

STAT656: Homework 3

Classification with Logistic Regression and Linear Discriminant Analysis

When customers leave a company, this is often referred to as ‘churn’. Churn is a big concern for many service-oriented businesses as there is a cost of acquiring new customers, so keeping existing customers is important to keep costs down. Let’s look into predicting whether a customer will churn for a telecommunications company.

Let’s load in any required packages here (make sure to install them first)

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
```

```
##
```

```
## filter, lag
```

```
## The following objects are masked from 'package:base':
```

```
##
```

```
## intersect, setdiff, setequal, union
```

```
require(readr)
```

```
## Loading required package: readr
```

```
require(caret)
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
require(pROC)
```

```
## Loading required package: pROC
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
set.seed(1)

trainingData = read_csv('trainingData.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   SeniorCitizen = col_double(),
##   tenure = col_double(),
##   MonthlyCharges = col_double(),
##   TotalCharges = col_double()
## )

## See spec(...) for full column specifications.
```

```
testingData = read_csv('testingData.csv')
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   SeniorCitizen = col_double(),
##   tenure = col_double(),
##   MonthlyCharges = col_double(),
##   TotalCharges = col_double()
## )

## See spec(...) for full column specifications.
```

Let's modify the objects somewhat to get a consistent naming convention

```
Xtrain = select(trainingData, -Churn)
Xtest = select(testingData, -Churn)
Ytrain = select(trainingData, Churn) %>% unlist()
Ytest = select(testingData, Churn) %>% unlist()
```

Let's look at the data structures for the training data:

```
str(Xtrain)

## tibble [6,340 x 19] (S3: tbl_df/tbl/data.frame)
## $ customerID      : chr [1:6340] "5575-GNVDE" "3668-QPYBK" "7795-CFOCW" "9237-HQITU" ...
## $ gender          : chr [1:6340] "Male" "Male" "Male" "Female" ...
## $ SeniorCitizen   : num [1:6340] 0 0 0 0 0 0 0 0 0 0 ...
## $ Partner         : chr [1:6340] "No" "No" "No" "No" ...
## $ Dependents      : chr [1:6340] "No" "No" "No" "No" ...
```

```
## $ tenure      : num [1:6340] 34 2 45 2 8 22 10 62 13 16 ...
## $ MultipleLines : chr [1:6340] "No" "No" "No phone service" "No" ...
## $ InternetService : chr [1:6340] "DSL" "DSL" "DSL" "Fiber optic" ...
## $ OnlineSecurity : chr [1:6340] "Yes" "Yes" "Yes" "No" ...
## $ OnlineBackup : chr [1:6340] "No" "Yes" "No" "No" ...
## $ DeviceProtection: chr [1:6340] "Yes" "No" "Yes" "No" ...
## $ TechSupport : chr [1:6340] "No" "No" "Yes" "No" ...
## $ StreamingTV : chr [1:6340] "No" "No" "No" "No" ...
## $ StreamingMovies : chr [1:6340] "No" "No" "No" "No" ...
## $ Contract : chr [1:6340] "One year" "Month-to-month" "One year" "Month-to-month" ...
## $ PaperlessBilling: chr [1:6340] "No" "Yes" "No" "Yes" ...
## $ PaymentMethod : chr [1:6340] "Mailed check" "Mailed check" "Bank transfer (automatic)" "Electron
## $ MonthlyCharges : num [1:6340] 57 53.9 42.3 70.7 99.7 ...
## $ TotalCharges : num [1:6340] 1890 108 1841 152 820 ...
```

#Note: we can put the number of unique values with the data structure:

```
rbind(sapply(Xtrain,function(x){ length(unique(x))}),
      sapply(Xtrain,class))
```

