperplane, we can define the *margin* as the minimal distance from each observation to the hyperplane.
* Define distance between the hyperplane and -th training point.
* The margin is defined as .

The optimal classification rule is the line that *maximizes the margin* around the separating line. Such a classifier is called **the maximal margin classifier**.

Formally, we consider the optimization problem:

The two conditions ensure that, when , each observation will remain on the correct side of the boundary and is at least distance from the boundary.

It can be shown that the optimization problem above is equivalent to the following optimization problem:

To understand the geometry behind the optimization problem above, we note that for perfectly separable data,

* for each .
* The constraint implies if , and that if .

The constraints therefore form an **empty slab** or **margin** around the linear decision boundary. The thickness of the margin is .

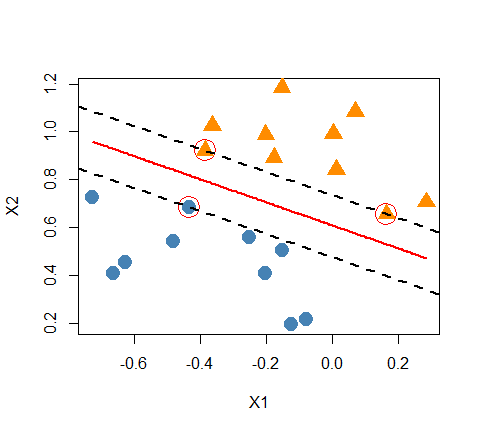
* Hence we are choosing and to maximize the thickness
* Equivalently this is just minimizing .

The boundaries of this empty slab/margin are determined by the points that exactly satisfies the condition

Define the *optimal* separating hyperplane as . Thus our classification rule is as follows:

For a new observation ,

In figure below, the optimal separating line is shown as the solid red line, the closest points to the line are circled, and the separation between the classes is shown using the dashed black lines.



The scatterplot of our linearly separable simulated data is shown with the optimal line between the observations. Dashed black lines running parallel to the optimal line show the thickness of the margin. Three points fall directly on these dashed black lines. One point from the first class touches one dashed line and two points from the other class touch the other dashed line.

Essentially, the separating line corresponding to the maximal margin classifier represents the middle line of the widest space that we can fit between the two classes.

Although none of the training observations fall in the margin (by construction), this will not necessarily be the case for test observations. The intuition is that a large margin on the training data will lead to good separation on the test data.

### Support Vectors

In the plots, we noticed that there were three points that are closest, and equidistant, to the optimal separating line.

These points exactly satisfy the condition

These points are called the **support vectors** for this problem.

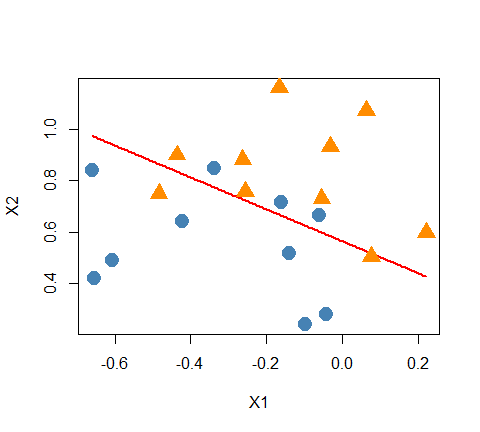
* It can be shown that *only the support vectors are enough to define the optimal classification rule fully*.
* If we move the support vectors, the optimal separating line also changes.
* A movement to any of the other observations would not affect the separating hyperplane, provided that the observation’s movement does not cause it to cross into this margin area.
* This property of the maximal margin classifier is important in development of support vector classifier and support vector machine!

### Predictors

Everything above generalizes to the case where we have predictors!

## Support vector classifiers

Of course, our data isn’t usually going to be linearly separable! Consider a more realistic example.

