Tutorial 8: Support Vector Machines

In this tutorial, we use **tidyverse** to perform data manipulation, the **kernlab** and **e1071** packages to perform calculations and produce visualisations for SVMs and the **ISLR** package to load data and show SVMs in action. **RColorBrewer** helps with visualising output generated by **kernlab** when there are more than 2 classes of data.

The data sets used, with the exception of `Khan`, will be generated using built-in R commands. The Support Vector Machine methodology is sound for any number of dimensions, but becomes difficult to visualise for more than 2. As previously mentioned, SVMs are robust for any number of classes, but we will stick to 2 and 3 dimension for this tutorial.

Maximal Margin Classifier If the classes are separable by a linear boundary, we can use a Maximal Margin Classifier to find the classification boundary. To visualise an example of separated data, we generate 40 random observations and assign them to two classes.

The goal of the maximal margin classifier is to identify the linear boundary that maximises the total distance between the line and the closest point in each class. We can use the `svm()` function from **e1071** to find this boundary.

Follow the code in the accompanying script

In the plot, points that are represented by an X are the support vectors, or the points that directly affect the classification line. The points marked with an o are the other points, which don't affect the calculation of the line. This principle will lay the foundation for support vector machines. The same plot can be generated using the **kernlab** package.

Follow the R commands in the accompanying script

kernlab shows a little more detail than **e1071**, showing a colour gradient that indicates how confidently a new point would be classified based on its features. Just as in the first plot, the support vectors are marked, in this case as filled-in points, while the classes are denoted by different shapes.

Support Vector Classifiers As convenient as the maximal marginal classifier is to understand, most real data sets will not be fully separable by a linear boundary. To handle such data, we use the modification described in the lecture. Firstly, simulate a new data set where the classes are more mixed (follow the script).

Whether the data is separable or not, the `svm()` command syntax is the same. When the data that is not linearly separable, the `cost` argument, described in the lecture, is now of crucial importance. This quantifies the penalty associated with having an observation on the wrong side of the classification boundary. We can plot the fit in the same way as the completely separable case. Firstly, see how **e1071**

deals with it (follow the script).

By increasing the cost C of misclassification from 10 to 100, we can see the difference in the classification line. Now repeat the process of plotting the SVM using the kernlab package (follow script).

Cost of Misclassification How do we decide how costly these misclassifications are? Instead of specifying a cost up front, we can use the `tune()` function from **e1071** to test various costs and identify which value produces the best fitting model. This is described in the lecture.

For the data set, the optimal cost (from the choices provided) is calculated to be 0.1, which does not penalise the model much for misclassified observations. Once this model has been identified, we can construct a table of predicted classes against true classes using the `predict()` command (shown in the script).

Using this support vector classifier, 80% of the observations were correctly classified, which is in line with the plot. To test the classifier more rigorously, we could split the data into training and testing sets and then see how the SVC performed with the observations not used to construct the model.

Support Vector Machines

Support Vector Classifiers are a subset of the group of classification structures known as Support Vector Machines. Support Vector Machines can construct classification boundaries that are nonlinear in shape. The options for classification structures using the `svm()` command from the **e1071** package are linear, polynomial, radial, and sigmoid. To demonstrate a nonlinear classification boundary, construct a new data set.

Follow the script.

Notice that the data is not linearly separable, and furthermore, is not all clustered together in a single group. There are two sections of class 1 observations with a cluster of class 2 observations in between. To demonstrate the power of SVMs, we'll take 100 random observations from the set and use them to construct our boundary. We set `kernel = ''radial''` based on the shape of our data and plot the results. (Follow script)

The same procedure can be run using the **kernlab** package, which has more kernel options than the corresponding function in **e1071**. In addition to the four choices in **e1071**, this package allows use of a hyperbolic tangent, Laplacian, Bessel, Spline, String, or ANOVA RBF kernel. To fit this data, we set the cost to be the same as it was before, $C = 1$.(follow script)

Visually (from the plot), it looks as if the SVM does a reasonable job of separating the two classes. To fit the model, we used cost $C = 1$. It is not completely clear a priori which cost will produce

the optimal classification boundary, so we use the `tune()` command, which implements the techniques described in the lecture, to try several different values of cost as well as several different values of the scaling parameter γ described in the lecture, for fitting nonlinear boundaries. (Follow script).

The model that reduces the error the most in the training data uses a cost $C = 1$ and $\gamma = 0.5$. We can now see how well the SVM performs by predicting the class of the 100 testing observations (follow script)

The best-fitting model produces 65% accuracy in identifying classes. For such a complicated shape of observations, this performed reasonably well.

SVMs for Multiple Classes

We did not deal with multiple classes in the lecture, but the idea extends quite easily and the implementation is the same. We construct our data set the same way as we have previously, only now specifying three classes instead of two (follow script)

The commands remain the same for the **e1071** package; we specify a cost and tuning parameter γ and fit a support vector machine. The results and interpretation are similar to two-class classification (follow script).

Now check to see how well the model fits the data using the `predict()` command (follow script).

As shown in the resulting table, 89% of the training observations were correctly classified. Since we did not split the data into training and testing sets, we have not validated the results properly.

The **kernlab** package can fit more than 2 classes, but it cannot plot the results. To visualise the results of the `ksvm` function, follow the R script to create a grid of points, predict the value of each point, and plot the results.

Application

The `Khan` data set contains data on 83 tissue samples with 2308 gene expression measurements on each sample. These were split into 63 training observations and 20 testing observations, and there are four distinct classes in the set. It would be impossible to visualise such data, so we choose the simplest classifier (linear) to construct our model. We will use the `svm` command from **e1071** for the data analysis. (Follow script).

First, check how well the model classified the training observations. This is high, but again, does nott validate the model. If the model does not do a good job of classifying the training set, it could be a

red flag. In our case, all 63 training observations were correctly classified.

For validation, check how the model performs on the testing set (follow script).

The model correctly identifies 18 of the 20 testing observations. SVMs and the boundaries they impose are more difficult to interpret at higher dimensions, but these results seem to suggest that our model is a good classifier for the gene data.