```
##      customerID gender      SeniorCitizen Partner      Dependents tenure
## [1,] "6340"      "2"        "2"              "2"        "2"        "73"
## [2,] "character" "character" "numeric"      "character" "character" "numeric"
##      MultipleLines InternetService OnlineSecurity OnlineBackup DeviceProtection
## [1,] "3"          "3"            "3"            "3"            "3"
## [2,] "character"  "character"  "character"    "character"    "character"
##      TechSupport StreamingTV StreamingMovies Contract      PaperlessBilling
## [1,] "3"          "3"            "3"            "3"            "2"
## [2,] "character"  "character"  "character"    "character"    "character"
##      PaymentMethod MonthlyCharges TotalCharges
## [1,] "4"          "1546"        "5918"
## [2,] "character"  "numeric"    "numeric"
```

```
str(Ytrain)
```

```
## Named chr [1:6340] "No" "Yes" "No" "Yes" "Yes" "No" "No" "No" "No" "No" ...
## - attr(*, "names")= chr [1:6340] "Churn1" "Churn2" "Churn3" "Churn4" ...
```

```
table(Ytrain)
```

```
## Ytrain
## No Yes
## 4657 1683
```

Problem 0 (15 pts)

Remember the items that usually need to be checked:

- data structures
- checking for missing data
- Converting qualitative features to dummy variables

- extreme observations
- transformations
- correlations

For this assignment, in the interest of keeping it short(er), let's just look at data structures and missing data

0.1 Missing data

```
anyNA(Xtrain)
```

```
## [1] TRUE
```

```
anyNA(Ytrain)#### Answer 0.1.1 Check to see if there are any missing values in the training data
```

```
## [1] FALSE
```

```
anyNA(Xtest)
```

```
## [1] TRUE
```

```
anyNA(Ytest)#### Answer 0.1.2 Check to see if there are any missing values in the test data
```

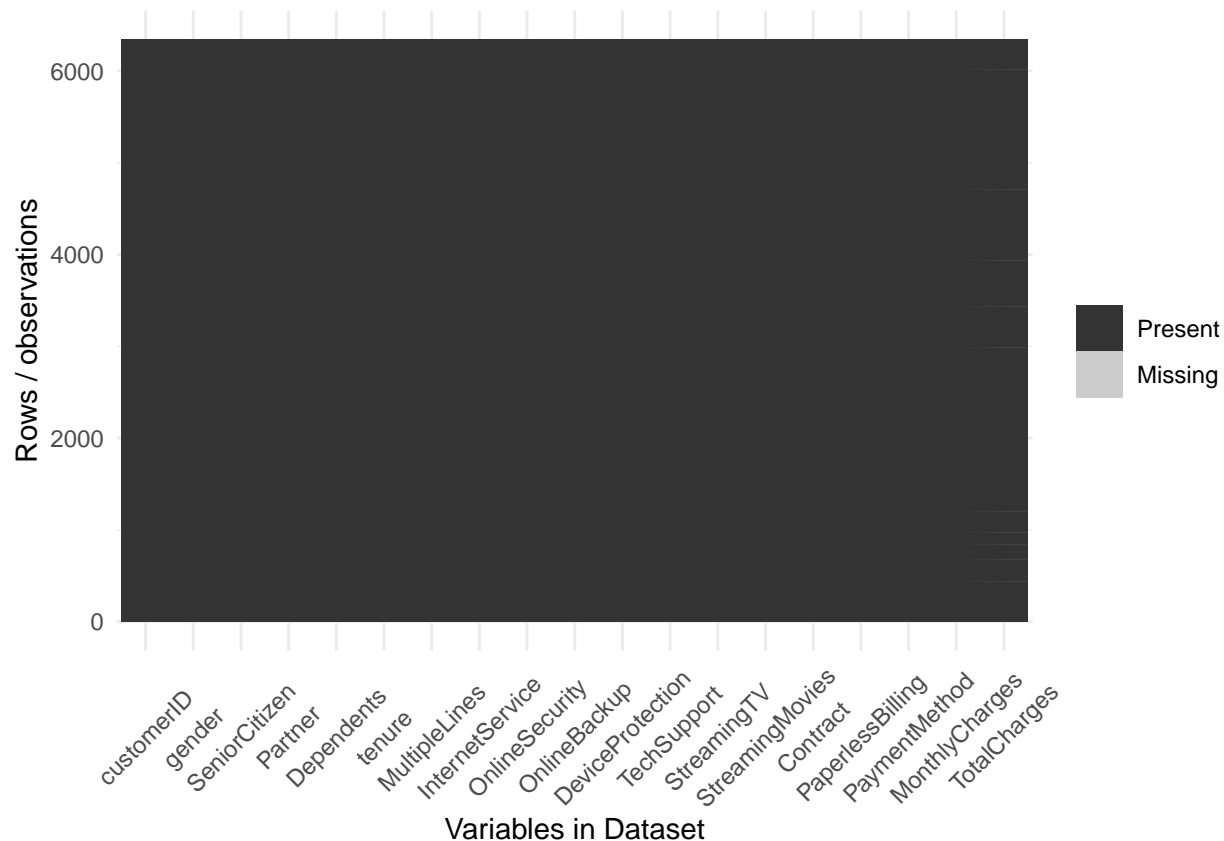
```
## [1] FALSE
```