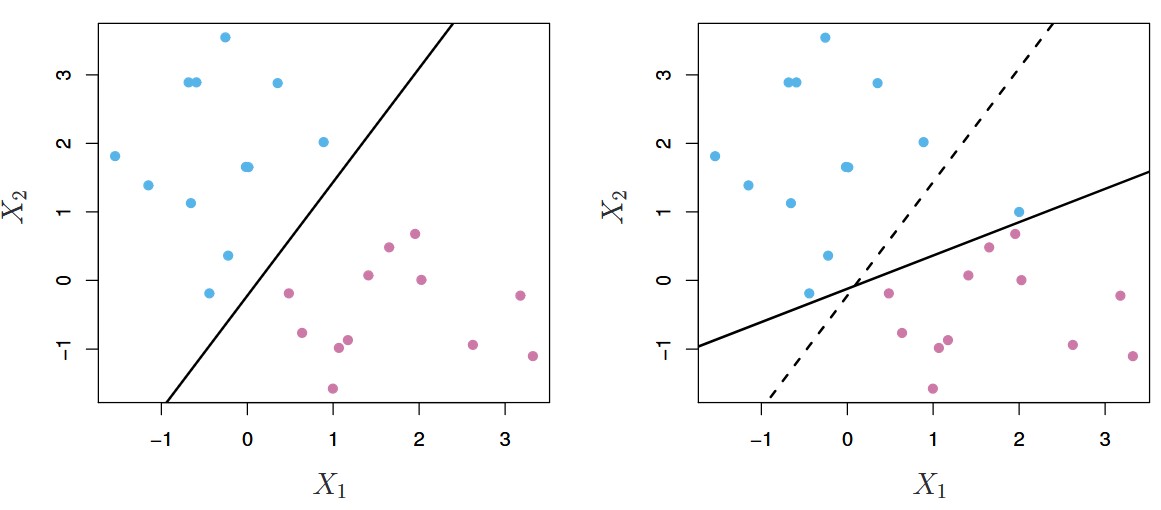


Scatterplot of simulated data that are not completely separable. The optimal separating hyperplane (line) is shown with some observations from the first class on both sides of the line. Similarly, some observations from the other class are on both sides of the line.

Here we will not be able to find a line that entirely separates the groups. That is, the maximal margin classifier can not be computed.

However, we can generalize the ideas to develop a classification rule that *almost* separates the classes. To do so,

* we allow a few points to fall on the wrong side of the margin or separating hyperplane
* the classifier is called a *support vector classifier* or a *soft-margin classifier*
* Even in the completely separable case, we still may consider such soft margin classifier for robustness. The graph below shows the drastic change of the maximal margin classifier from the addition of a single observation.



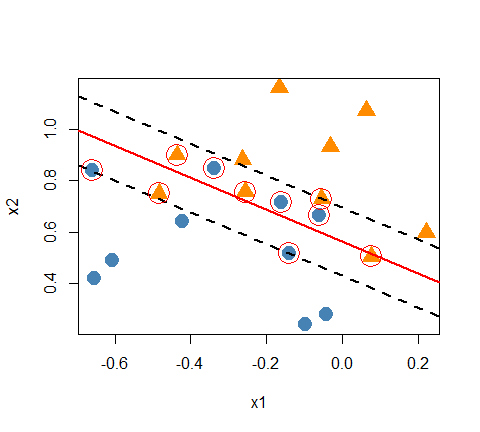
Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane (left panel). An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line (right panel). The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.

### Slack Variables

In the support vector classifier, each data point is given a **slack variable** that allows individual data points to be on the wrong side of the margin or the separating hyperplane.

quantifies where the -th observation is located relative to the hyperplane and the margin:

* If then the -th data point is on the correct side of the margin;
* If then the -th observation is on the wrong side of the margin (the data point has *violated the margin*),
* If then it is on the wrong side of the hyperplane.



A scatterplots showing a simulatd two-class data set is shown. The support vector, separating hyperplane, and margin are highlighted. Points that fall outside the Margin that are correctly classified have their slack variable at 0. Those within the margin but still on the correct side have their slack variable between 0 and 1. Those on the wrong side of the margin have their slack variable larger than 1.

The support vector classifiers then attempt to maximize the margin such that

for a pre-specified constant .

For the two predictor case, we solve the optimization problem:

where is a nonnegative tuning parameter.

* Conceptually, the value in the constraint is the proportional amount by which the quantity is on the wrong side of its margin.
* By bounding the sum , we bound the total proportional amount by which falls on the wrong side of their margin.
* Misclassifications occur when , so bounding at a value bounds the total number of training misclassifications at .
* Specifying gives us the maximal margin classifier, if it exists.
* In contrast, as increases we become more tolerant of violations to the margin, and so the margin will widen.

**In practice, we need to choose via cross-validation or other appropriate method.**

As before, we can write an equivalent optimization problem:

subject to the constraints

This is the usual way the support vector classifier is defined for the nonseparable case (and the equations generalize for predictors).

As before, once we obtain the estimators and , the estimated hyperplane is , and our classification rule is:

* for a new observation ,
* The classification boundary, and the two margins are
* respectively.

As with the maximal margin classifier, *the classifier is affected only by the observations that lie on the margin or violates the margin*.

* Data points that fall directly on the margin, or on the wrong side of the margin for their class, are known as *support vectors*.

### Support Vector Classifier Example

Let us now revisit the wines data. We can use the svm() function in the e1071 library to implement support vector classifier.

We will only use two classes to begin with (1 and 2) and two covariates, Alcohol and Proline, so that we can plot the results. Support vector classifiers however can be implemented for much larger number of predictors.

* While it is technically not needed, we often standardive the predictors beforehand – svm() does standardization by default.
  + We perform standardization manually so that we can compare the coefficients of the resulting hyperplane.

