Predictive Analytics: practical 3

The `OJ` data set

The `OJ` data set from the `ISLR` package contains information on which of two brands of orange juice customers purchased[1] and can be loaded using

```
data(OJ, package = "ISLR")
```

After loading the `caret` and `nclRpredictive` package

```
library("caret")
library("nclRpredictive")
```

make an initial examination of the relationships between each of the predictors and the response[2]

```
par(mfrow = c(4, 5), mar = c(4, 0.5, 0.5, 0.5))
plot(Purchase ~ ., data = OJ)
```

Initial model building

- To begin, create a logistic regression model that takes into consideration the prices of the two brands of orange juice, `PriceCH` and `PriceMM`.[3]
- What proportion of purchases does this model get right?
- How does this compare to if we used no model?

Visualising the boundary

The `nclRpredictive` package contains following code produces a plot of the decision boundary as seen in figure 1.

```
boundary_plot(m1,OJ$PriceCH, OJ$PriceMM, OJ$Purchase,
              xlab="Price CH", ylab="Price MM")
```

Run the boundary code above, and make sure you get a similar plot.

- What happens if we add an interaction term? How does the boundary change?
- Try adding polynomial terms.

Using all of the predictors

- Fit a logistic regression model using all of the predictors.
- Is there a problem?

  We can view the most recent warning messages by using the `warnings` function

[1] The response variable is `Purchase`.

[2] Use the `plot` function with a model formula or the `pairs` function.

[3] Hint: Use the `train` function, with `method = 'glm'`. Look at the help page for the data set to understand what these variables represent.

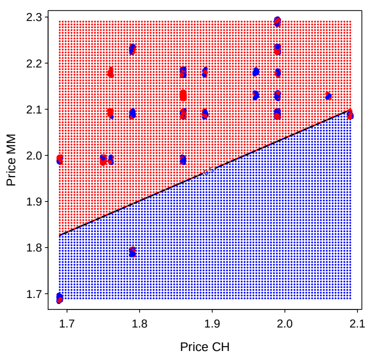

Figure 1: Examining the decision boundary for orange juice brand purchases by price.

```
warnings()
```

This suggests some rank–deficient fit problems,

- Look at the final model, you should notice that a number of parameters have not been estimated

  The help page

```
?ISLR::OJ
```

gives further insight: the `PriceDiff` variable is a linear combination of `SalePriceMM` and `SalePriceCH` so we should remove this. In addition the `StoreID` and `STORE` variable are different encodings of the same information so we should remove one of these too. We also have `DiscCH` and `DiscMM` which are the differences between `PriceCH` and `SalePriceCH` and `PriceMM` and `SalePriceMM` respectively and `ListPriceDiff` is a linear combination of these prices. Removing all of these variables allows the model to be fit and all parameters to be estimated.[4]

[4] This is to highlight that we need to understand what we have in our data.

```
OJsub = OJ[, !(colnames(OJ) %in% c("STORE", "SalePriceCH",
    "SalePriceMM", "PriceDiff", "ListPriceDiff"))]
OJsub$Store7 = as.numeric(OJsub$Store7) - 1
m_log = train(Purchase ~ ., data = OJsub, method = "glm")
```

The problem of linear combinations of predictors can be shown with this simple theoretical example. Suppose we have a response y and three predictors x_1, x_2 and the linear combination $x_3 = (x_1 + x_2)$. On fitting a linear model we try to find estimates of the parameters in the equation

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3(x_1 + x_2).$$

However we could just as easily rewrite this as

$$\begin{aligned} y &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3(x_1 + x_2) \\ &= \beta_0 + (\beta_1 + \beta_3)x_1 + (\beta_2 + \beta_3)x_2 \\ &= \beta_0 + \beta_1^* x_1 + \beta_2^* x_2. \end{aligned}$$

This leads to a rank–deficient model matrix, essentially we can never find the value of the β_3 due to the fact we have the linear combination of predictors.

We could achieve the same using the `caret` package function `findLinearCombos`. The function takes a model matrix as an argument. We can create such a matrix using the `model.matrix` function with our formula object

```
remove = findLinearCombos(model.matrix(Purchase ~ ., data = OJ))
```

The output list has a component called `remove` suggesting which variables should be removed to get rid of linear combinations

```
(badvar = colnames(OJ)[remove$remove])

## [1] "SalePriceMM"   "SalePriceCH"   "PriceDiff"
## [4] "ListPriceDiff" "STORE"

OJsub = OJ[, -remove$remove]
```

- How accurate is this new model using more predictors?]
- What are the values of sensitivity and specificity?
- What does this mean?