```
ggplot_missing <- function(x){
  if(!require(reshape2)){warning('you need to install reshape2')}
  require(reshape2)
  require(ggplot2)

  x %>%
    is.na %>%
    melt %>%
    ggplot(data = .,
           aes(x = Var2,
               y = Var1)) +
    geom_raster(aes(fill = value)) +
    scale_fill_grey(name = "",
                   labels = c("Present", "Missing")) +
    theme_minimal() +
    theme(axis.text.x = element_text(angle=45, vjust=0.5)) +
    labs(x = "Variables in Dataset",
         y = "Rows / observations")
}
```

```
ggplot_missing(Xtrain)#### Answer 0.1.3 Make a visualization with the function ggplot_missing of the mi.
```

```
## Loading required package: reshape2
```



It looks like there are just missing values for TotalCharges in both Xtrain and Xtest. Let's use a linear regression model to impute TotalCharges with MonthlyCharges. Here is how that would work using the training data

```
trControl = trainControl(method='none')

imputationScheme = train(TotalCharges~MonthlyCharges,
                          data = select(Xtrain,MonthlyCharges, TotalCharges) %>% na.omit,
                          method = 'lm', trControl = trControl)

XtrainImp      = Xtrain
M              = is.na(XtrainImp$TotalCharges)
XtrainImp$TotalCharges[M] = predict(imputationScheme,
                                    select(Xtrain, MonthlyCharges, TotalCharges) %>% filter(M))
```

Now, we want to impute the test features using the imputation scheme learned using the training features.

```
XtestImp = Xtest
N        = is.na(XtestImp$TotalCharges)
XtestImp$TotalCharges[N] = predict(imputationScheme,
                                   select(Xtest, MonthlyCharges, TotalCharges) %>% filter(N))

#### Answer 0.1.4 Using 'imputationScheme' from 0.1.3, impute the missing value(s) in Xtest
#### Don't retrain a new imputation scheme on the test data, just use the training
#### imputation scheme
```

0.2 Data structures

Now, convert the qualitative features and supervisors to factors.

Hint: you should be on the look out for qualitative features that have too many levels. Encoding them as factors creates a massive number of dummy variables. Also, of course, qualitative features that have the wrong data structure e.g. integer or numeric.

