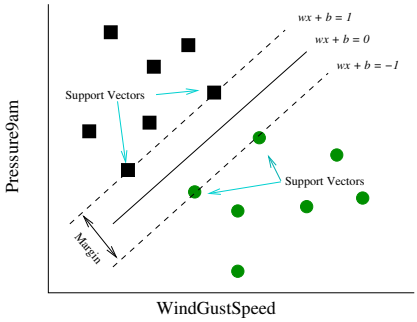
Chapter 14

Support Vector Machines



A support vector machine (SVM) searches for so-called *support vectors* which are observations that are found to lie at the edge of an area in space which presents a boundary between one of these classes of observations (e.g., the squares) and another class of observations (e.g., the circles). In the terminology of SVM we talk about the space between these two regions as the *margin* between the classes. Each region contains observations with the same value for the target variable (i.e., the class). The support vectors, and only the support vectors, are used to identify a hyperplane (a straight line in two dimensions) that separates the classes. The maximum margin between the separable classes is sought. This then represents the model.

It is usually quite rare that we can separate the data with a straight line (or a hyperplane when we have more than two input variables). That is, the data is not usually distributed in such a way that it is linearly separable. When this is the case, a technique is used to combine (or remap) the data in different ways, creating new variables so that the classes are then more likely to become linearly separable by a hyperplane (i.e., so that with the new dimensional data there is a gap between observations in the two classes). We can use the model we have built to score new observations by mapping the data in the same way as when the model was

G. Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*, Use R, DOI 10.1007/978-1-4419-9890-3_14, © Springer Science+Business Media, LLC 2011

built, and then decide on which side of the hyperplane the observation lies and hence the decision associated with it.

Support vector machines have been found to perform well on problems that are nonlinear, sparse, and high-dimensional. A disadvantage is that the algorithm is sensitive to the choice of tuning option (e.g., the type of transformations to perform), making it harder to use and time-consuming to identify the best model. Another disadvantage is that the transformations performed can be computationally expensive and are performed both whilst building the model and when scoring new data.

An advantage of the method is that the modelling only deals with these support vectors rather than the whole training dataset, and so the size of the training set is not usually an issue. Also, as a consequence of only using the support vectors to build a model, the model is less affected by outliers.

14.1 Knowledge Representation

The approach taken by a support vector machine model is to build a linear model; that is, to identify a line (in two dimensions) or a flat plane (in multiple dimensions) that separates observations with different values of the target variable. If we can find such a line or plane, then we find one that maximises the area between the two groups (when we are looking at binary classification, as with the *weather* dataset).

Consider a simple case where we have just two input variables (i.e., two-dimensional space). We will choose `Pressure3pm` and `Sunshine`. We will also purposefully select observations that will clearly demonstrate a significant separation (margin) between the observations for which it rains tomorrow and those for which it does not. The R code here illustrates our selection of the data and drawing of the plot. From the *weather* dataset, we only select observations that meet a couple of conditions to get two clumps of observations, one with `No` and the other with `Yes` for `RainTomorrow` (see Figure 14.1):

```
> library(rattle)
> obs <- with(weather, Pressure3pm+Sunshine > 1032 |
              (Pressure3pm+Sunshine < 1020 &
               RainTomorrow == "Yes"))
> ds <- weather[obs,]
> with(ds, plot(Pressure3pm, Sunshine,
                pch=as.integer(RainTomorrow),
                col=as.integer(RainTomorrow)+1))
> lines(c(1016.2, 1019.6), c(0, 12.7))
> lines(c(1032.8, 1001.5), c(0, 12.7))
> legend("topleft", c("Yes", "No"), pch=2:1, col=3:2)
```