ROC curves

If we were interested in the area under the ROC curve, we could retrain the model using the `twoClassSummary` function as an argument to a train control object. Alternatively we can use the `pROC` package

```
library("pROC")
```

This also allows us to view the ROC curve, via

```
curve = roc(response = OJsub$Purchase,
  predictor = predict(m_log, type = "prob")[,"CH"])
## this makes CH the event of interest
plot(curve, legacy.axes = TRUE)
```

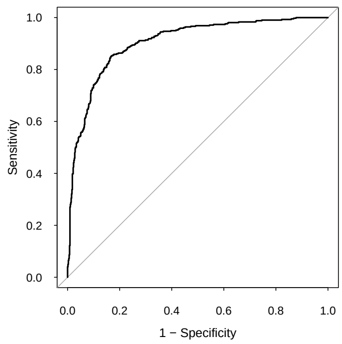

Figure 2: An example of a ROC curve for the logistic regression classifier. We can overlay ROC curves by adding the `add = TRUE` argument.

Other classification models

- Try fitting models using the other classification algorithms we have seen so far. To begin with, just have two covariates and use the `boundary_plot` function to visualise the results
- How do they compare?
- How does varying the number of nearest neighbours in a KNN affect the model fit?

We have seen LDA, QDA, KNN and logistic regression. Tomorrow we will cover support vector machines and neural nets; we can visualise the results in the same way.

The KNN algorithm described in the notes can also be used for regression problems. In this case the predicted response is the mean of the k nearest neighbours.

- Try fitting the KNN model for the regression problem in practical 1.
- How does this compare to the linear regression models?

Resampling methods

- Fit a KNN regression model to the `cars2010` data set with `FE` as the response.
- Estimate test error using the validation set approach explored at the beginning of the chapter

The data set can be loaded `data("FuelEconomy", package = "AppliedPredictiveModeling")`.

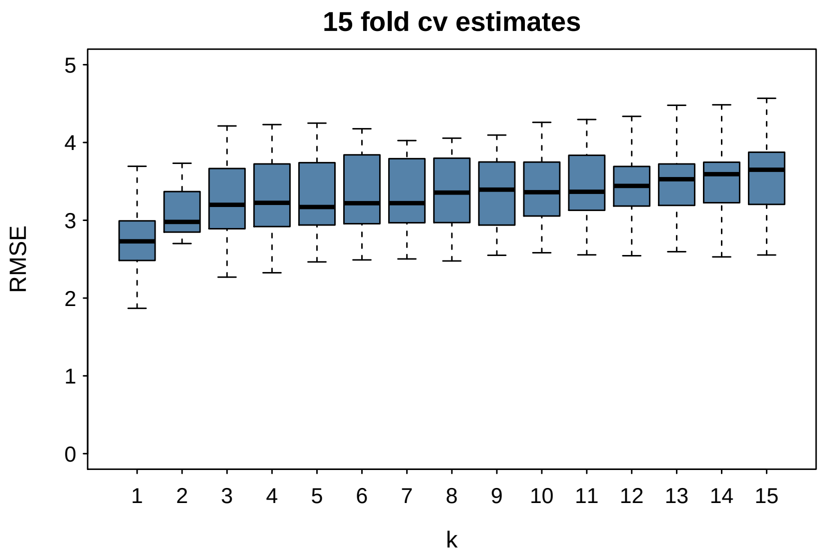

Figure 3: 15 fold cross validation estimates of RMSE in a K nearest neighbours model against number of nearest neighbours.

- Using the same validation set, estimate the performance of the k nearest neighbours algorithm for different values of k.
- Which model is chosen as the best when using the validation set approach?
- Create new `trainControl` objects to specify the use of 5 fold and 10 fold cross validation as well as bootstrapping to estimate test MSE.
- Go through the same training procedure attempting to find the best KNN model.
- How do the results vary based on the method of estimation?
- Are the conclusions always the same?

If we add the `returnResamp = "all"` argument in the trainControl function we can plot the resampling distributions, see figure 3.

```
tc = trainControl(method = "cv", number = 15, returnResamp = "all")
m = train(FE ~ ., data = cars2010, method = "knn", tuneGrid = data.frame(k = 1:15),
    trControl = tc)
boxplot(RMSE ~ k, data = m$resample)
```

We can overlay the information from each method using `add = TRUE`. In addition we could compare the computational cost of each of the methods. The output list from a `train` object contains timing information which can be accessed

```
m$time
```

- Which method is the most computationally efficient?

An example with more than two classes

The `Glass` data set in the `mlbench` package is a data frame containing examples of the chemical analysis of 7 different types of glass. The goal is to be able to predict which category glass falls into based on the values of the 9 predictors.

```
data(Glass, package = "mlbench")
```

A logistic regression model is typically not suitable for more than 2 classes, so try fitting the other models using a training set that consists of 90% of the available data.

The function `createDataPartition` can be used here, see notes for a reminder.