```
XtrainImpFact = select(XtrainImp, -customerID,
                      -tenure, -MonthlyCharges, -TotalCharges) %>%
  mutate_all(factor)

XtestImpFact = select(XtestImp, -customerID,
                     -tenure, -MonthlyCharges, -TotalCharges) %>%
  mutate_all(factor)#### Answer 0.2.1 Convert the character data structures in Xtest to fa

Ytrain  = factor(Ytrain)

Ytest   = factor(Ytest)#### Answer 0.2.2 Convert the character data structures in Ytest to factor
```

0.3 Dummy variables

```
dummyModel = dummyVars(~ ., data = XtrainImpFact, fullRank = TRUE)

XtrainQualDummy = predict(dummyModel, XtrainImpFact)
XtestQualDummy  = predict(dummyModel, XtestImpFact)
```

Let's create the full feature matrices for the training and test set

```
XtrainQuan = select(XtrainImp, tenure, MonthlyCharges, TotalCharges)
XtrainFull = cbind(XtrainQualDummy, XtrainQuan)

XtestQuan  = select(XtestImp, tenure, MonthlyCharges, TotalCharges)
XtestFull  = cbind(XtestQualDummy, XtestQuan)
```

Some additional processing

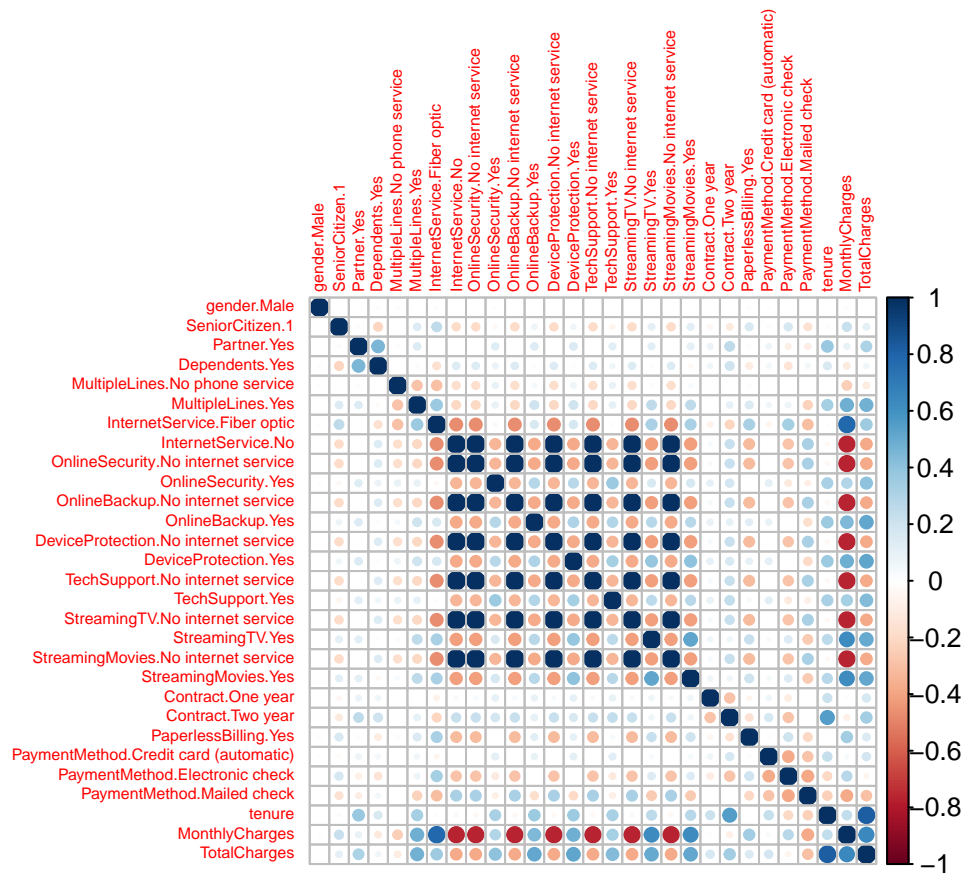
Note that there is some redundant information in the qualitative features. Though it isn't really "correlation" (because we are referring to indicator features), we can still use a corrplot to visualize the problem