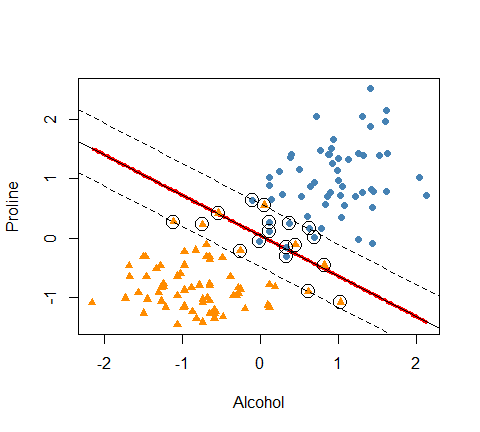
# Read wine data  
wines <- read\_table("https://www4.stat.ncsu.edu/online/datasets/Wines.txt")

-- Column specification --------------------------------------------------------  
cols(  
 Class = col\_double(),  
 Alcohol = col\_double(),  
 Malic = col\_double(),  
 Ash = col\_double(),  
 Alcal = col\_double(),  
 Mg = col\_double(),  
 Phenol = col\_double(),  
 Flav = col\_double(),  
 Nonf = col\_double(),  
 Proan = col\_double(),  
 Color = col\_double(),  
 Hue = col\_double(),  
 Abs = col\_double(),  
 Proline = col\_double()  
)

# Subset to two classes (1 and 2)  
wine\_twoclass <- wines[wines$Class < 3, ]  
#make the response a factor  
wine\_twoclass$Class <- as.factor(wine\_twoclass$Class)  
  
# Standardize the predictors (excluding Class)   
wine\_twoclass[,-1] <- scale(wine\_twoclass[,-1],   
 center = TRUE,   
 scale = TRUE)  
# SVC with linear boundary  
sv.wine <- svm(Class ~ Proline + Alcohol,   
 data = wine\_twoclass,   
 type = "C-classification",   
 kernel = "linear",   
 cost = 1)

* The argument type = "C-classification" specifies that we want to perform the support vector classification.
* The argument kernel = "linear" ensures that we are using support vector classifier.
  + Later, when we learn *support vector machine*, we will specify different nonlinear kernels.
* The parameter cost takes the role of , however, the svm() function uses a different mathematical formulation than what we discussed above.
  + Thus cost is not exactly same as . The main concepts, however, remain the same. When the cost argument is small, then the margins will be wide and many support vectors will be on the margin or will violate the margin.
  + When the cost argument is large, then the margins will be narrow and there will be few support vectors on the margin or violating the margin.
  + The case cost corresponds to the perfectly separable case.

Now let’s plot the support vector classifier.



A scatter plots between Alcohol (x) and Proline (y) is shown. The points separate reasonably well with most of one class in the bottom left of the graph and most of the other class in the top right. There is some overlap of points in the middle region. The Support vector classifier is shown. This line has a negative slope and splits the points well. The margin around the line is shown and all points within the margin are highlighted as they are the support vectors. Three of of the first class are misclassified and two of the second class.

The coefficients of the separating hyperplane can be extracted using the coef() function.

beta\_hat <- coef(sv.wine)  
beta\_hat

(Intercept) Proline Alcohol   
-0.09306929 1.87645694 1.28020122

The hyperplane is

The lines indicating the two margins are

and

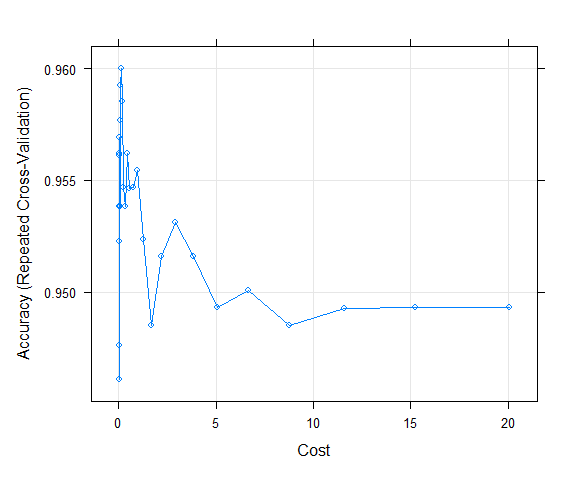
respectively.

The points that are on the margin or violate the margin are support vectors!

cost of course is a tuning parameter. Let’s use the caret package to employ cross-validation to choose cost.

# Set up repeated cv option  
set.seed(1001)  
tr <- trainControl(method = "repeatedcv",   
 number = 5, repeats = 10)  
# Tuning grid  
tune\_grid <- expand.grid(cost = exp(seq(-5,3,len=30)))  
# Train the model  
sv\_caret <- train(Class ~ Proline + Alcohol,  
 data = wine\_twoclass,  
 method = "svmLinear2",  
 tuneGrid = tune\_grid,  
 trControl = tr)

We can plot the accuracy as a function of the cost parameter.



Results from repeated CV using support vector classifier on two-class wines data. A line chart showing cost (x) and accuracy (y) is shown. The accuracy starts off lower and then jumps to a higher value quickly. The accuracy then drops off for larger values of cost.

