Predictive Analytics: practical 2 solutions

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, .5, .5, .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]

```
m1 = train(Purchase ~ PriceCH + PriceMM,
    data = OJ, method = "glm")
```

- What proportion of purchases does this model get right?

```
mean(predict(m1) != OJ$Purchase)

## [1] 0.3776
```

- How does this compare to if we used no model?

```
# with no model we essentially predict according to
# proportion of observations in data
probs = table(OJ$Purchase)/nrow(OJ)
preds = sample(levels(OJ$Purchase), prob = probs)
mean(preds != OJ$Purchase)

## [1] 0.5009
```

[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.

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")

## Error in localPlotWindow(xlim, ylim, ...):  formal argument
"xlab" matched by multiple actual arguments
```

```
## Error in localPlotWindow(xlim,
ylim, ...):  formal argument
"xlab" matched by multiple actual
arguments
```

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

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?

```
# We now have a curved decision boundary.  There are two
# regions of where we would predict MM, bottom left, and a
# tiny one up in the top right.
```

- Try adding polynomial terms.

Using all of the predictors

- Fit a logistic regression model using all of the predictors.

```
mLM = train(Purchase ~ ., data = OJ, method = "glm")
```

- Is there a problem?

```
## YES!
```

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

```
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

```
m_log$finalModel

##
## Call:  NULL
##
## Coefficients:
##    (Intercept)  WeekofPurchase         StoreID
##         5.1581         -0.0118         -0.1709
```

```
##        PriceCH         PriceMM          DiscCH
##         4.5865         -3.6249         10.7967
##         DiscMM       SpecialCH       SpecialMM
##        26.4615          0.2672          0.3169
##        LoyalCH     SalePriceMM     SalePriceCH
##        -6.3023              NA              NA
##      PriceDiff       Store7Yes       PctDiscMM
##             NA          0.3113        -50.6976
##      PctDiscCH    ListPriceDiff           STORE
##       -27.3399              NA              NA
##
## Degrees of Freedom: 1069 Total (i.e. Null);  1057 Residual
## Null Deviance:      1430
## Residual Deviance: 817   AIC: 843
```

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?]

```
# the corrected model
remove = findLinearCombos(model.matrix(Purchase~., data = OJ))
(badvar = colnames(OJ)[remove$remove])

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

OJsub = OJ[,-(remove$remove)]
mLM = train(Purchase~., data = OJsub, method = "glm")
mean(predict(mLM,OJsub) == OJsub$Purchase)

## [1] 0.8355
```

- What are the values of sensitivity and specificity?

```
## could use confusionMatrix
(cmLM = confusionMatrix(predict(mLM,OJsub),OJsub$Purchase))

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 577 100
##         MM  76 317
##
##                Accuracy : 0.836
##                  95% CI : (0.812, 0.857)
##     No Information Rate : 0.61
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.651
```

```
##  Mcnemar's Test P-Value : 0.083
##
##             Sensitivity : 0.884
##             Specificity : 0.760
##          Pos Pred Value : 0.852
##          Neg Pred Value : 0.807
##              Prevalence : 0.610
##          Detection Rate : 0.539
##    Detection Prevalence : 0.633
##       Balanced Accuracy : 0.822
##
##        'Positive' Class : CH
##

# or
sensitivity(predict(mLM,OJsub),OJsub$Purchase)

## [1] 0.8836

specificity(predict(mLM,OJsub),OJsub$Purchase)

## [1] 0.7602
```

- What does this mean?

```
# The model is fairly good at picking up both positive
# events, person buys CH, and negative events, MM.
```

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)
auc(curve)
```

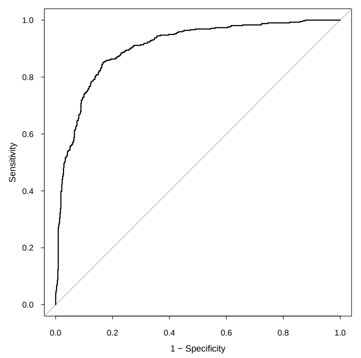

Figure 2: An example of a ROC curve for the logistic regression classifier. We can overlay ROC curves by adding the `add = TRUE` argument.

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

We have seen LDA, QDA, KNN and logistic regression.