```
require(corrplot)

## Loading required package: corrplot

## corrplot 0.84 loaded

corrplot(cor(XtrainFull), tl.cex = 0.5)
```



There are qualitative features like “InternetService” and then other qualitative features like “OnlineBackup” which have a level “no internet service”. These new features overlap exactly and hence we need to remove some of them

```
XtrainFull = select(XtrainFull,-contains('.No internet'))
XtestFull  = select(XtestFull,-contains('.No internet'))
```

Problem 1: Logistic Regression (30 pts)

Let’s go through and make a predictive model out of logistic regression

Train the logistic regression model on the training data

Make sure you are using the trainControl to only train the model by setting method = ‘none’. ‘train’ treats the first level as the event of interest, which is in alphabetical order. So, ‘no’ would be the event. However, we usually want to code results so that the outcome of interest is the event. We can make this adjustment in R via ‘relevel’ on the supervisor

```
YtrainRelevel = relevel(Ytrain, ref = 'Yes')
YtestRelevel  = relevel(Ytest, ref = 'Yes')

trControl     = trainControl(method = 'none')
outLogistic   = train(x = XtrainFull, y = YtrainRelevel,
                      method = 'glm', trControl = trControl)
```

Problem 1.2: Get the test predicted probabilities

Get the predicted probabilities for the test data and print out the first few predictions with the ‘head’ function

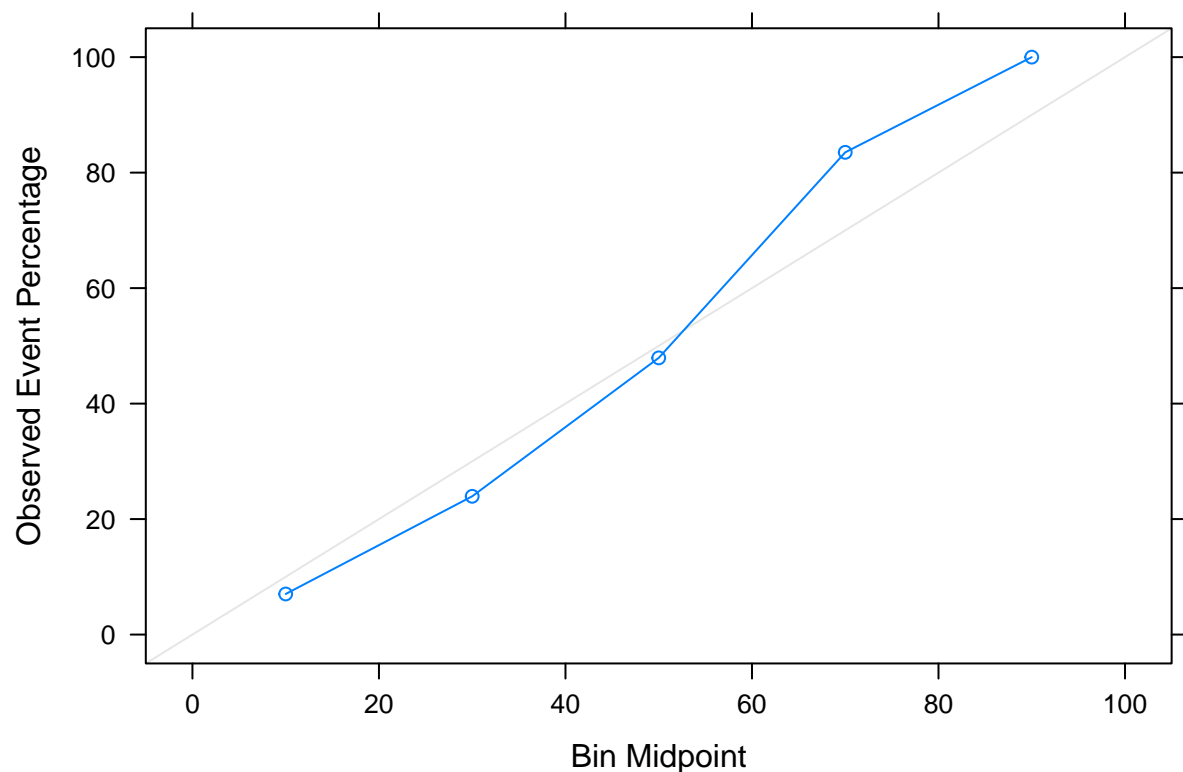
```
YhatTestProb = predict(outLogistic, XtestFull, type = 'prob')#### Answer 1.2.1.  
head(YhatTestProb)
```