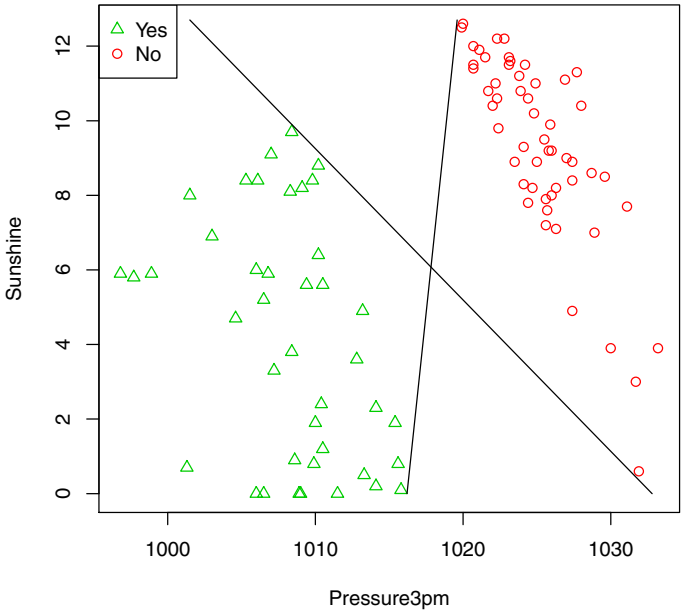


Figure 14.1: A simple and easily linearly separable collection of observations.

Two lines are shown in Figure 14.1 as two possible linear models. Taking either line as a model, the observations to the left will be classified as `Yes` and those to the right as `No`. However, there is an infinite collection of possible lines that we could draw to separate the two regions.

The support vector approach suggests that we find a line in the separating region such that we can make the line as thick as possible to butt up against the observations on the boundary. We choose the line that fills up the maximum amount of space between the two regions, as in Figure 14.2. The observations that butt up against this region are the support vectors.

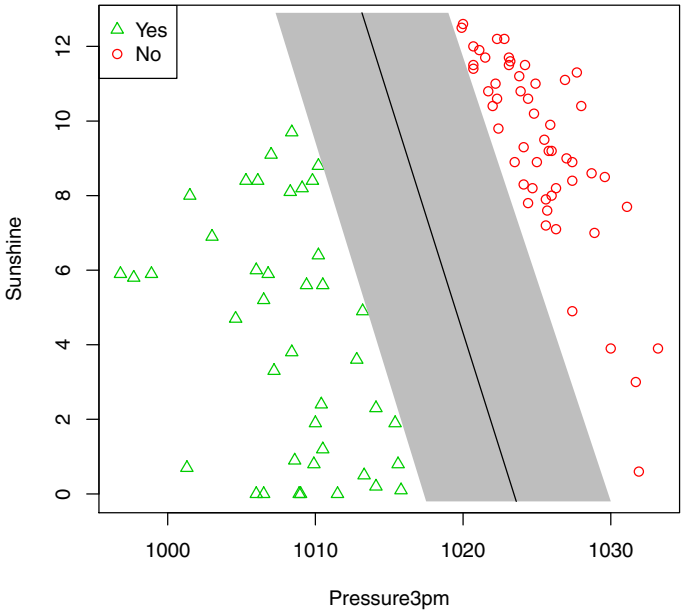


Figure 14.2: A maximal region or margin between the two classes of observations.

This is the representation of the model that we build using the approach of identifying support vectors and a maximal region between the classifications. The approach generalises to multiple dimensions (i.e., many input variables), where we search for hyperplanes that maximally fill the space between classes in the same way.

14.2 Algorithm

It is rarely the case that our observations are linearly separable. More likely, the data will appear as it does in Figure 14.3, which was generated directly from the data.

```
> ds <- weather
> with(ds, plot(Pressure3pm, Sunshine,
                pch=as.integer(RainTomorrow),
                col=as.integer(RainTomorrow)+1))
> legend("topleft", c("Yes", "No"), pch=2:1, col=3:2)
```