```
mKNN = train(Purchase~., data = OJsub, method = "knn")
mLDA = train(Purchase~., data = OJsub, method = "lda")
mQDA = train(Purchase~., data = OJsub, method = "qda")
cmKNN = confusionMatrix(predict(mKNN,OJsub),OJsub$Purchase)
cmLDA = confusionMatrix(predict(mLDA,OJsub),OJsub$Purchase)
cmQDA = confusionMatrix(predict(mQDA,OJsub),OJsub$Purchase)
(info = data.frame(Model = c("logistic","knn","lda","qda"),
           Accuracy = c(cmLM$overall["Accuracy"],
               cmKNN$overall["Accuracy"],
               cmLDA$overall["Accuracy"],
               cmQDA$overall["Accuracy"]),
           Sensitivity = c(cmLM$byClass["Sensitivity"],
               cmKNN$byClass["Sensitivity"],
               cmLDA$byClass["Sensitivity"],
               cmQDA$byClass["Sensitivity"]),
           Specificity = c(cmLM$byClass["Specificity"],
               cmKNN$byClass["Specificity"],
               cmLDA$byClass["Specificity"],
               cmQDA$byClass["Specificity"])))

##      Model Accuracy Sensitivity Specificity
## 1 logistic   0.8355      0.8836      0.7602
## 2      knn   0.8065      0.8943      0.6691
## 3      lda   0.8374      0.8790      0.7722
## 4      qda   0.8168      0.8407      0.7794
```

- How do they compare?

```
#Logistic regression and LDA have highest accuracy, QDA is poorest at classifying events, KNN gives m
```

- How does varying the number of nearest neighbours in a KNN affect the model fit?

```
# Accuracy increases at first with knn before then getting
# worse after a peak value of 9.
(mKNN2 = train(Purchase ~ ., data = OJsub, method = "knn",
    tuneGrid = data.frame(k = 1:30)))

## k-Nearest Neighbors
##
## 1070 samples
##   12 predictors
##    2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 1070, 1070, 1070, 1070, 1070, 1070, ...
##
```

```
## Resampling results across tuning parameters:
##
##   k   Accuracy  Kappa   Accuracy SD  Kappa SD
##    1  0.6899    0.3488  0.01841      0.03965
##    2  0.6819    0.3330  0.02358      0.04886
##    3  0.6875    0.3429  0.02288      0.04676
##    4  0.6976    0.3623  0.02425      0.04919
##    5  0.7035    0.3720  0.02367      0.04591
##    6  0.7028    0.3706  0.02484      0.05051
##    7  0.7092    0.3832  0.02230      0.04454
##    8  0.7059    0.3753  0.02348      0.04835
##    9  0.7027    0.3680  0.02346      0.04739
##   10  0.7021    0.3642  0.02209      0.04608
##   11  0.6995    0.3575  0.02507      0.05257
##   12  0.6970    0.3529  0.02308      0.04998
##   13  0.6929    0.3435  0.01895      0.03753
##   14  0.6898    0.3366  0.01736      0.03794
##   15  0.6899    0.3361  0.02210      0.04810
##   16  0.6870    0.3293  0.02385      0.05150
##   17  0.6845    0.3231  0.02298      0.05053
##   18  0.6803    0.3144  0.02270      0.04846
##   19  0.6794    0.3122  0.02381      0.04969
##   20  0.6786    0.3086  0.02060      0.04550
##   21  0.6753    0.3007  0.01956      0.04538
##   22  0.6789    0.3082  0.02077      0.04708
##   23  0.6781    0.3054  0.02263      0.05287
##   24  0.6797    0.3087  0.02104      0.04802
##   25  0.6774    0.3025  0.02113      0.04835
##   26  0.6764    0.3000  0.02166      0.05089
##   27  0.6738    0.2944  0.01970      0.04519
##   28  0.6727    0.2919  0.02210      0.04973
##   29  0.6706    0.2861  0.02284      0.05242
##   30  0.6727    0.2904  0.02045      0.04555
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was k = 7.
```

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.

```r
library("nclRpredictive")
data(FuelEconomy, package = "AppliedPredictiveModeling")
regKNN = train(FE~., data = cars2010, method = "knn")
regLM = train(FE~., data = cars2010, method = "lm")
```

```
regKNN= validate(regKNN)
regLM = validate(regLM)
mark(regKNN)
mark(regLM)
```

- How does this compare to the linear regression models?

```
# The KNN regression model is not as good as the linear
# model at predicting the test set. It overestimates more
# at the lower end.
```

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.