We can get our optimal cost value using the bestTune argument.

# Best C  
sv\_caret$bestTune

cost  
12 0.1400834

Now let’s refit using the svm() function with this optimal value.

# Final fit  
wine\_sv\_final <- svm(Class ~ Proline + Alcohol,   
 data = wine\_twoclass,   
 type = "C-classification",   
 kernel = "linear",   
 cost = sv\_caret$bestTune$cost)

To obtain the predicted classes, we can use the predict() function. Let’s check out our confusion matrix on the training data.

pred.class <- predict(object = wine\_sv\_final,   
 newdata = wine\_twoclass,   
 type = "response")  
err <- klaR::errormatrix(true = wine\_twoclass$Class,   
 predicted = pred.class)  
#rows are true values and columns predicted  
round(err, 3) |>  
 kable()

|  | 1 | 2 | -SUM- |
| --- | --- | --- | --- |
| 1 | 57 | 2 | 2 |
| 2 | 3 | 68 | 3 |
| -SUM- | 3 | 2 | 5 |

## More than two classes

What do we do when our response isn’t binary? We have two options:

* one-versus-one approach
* one-versus-all approach

### one-versus-one

Here we compute all pair-wise classifiers.

* That is, compute all classification rules.
* Given a test observation, we classify it using *each* of the classifiers, and record the number of times that the test observation is assigned to each of the classes.
* The final classification is performed taking a *majority vote*.

### one-versus-all

Alternatively, we can compare class with the remaining classes pooled together.

* That is, class vs. not class .
* Let denote the parameters that result from fitting an SVM comparing the -th class (coded as ) to the others (coded as ).
* Given a test observation, , we assign the observation to the class for which is largest, as this amounts to a high level of confidence that the test observation belongs to the th class rather than to any of the other classes.

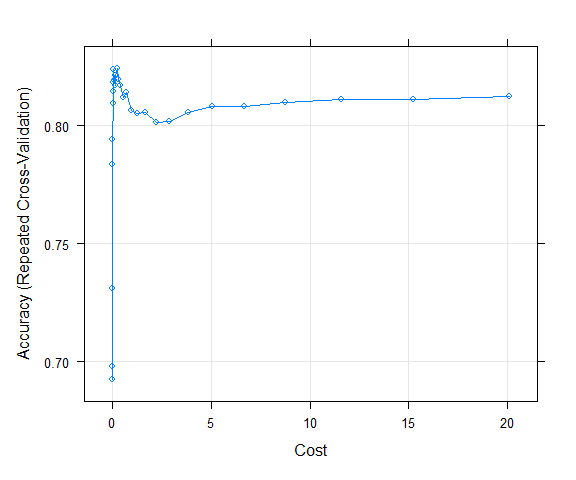
### Example

The svm() function uses the *one-versus-one* approach.

Let us use the full wines data with three classes, but with two predictors Alcohol and Proline for plotting purposes.

# Pre-process wines data  
wines$Class <- as.factor(wines$Class)  
wines[, -1] <- scale(wines[,-1],   
 center = TRUE, scale = TRUE)  
# Train the model  
set.seed(1001)  
tr <- trainControl(method = "repeatedcv",   
 number = 5, repeats = 10)  
tune\_grid <- expand.grid(cost = exp(seq(-5,3,len=30)))  
sv\_caret <- train(as.factor(Class) ~ Proline + Alcohol,  
 data = wines,  
 method = "svmLinear2",  
 tuneGrid = tune\_grid,  
 trControl = tr)

Now we can plot the accuracy as a function of cost.

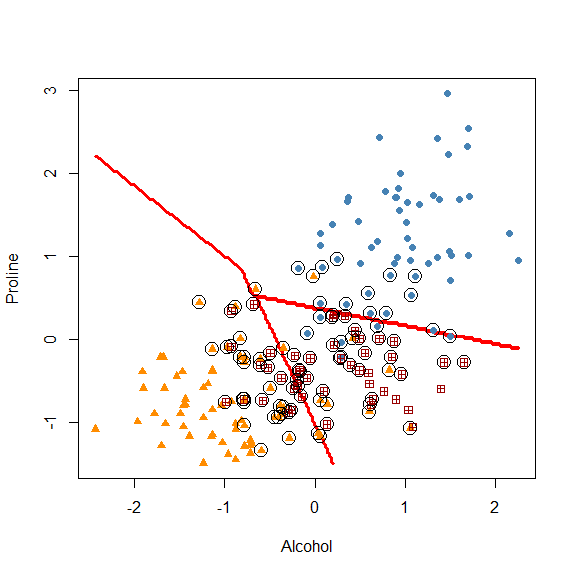


CV results for three class wines data. A line plot is shown with cost (x) and accuracy (y). The accuracy starts low and jumps up quickly. It then drops a bit and levels off.

Let’s refit with the optimal tuning value.