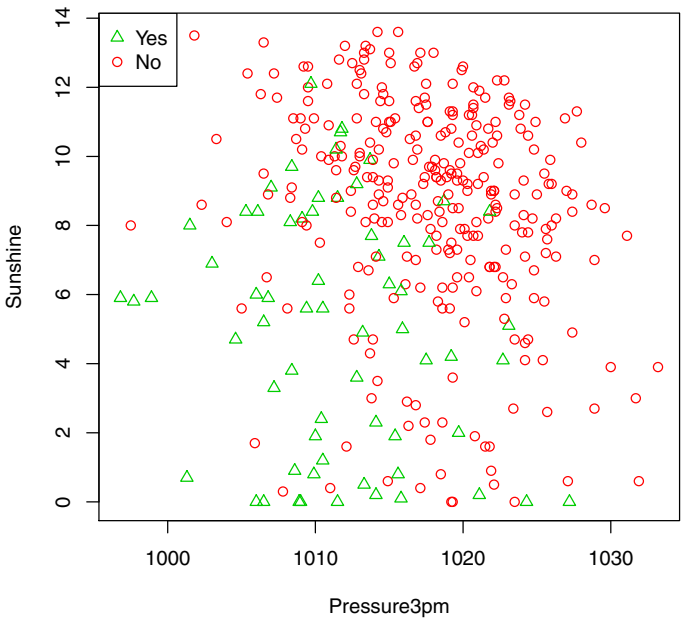


Figure 14.3: A nonlinearly separable collection of observations.

This kind of situation is where the kernel functions play a role. The idea is to introduce some other derived variables that are obtained from the input variables but combined and mathematically changed in some nonlinear way. Rather simple examples could be to add a new variable

which squares the value of `Pressure3pm` and another new variable that multiplies `Pressure3pm` by `Sunshine`. Adding such variables to the data can enhance separation. Figure 14.4 illustrates the resulting location of the observations, showing an improvement in separation (though to artificially exaggerate the improvement, not all points are shown).

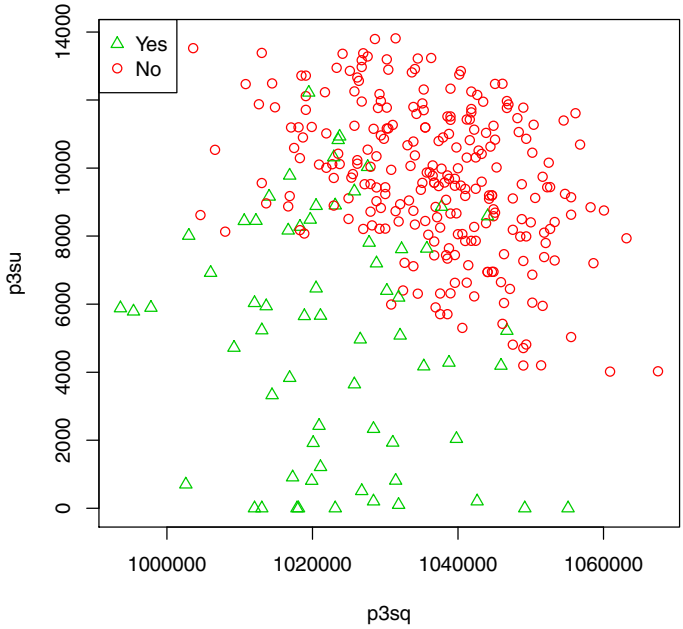


Figure 14.4: Nonlinearly transformed observations showing `Pressure3pm` squared (x-axis) against `Pressure3pm` multiplied by `Sunshine`, artificially enhanced.

A genuine benefit is likely to be seen when we add further variables to our dataset. It becomes difficult to display such multi-dimensional plots on the page, but tools like GGobican assist in visualising such data and confirming improved separation.

The basic kernel functions that are often used are the radial basis function, a linear function, and a polynomial function. The radial basis function uses a formula somewhat of the form $e^{-\gamma||x-x'||^2}$ for two observations x and x' (without going into the details). The polynomial function has the form $(1+\langle x, x'\rangle)^d$, for some integer d. For two input variables X_1

and X_2 and a power of 2, this becomes $(1+x_1x_1'+x_2x_2')^2$. Again, we skip the actual details of how such a formula is used. There are a variety of kernel functions available, but the commonly preferred one, and a good place to start, is the radial basis function.