```
##           Yes           No  
## 1 0.62959261 0.3704074  
## 2 0.60063123 0.3993688  
## 3 0.02781752 0.9721825  
## 4 0.50092343 0.4990766  
## 5 0.16875929 0.8312407  
## 6 0.36771916 0.6322808
```

Problem 1.3: Well-calibrated probabilities

Produce a calibration plot using your predictions of the test Data

```
calibProbs = calibration(YtestRelevel ~ YhatTestProb$Yes, cuts = 5)  
xyplot(calibProbs)
```



Answer 1.3.1. Are these well calibrated probabilities?

Each bin in the plot closely follows the diagonal, these are well calibrated probabilities.

Problem 1.4: The Confusion matrix

Get the classifications now using the default threshold.

```
YhatTest = predict(outLogistic, XtestFull, type = 'raw')
```

Look at the help file for the function ‘confusionMatrix’ function. Instead of producing all the output as in the lectures, produce only the confusion matrix, the accuracy rate, kappa, the sensitivity, and the specificity.

```
confusionMatrixOut = confusionMatrix(reference = YtestRelevel, data = YhatTest)

print(confusionMatrixOut$table)
```

```
##           Reference
## Prediction Yes  No
##           Yes 114  35
##           No  72 482
```

```
print(confusionMatrixOut$overall[1:2])
```

```
## Accuracy      Kappa
## 0.8477952 0.5822842
```

```
print(confusionMatrixOut$byClass[1:2])
```

```
## Sensitivity Specificity
## 0.6129032 0.9323017
```

```
logRegAcc = confusionMatrixOut$overall[1]#### Answer 1.4.1
```

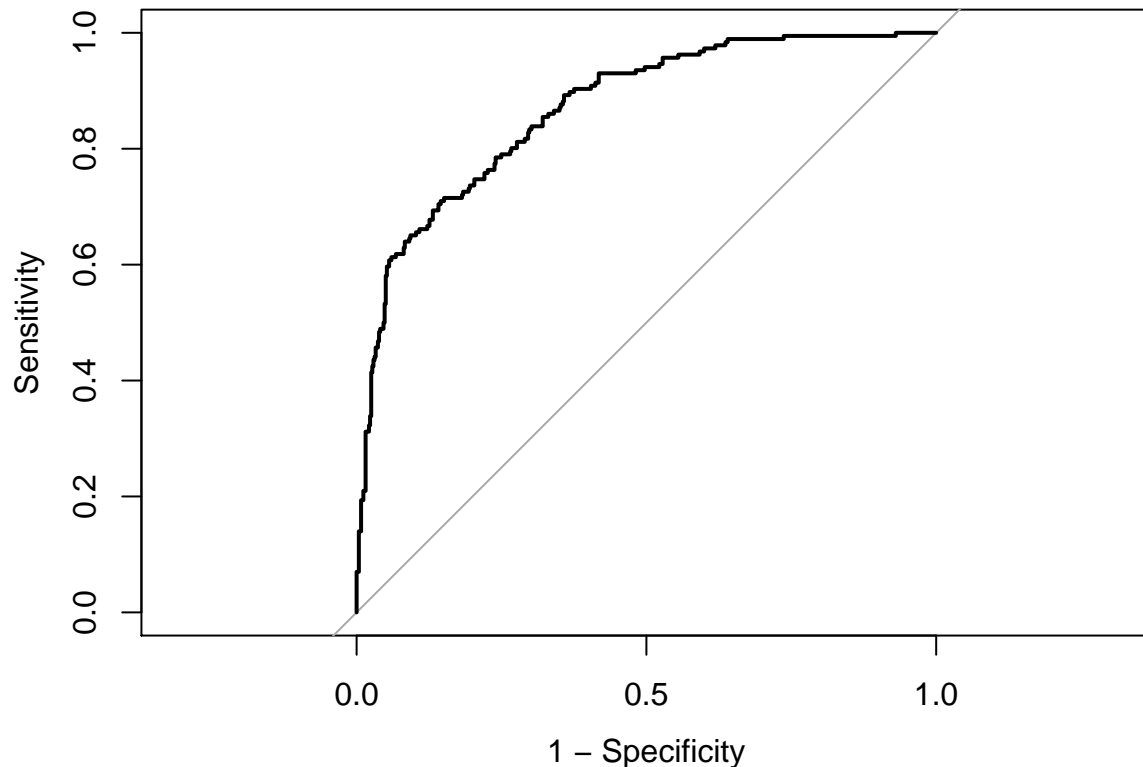
Problem 1.5: Produce the ROC curve using the test data