# SVC with optimal cost  
sv.wine <- svm(Class ~ Proline + Alcohol,   
 data = wines,  
 type = "C-classification",   
 kernel = "linear",   
 cost = sv\_caret$bestTune$cost)

Now we can plot our SVM with three classes.



A scatterplot between alcohol (x) and proline (y) is shown with points displaying the three different classes. One class mostly falls in the bottom left of the graph, another in the bottom right, and the other in the top right. The SVM is plotted and shows lines that roughly separate these three classes. Many points close to the decision boundary are highlighted and represent the support vectors.

# Support vector machines

The support vector classifier described so far finds linear boundaries in the input feature space. Often linear effects of the covariates are not enough for a classification problem.

Thus we might want to incorporate nonlinear terms (e.g., square or cubic terms).

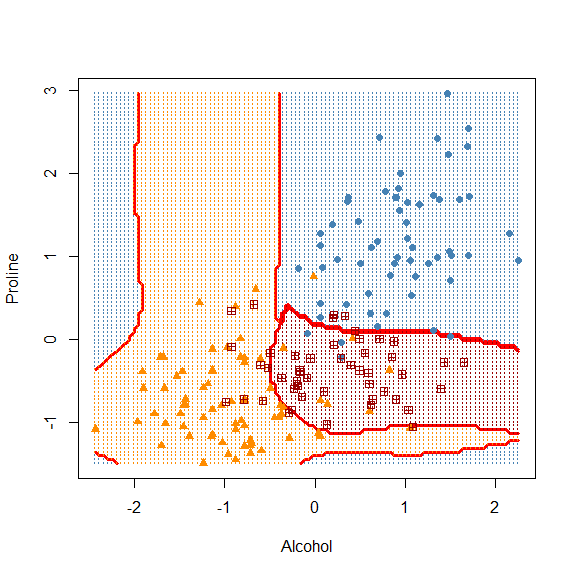
* With two covariates and , we might include and in our classifier.
* If we run the support vector classifier, the decision boundary would be a quadratic polynomial.
* In general, we can incorporate other nonlinear transformations as features!
  + This makes the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines.
  + Linear boundaries in the enlarged space tend to achieve better training-class separation, and translate to nonlinear boundaries in the original space.
  + Once the basis functions are selected, the procedure is the same as before.
  + We fit the support vector classifier using input features , , and produce the (nonlinear) function
  + The classifier is as before.

## Example SVM

The following code includes natural cubic spline terms with four degrees of freedom for Alcohol and Proline with cost .

library(splines)  
sv.wine <- svm(Class ~ ns(Proline, df=4) + ns(Alcohol, df=4),   
 data = wines,  
 type = "C-classification",   
 kernel = "linear",   
 cost = 0.2)

The decision boundaries are quadratic, as shown below.

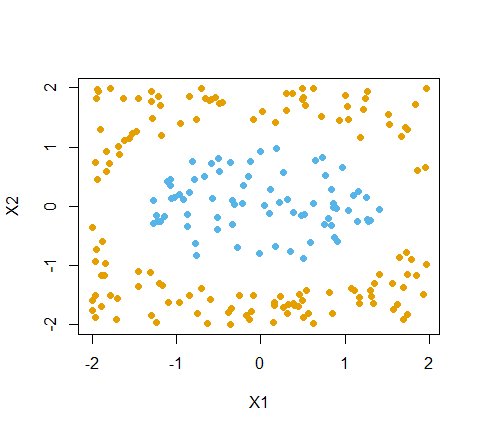


Classification of wine data using support vector classifier with natural cubic splines. The decision boundaries here are non-linear.

The core idea is that even though the two classes are not separable in the original feature space, **they may be separable in transformed, and/or expanded feature space**.

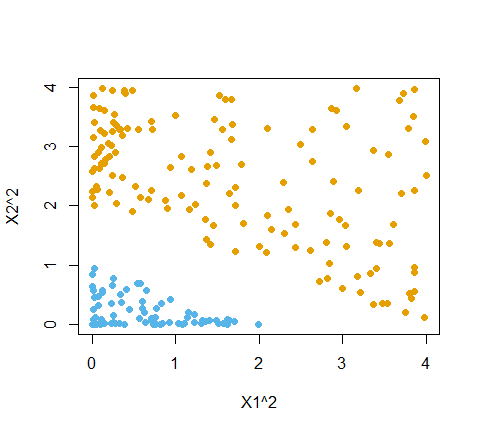
## Simulated Example

Consider the simulated data in the figure below.



Simulated example with a non-linear relationship. A scatterplot is shown between two variables. All points in the middle of the plots are from one class. Points along the outer parts of the graph come from the other class. These classes are completely separable using a circle.

