Predictive Analytics: practical 2solutions

The `OJ` data set

The `OJ` dataset of the `ISLR` package contains information on which of two brands of orange juice customers purchased and can be loaded in using

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

After loading the `caret` package

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

Make an initial examination of the relationships between each of the predictors and the response[1]

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

```
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`.[2]

[2] Hint: The train function does model fitting, the method argument specifies the type of model. `method = "glm"` is used for logistic regression.

```
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] 0.4991
```

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

```
## Set up a grid for prediction
chrange = range(OJ$PriceCH)
mmrange = range(OJ$PriceMM)
chseq = seq(chrange[1],chrange[2],length.out = 100)
mmseq = seq(mmrange[1],mmrange[2],length.out = 100)
grid = expand.grid("PriceCH" = chseq, "PriceMM" = mmseq)

# make the predictions
predictions = predict(m1,grid,type = "prob")
# turn the predictions into a matrix for a contour plot
predmat = matrix(predictions[,2], nrow=100)
contour(chseq, mmseq, predmat, levels = 0.5,
        xlab = "Price CH", ylab = "Price MM",
        lwd = 2, main = "Blue = MM")

# the background points indicating prediction
points(grid,col = c("red","blue")[predict(m1,grid)],
       cex = 0.2)
# there are few unique combinations of prices,
# jitter can help see the points
# points of prices coloured by purchased brand
points(jitter(OJ$PriceCH,factor = 2),
       jitter(OJ$PriceMM, factor = 2),
       col = c("red","blue")[OJ$Purchase],
       pch = 19, cex  =0.6)

# add dashed line of equal price
abline(0,1,lwd = 2, lty = 2)
```

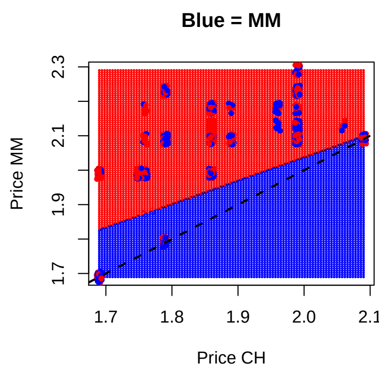

Figure 1: Examining the decision boundary for orange juice brand purchases by price.

- 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.
```

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()

## NULL
```

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         PriceCH
##         5.1581         -0.0118         -0.1709          4.5865
##        PriceMM          DiscCH          DiscMM       SpecialCH
##        -3.6249         10.7967         26.4615          0.2672
##      SpecialMM          LoyalCH     SalePriceMM     SalePriceCH
##         0.3169         -6.3023              NA              NA
##      PriceDiff       Store7Yes       PctDiscMM       PctDiscCH
##             NA          0.3113        -50.6976        -27.3399
##  ListPriceDiff           STORE
##             NA              NA
##
## Degrees of Freedom: 1069 Total (i.e. Null);  1057 Residual
## Null Deviance:     1430
## Residual Deviance: 817  AIC: 843
```

```
?ISLR::OJ
```

gives further insight, the `PriceDiff` variable is a linear combination of `SalePriceMM` and `SalePriceCH` so we should remove this. In addition we have a `StoreID` variable and a `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 vairables allows the model to be fit and all parameters to be estimated.[3]

[3] 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$Store7 = as.double(OJsub$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 $(x_1 + x_2)$. 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 acheive 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"     "ListPriceDiff"
## [5] "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"     "ListPriceDiff"
## [5] "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.
```

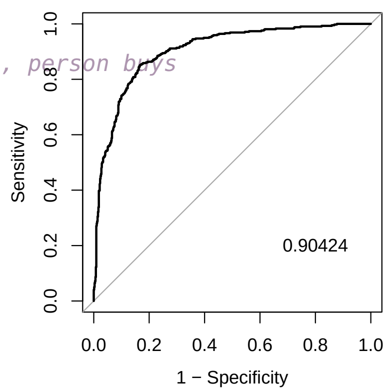

Figure 2: An example of a ROC curve for the logistic regression classifier. We can overlay ROC curves by adding the `add = TRUE` argument.

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 `roc` function in the `pROC` package. This also allows us to view the ROC curve, see figure 2.

```
library("pROC")

## Type 'citation("pROC")' for a citation.
##
## Attaching package:  'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

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)
```

Other classification models

- Try fitting models using the other classification algorithms we have seen so far.

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   0.8103      0.8775      0.7050
## 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  0.6865    0.3429  0.01905      0.04086
##    2  0.6789    0.3256  0.01482      0.03211
##    3  0.6878    0.3433  0.01607      0.03650
##    4  0.6903    0.3473  0.02142      0.04852
##    5  0.6972    0.3599  0.01817      0.03885
##    6  0.6950    0.3536  0.02107      0.04507
##    7  0.7024    0.3673  0.02182      0.04477
##    8  0.7043    0.3705  0.01854      0.03940
##    9  0.7043    0.3700  0.02122      0.04357
##   10  0.7028    0.3647  0.02102      0.04496
##   11  0.7028    0.3624  0.02284      0.04627
##   12  0.7003    0.3570  0.02264      0.04720
##   13  0.6980    0.3521  0.02138      0.04282
##   14  0.6934    0.3425  0.02118      0.04166
##   15  0.6891    0.3334  0.02144      0.04248
##   16  0.6863    0.3254  0.02116      0.04236
##   17  0.6801    0.3108  0.02387      0.05030
##   18  0.6800    0.3108  0.02070      0.04282
##   19  0.6758    0.3011  0.02541      0.05330
##   20  0.6722    0.2918  0.02509      0.05249
##   21  0.6771    0.3014  0.02209      0.04607
##   22  0.6782    0.3052  0.02544      0.05194
##   23  0.6755    0.2985  0.02206      0.04714
##   24  0.6752    0.2973  0.02209      0.04561
##   25  0.6763    0.2992  0.02249      0.04667
```

```
##   26  0.6714    0.2882  0.01968      0.03993
##   27  0.6749    0.2953  0.01915      0.03983
##   28  0.6700    0.2843  0.02100      0.04432
##   29  0.6694    0.2828  0.02126      0.04416
##   30  0.6677    0.2812  0.02121      0.04395
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was k = 8.
```

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.

```
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.