Once the input variable space has been appropriately transformed, we then proceed to build a linear model as described in Section 14.1.

14.3 Tutorial Example

Building a Model Using Rattle

Rattle supports the building of support vector machine (SVM) models through **kernlab** (Karatzoglou et al., 2004). This package provides an extensive collection of kernel functions that can be applied to the data. This works by introducing new variables. Quite a variety of tuning options are provided by **kernlab**, but only a few are given through Rattle.

It is quite easy to experiment with different kernels using the *weather* dataset provided. The default kernel (radial basis function) is a good starting point. Such models can be quite accurate with no or little tuning. Two parameters are available for the radial basis function. `C=` is a penalty or cost parameter with a default value of 1. The Options widget can be used to set different values (e.g., `C=10`).

We review here the information provided to summarise the model, as displayed in Figure 14.5. The Textview begins with the summary of the model, identifying it as an object of class (or type) `ksvm` (kernel support vector machine):

```
Summary of the SVM model (built using ksvm):

Support Vector Machine object of class "ksvm"
```

The type of support vector machine is then identified. The `C-svc` indicates that the standard regularised support vector classification (svc) algorithm is used, with parameter `C` for tuning the algorithm. The value used for `C` is also reported:

```
SV type: C-svc  (classification)
 parameter : cost C = 1
```

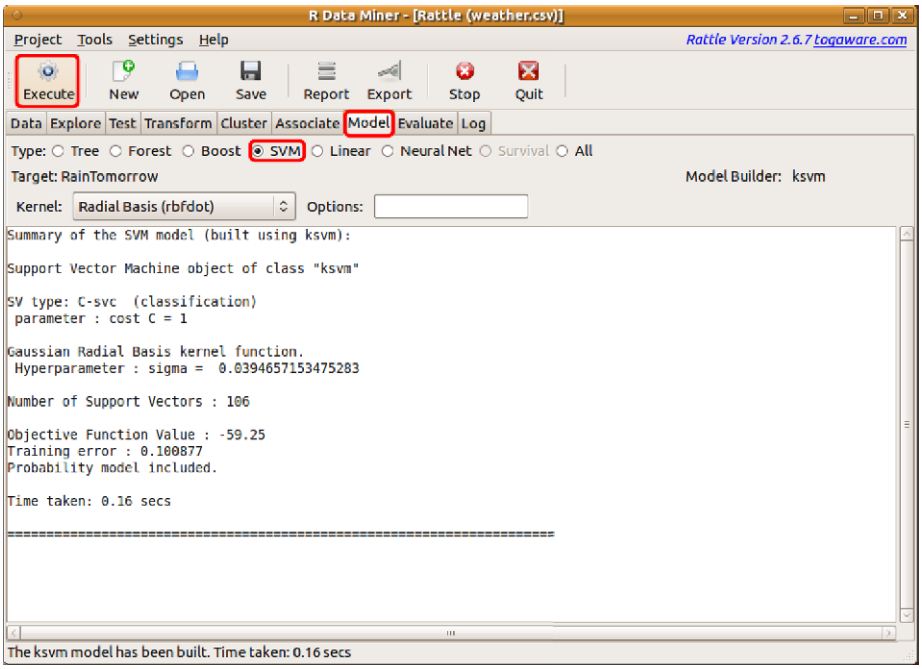


Figure 14.5: Building a support vector machine classification model.