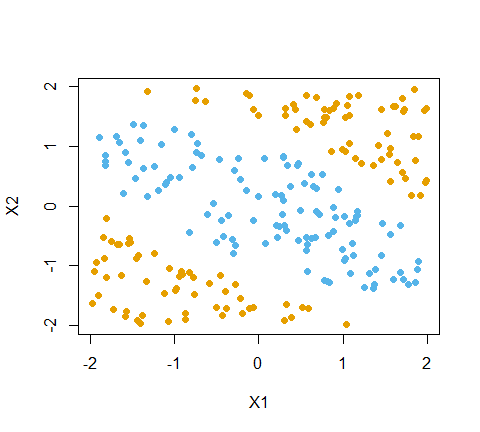
If we consider this data on a transformed scale, we can see that they are linearly separable in that space.



Simulated example of transformed and enlarged feature space. THe x-axis is given by X1 squared and the y-axis is given by X2 squared. The points are linearly separable in this space.

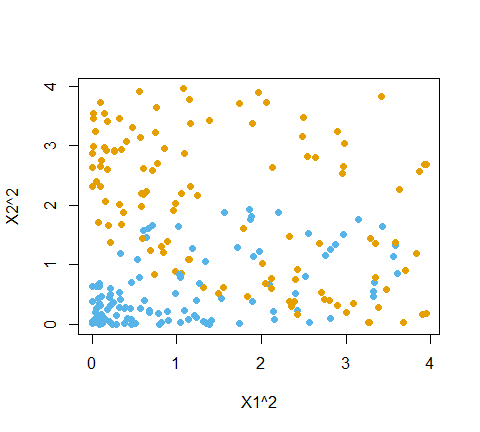
* We see that there is a linear combination of the form that separates the two classes.
* By including basis functions that allow for this relationship, we can achieve highly accurate predictions.

Consider another simulated example given below.



A scatterplot of simulated data between x1 and x2 is shown. Two lines would perfectly separate the data by class with all of one class falling between the lines and the other class outside of the lines.

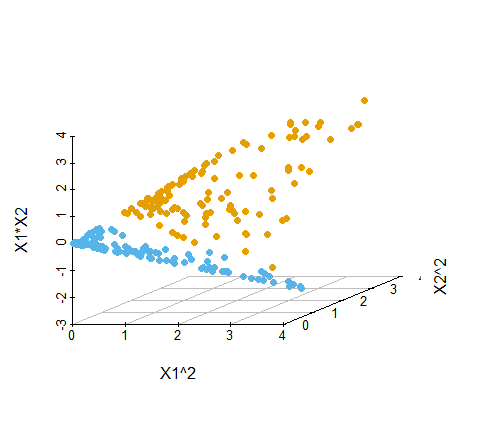
We could consider a similar transformed space of and .



A scatterplot of simulated data is shown between x1 squared and x2 squared. The data are nearly separable in this case with some overlap.

Here we almost have separability but there is still a good bit of overlap.

Instead, we could consider a third transformation, an interaction between and . Plotting the three variables, , , and requires us to use three dimensions.



A three dimensional scatterplot between x1 squared, x2 squared, and x1\*x2 is shown. Here we get complete separation of our classes in this enlarged feature space.

* We see that there is a linear combination of the form that separates the two classes.
* In general, we can include other nonlinear transformations as features in support vector classifiers (we then call them support vector machines)!

## Kernel Functions

We’ve seen that SVMs generalize support vector classifiers by including nonlinear features.

* The dimension of the enlarged space is allowed to get very large, infinite in some cases.

How? Without going into mathematics, SVM does so using the so called *kernel trick*, that is, by specifying a *kernel function* that controls which nonlinear features to include in the classifier.

* Define . It turns out that
* for some weights .
* Thus the solution to the support vector classifier problem can be represented as
* To estimate and , it can be shown that we only need the all *pair-wise* inner products of the training data .
* Many of the resulting solutions are zero. The observations for which are nonzero are called the *support vectors*.

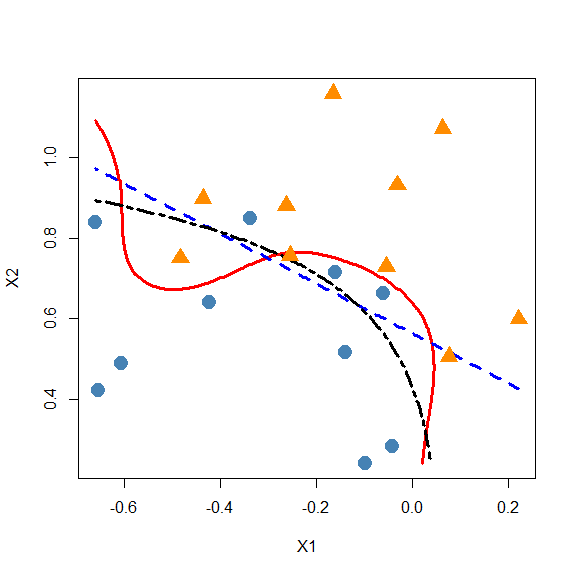
To summarize, in representing the linear classifier , and in computing its coefficients, all we need are inner products: , and .

For general nonlinear features

the classifier can be computed using the inner products: and .