An automatic algorithm is used to estimate another parameter (sigma) for the radial basis function kernel. The next two lines include an indication of the estimated value:

```
Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0394657153475283
```

The remaining lines report on the characteristics of the model, including how many observations are on the boundary (i.e., the support vectors), the value of the so-called objective function that the algorithm optimises, and the error calculated on the training dataset:

```
Number of Support Vectors : 106

Objective Function Value : -59.25
Training error : 0.100877
Probability model included.
```

Time Taken

The support vector machine is reasonably efficient:

```
Time taken: 0.16 secs
```

R

There is a wealth of functionality provided through **kernlab** and `ksvm()` for building support vector machine models. We will cover the basic functionality here. As usual, we begin with the dataset object from which we will be building our model:

```
> library(rattle)
> weatherDS <- new.env()
> evalq({
    data <- weather
    nobs <- nrow(weather)
    target <- "RainTomorrow"
    vars <- -grep('^(Date|Location|RISK_)', names(data))
    set.seed(42)
    train <- sample(nobs, 0.7*nobs)
    form <- formula(RainTomorrow ~ .)
  }, weatherDS)
```

We can now build the boosted model based on this dataset:

```
> library(kernlab)
> weatherSVM <- new.env(parent=weatherDS)
> evalq({
    model <- ksvm(form,
                  data=data[train, vars],
                  kernel="rbfdot",
                  prob.model=TRUE)
  }, weatherSVM)
```

The `kernel=` argument indicates that we will use the radial basis function as the kernel function. The `prob.model=` argument, set to `TRUE`, results in a model that predicts the probability of the outcomes. We obtain the usual overview of the model by simply printing its value:

```
> weatherSVM$model

Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0394335857291656

Number of Support Vectors : 107

Objective Function Value : -59.23
Training error : 0.100877
Probability model included.
```

14.4 Tuning Parameters

We describe here a number of tuning parameters, but many other options are available and are documented as part of **kernlab**.

Model Type `type=`

The `ksvm()` function can be used for a variety of modelling tasks, depending on the type of target variable. We are generally interested in classification tasks using the so-called `C-svc` formulation (support vector classification with a `C` parameter for tuning). This is a standard formulation for SVMs and is referred to as regularised support vector classification. Other options here include `nu-svc` for automatically regularised support vector classification, `one-svc` for novelty detection, `eps-svr` for support vector regression that is robust to small (i.e., epsilon) errors, and `nu-svr` for support vector regression that automatically minimises epsilon. Other options are available.

Kernel Function `kernel=`

The kernel method is the mechanism used to map our observations into a higher dimensional space. It is then within this higher dimensional space that the algorithm looks for a hyperplane that partitions our observations

to find the maximal margin between the different values of the target variable.

The `ksvm()` function supports a number of kernel functions. A good starting point is the radial basis function (using a Gaussian type of function). This is the `rfdot` option. The "`dot`" refers to the mathematical dot function or inner product between two vectors. This is integral to how support vector machines work, though not covered here. Other options include `polydot` for a polynomial kernel, `vanilladot` for a linear kernel, and `splinedot` for a spline kernel, amongst others.

Class Probabilities `prob.model=`

If this is set to `TRUE`, then the resulting model will calculate class probabilities.

Kernel Parameter: Cost of Constraints Violation `C=`

The cost parameter `C=` is by default 1. Larger values (e.g., 100 or 10,000) will consider more the points near the decision boundary, whilst smaller values relate to points further away from the decision boundary. Depending on the data, the choice of the cost argument may only play a small role in the resulting model.

Kernel Parameter: Sigma `sigma=`

For a radial basis function kernel, the sigma value can be set. Rattle uses automatic sigma estimation (using `sigest()`) for this kernel. This will find a good sigma value based on the data.

To experiment with various sigma values we can use the R code from Rattle's Log tab and paste that into the R Console and then add in the additional settings and run the model. This parameter tunes the kernel function selected, and so is listed as the `kparm=` list.

Cross Validation `cross=`

We can specify an integer value here to indicate whether to perform k-fold cross-validation.

14.5 Command Summary

This chapter has referenced the following R packages, commands, functions, and datasets:

kernlab	package	Kernel-based algorithms for machine learning.
`ksvm()`	function	Build an SVM model.
rattle	package	The *weather* dataset and GUI.
`sigest()`	function	Sigma estimation for kernel.
weather	dataset	Sample dataset from **rattle**.