* In fact, we need not specify the transformation at all, but require only knowledge of the *kernel function*
* that computes inner products in the transformed space.
* In general, the kernel function should be a symmetric positive (semi-) definite function.
  + Some popular choices for in the SVM are
* Using the “linear” or “quadratic” ( degree polynomial) kernel will result in linear or quadratic classification boundaries, respectively.
* Using a “radial basis kernel” captures other nonlinear features.

As an example, the plot below shows three classification rules corresponding to linear, quadratic and radial kernels based on a simulated data set.



A scatterplot of simulated two-class data is shown. A linear boundary (blue dashed), a quadratic boundary (black dash-dotted), and a radial-basis based boundary (red solid) classification rules are shown. We see the linear boundary is the simplest, the quadratic more flexible, and the radial much more flexible.

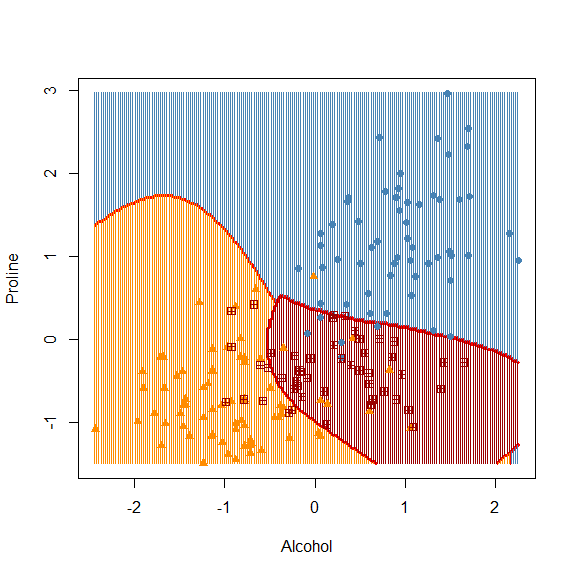
To see the correspondence between kernels and original features, consider an example a feature space with two inputs and , and a polynomial kernel of degree 2.

* Here, .
* We can define (not unique)
* Then we have . ## Another SVM Example

Let us now revisit the wines data. The figure below shows the decision boundaries using the radial basis kernel SVM.

sv.wine <- svm(Class ~ Proline + Alcohol,   
 data = wines,  
 type = "C-classification",   
 kernel = "radial",  
 gamma = 1,  
 cost = 0.2)

* Here we set .
* In practice, we need to explore a few values of to select an optimal value based on test performance.



A scatterplot between alcohol (x) and proline (y) is shown with points differentiated by wine class (3 classes). The decision boundaries for the radial basis SVM are shown. These boundaries are smooth but nonlinear in this case.

* In general, non-linear kernels are better in capturing nonlinear boundaries compared to linear methods like LDA, Logistic regression etc.

One advantage of using a kernel rather than simply enlarging the feature space using transformation of the original features is ease of computation.

* Using kernels, we only need to compute inner products distinct pairs of observations .
* This can be done without explicitly working in the enlarged feature space.
* This is important because in many applications of SVMs, the enlarged feature space is so large that computations are intractable.
* For some kernels, e.g., radial kernel, the feature space is implicit and infinite-dimensional, so it is not feasible to specify the entire feature space using explicit basis functions.

# SVM as a Penalized Method

Suppose we define , where are transformations of the original functions.

* The SVM problem can be shown to be equivalent to
* where denotes the positive part, that is,, and is a penalty parameter.
* The loss function above is also called the *hinge loss*.
* The optimization problem has the typical form of loss + penalty that we have encountered in our discussion of regularized regression.

The penalty above is the ridge penalty. Some recent work (Yi, C. and Huang, J. (2017) Semismooth Newton Coordinate Descent Algorithm for Elastic-Net Penalized Huber Loss Regression and Quantile Regression, Journal of Computational and Graphical Statistics, 547-557.) replaces the ridge penalty with the lasso and elastic net penalty to obtain sparse solution of the coefficients.

* The sparseSVM library performs such a procedure in R. For the two-class wines data, let us run the support vector classifier based on all 13 predictors.

# Set up repeated cv option  
set.seed(1001)  
tr <- trainControl(method = "repeatedcv",  
 number = 5, repeats = 10)  
# Tuning grid  
tune\_grid <- expand.grid(cost = exp(seq(-5,3,len=30)))

Let us first fit an optimized support vector classifier.

* Recall that we have already standardized the two-class wines data.

# Train the model  
sv\_caret <- train(Class ~ .,  
 data = wine\_twoclass,  
 method = "svmLinear2",  
 tuneGrid = tune\_grid,  
 trControl = tr)  
# Final model  
wine\_sv\_final <- svm(Class ~ .,  
 data = wine\_twoclass,   
 type = "C-classification",  
 kernel = "linear",  
 cost = sv\_caret$bestTune$cost)  
sv\_caret$bestTune

cost  
11 0.1063118

The coefficients of this fit are given below.

beta\_hat <- coef(wine\_sv\_final)  
beta\_hat |>  
 kable()

|  | x |
| --- | --- |
| (Intercept) | -0.1203488 |
| Alcohol | 0.5673738 |
| Malic | 0.1908668 |
| Ash | 0.4239142 |
| Alcal | -0.4932079 |
| Mg | 0.0306272 |
| Phenol | -0.0596313 |
| Flav | 0.0450428 |
| Nonf | -0.0931409 |
| Proan | -0.0633336 |
| Color | 0.2187538 |
| Hue | -0.1169929 |
| Abs | 0.3065914 |
| Proline | 0.7282565 |

* Notice that the standard support vector classifier does not set any of the small coefficients to exactly zero.
* We may suspect predictors with smaller values of coefficients (e.g., Mg, Phenol, Flav etc) may not contribute to the classification rule as much as other variables like Alcohol, Proline, Ash, Alcal, which have relatively large coefficients.
* Let us run the same classifier with a LASSO penalty.
  + We will use CV to choose . By default, the R function uses 10-fold CV.

library(sparseSVM)  
set.seed(1001)  
X <- as.matrix(wine\_twoclass[,-1])  
y <- wine\_twoclass$Class  
# Cross validation to choose lambda   
spr.cv <- cv.sparseSVM(X = X, y = y, alpha = 1)  
spr.cv$lambda.min

[1] 0.08655872

The coefficients for the two fits are given below for comparison.

# Final fit  
beta\_sparse <- coef(spr.cv,   
 lambda = spr.cv$lambda.min)  
cbind(beta\_sparse, beta\_hat) |>  
 kable()

|  | beta\_sparse | beta\_hat |
| --- | --- | --- |
| (Intercept) | -0.0481631 | -0.1203488 |
| Alcohol | 0.5979170 | 0.5673738 |
| Malic | 0.0021007 | 0.1908668 |
| Ash | 0.0745652 | 0.4239142 |
| Alcal | -0.0287757 | -0.4932079 |
| Mg | 0.0000000 | 0.0306272 |
| Phenol | 0.0000000 | -0.0596313 |
| Flav | 0.0743918 | 0.0450428 |
| Nonf | 0.0000000 | -0.0931409 |
| Proan | 0.0000000 | -0.0633336 |
| Color | 0.0000000 | 0.2187538 |
| Hue | 0.0000000 | -0.1169929 |
| Abs | 0.1625234 | 0.3065914 |
| Proline | 0.8820206 | 0.7282565 |

* Notice that the LASSO penalty is setting some coefficients to 0.

# SVM Regression

SVM can be applied to regression problems with quantitative response as well.

Let us start with a linear regression problem

where

Support vector regression solves the following problem:

where the loss function has the form

- This loss function ignores errors of size less than .

* It can be shown that the solution function has the form
* where and are constants.
* Typically only a subset of the values are nonzero, and the associated data values are called the support vectors.
* As was the case in the support vector classification, the solution depends on the input values only through the inner products .
* Thus we can generalize the methods to richer spaces by defining an appropriate inner product and corresponding kernel function.

## SVM Regression in R

In R, the svm() functions can perform regression as well.

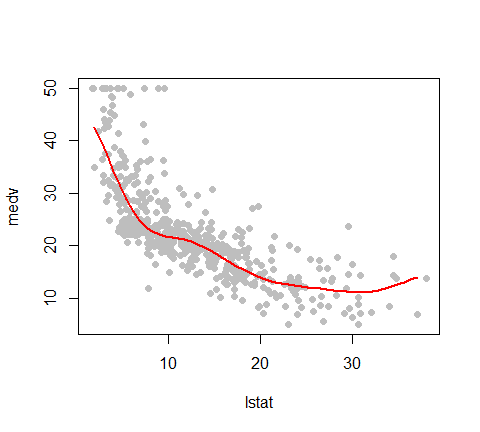
* Here we have two parameters to tune: and cost.
* This can be done using usual methods such as CV.

Here we demonstrate support vector regression for a specific value of and cost = 1, using the Boston housing data.

* We use radial kernel with .

library(ISLR2)  
svr <- svm(medv ~ lstat, data = Boston,  
 type = "eps-regression",  
 kernel = "radial", gamma = 1,  
 cost = 1, epsilon = 0.1  
 )

With only one predictor we can visualize the fit easily.



A scatterplot between lstat and medv is shown. The data generally start in the top left and decrease, leveling off for larger values of lstat. The SVM fit is overlayed and it roughly flows through the middle of the data points in a non-linear fashion.

Overall, SVMs are quite useful in practice, and extend beyond what we discussed in this chapter.

* For example, the penalized loss function formulation, as described in the classification and regression context can be used for *any* convex loss function with *any* kernel function.
* This enables us to use kernel methods in function estimation in linear, generalized linear (e.g., logistic), and other (e.g., least absolute deviation) regression models.