```
rocCurve = roc(Ytest, YhatTestProb$Yes)
```

```
## Setting levels: control = No, case = Yes
```

```
## Setting direction: controls < cases
```

```
plot(rocCurve, legacy.axes=TRUE)
```



Answer 1.5.1.

In no more than 2 sentences, briefly describe this curve. What the objects are being plotted? What value for the threshold would put a classifier at the point (0,0)?

ROC curve shows the relationship between sensitivity (true positive fraction) and 1 - specificity (false positive fraction) for each possible cut-offs. Threshold value of 1 would put the classifier at the point (0,0)

Problem 1.6: AUC

We can get a one number summary of the ROC from the 'rocCurve' object: auc

```
logRegAUC = rocCurve$auc
logRegAUC
```

```
## Area under the curve: 0.8689
```

Answer 1.6.1

Problem 1.7: Achieving a particular sensitivity

What is the specificity for a model that has a sensitivity of at least 0.8? What threshold does it occur at?

```

thresholds = rocCurve$thresholds

pt8        = which.min(rocCurve$sensitivities >= 0.8) #Answer 1.7.1. Fill in the appropriate value

threshold  = thresholds[pt8]
specificity = rocCurve$specificities[pt8]

```

The specificity is 0.7330754. It occurs at a threshold of 0.250096.

Problem 2: Linear Discriminant Analysis (LDA) (20 pts)

Let's do the same thing for LDA. The code should be nearly identical (hence the power of the 'caret' package).

Train the LDA model on the training data

Make sure you are using the trainControl to only train the model by setting method = 'none'. 'train' treats the first level as the event of interest, which is in alphabetical order. So, 'no' would be the event. However, we usually want to code results so that the outcome of interest is the event. We can make this adjustment in R via 'relevel' on the supervisor

```

YtrainRelevel = relevel(Ytrain, ref = 'Yes')
YtestRelevel  = relevel(Ytest, ref = 'Yes')

trControl = trainControl(method = 'none')
outLDA     = train(x = XtrainFull, y = YtrainRelevel,
                  method = 'lda', trControl = trControl)

```

Get the test predicted probabilities

```

YhatTestProb = predict(outLDA, XtestFull, type = 'prob')
head(YhatTestProb)

```

```

##           Yes           No
## 1 0.48308863 0.5169114
## 2 0.62943861 0.3705614
## 3 0.05668856 0.9433114
## 4 0.53500672 0.4649933
## 5 0.13409761 0.8659024
## 6 0.29653606 0.7034639

```

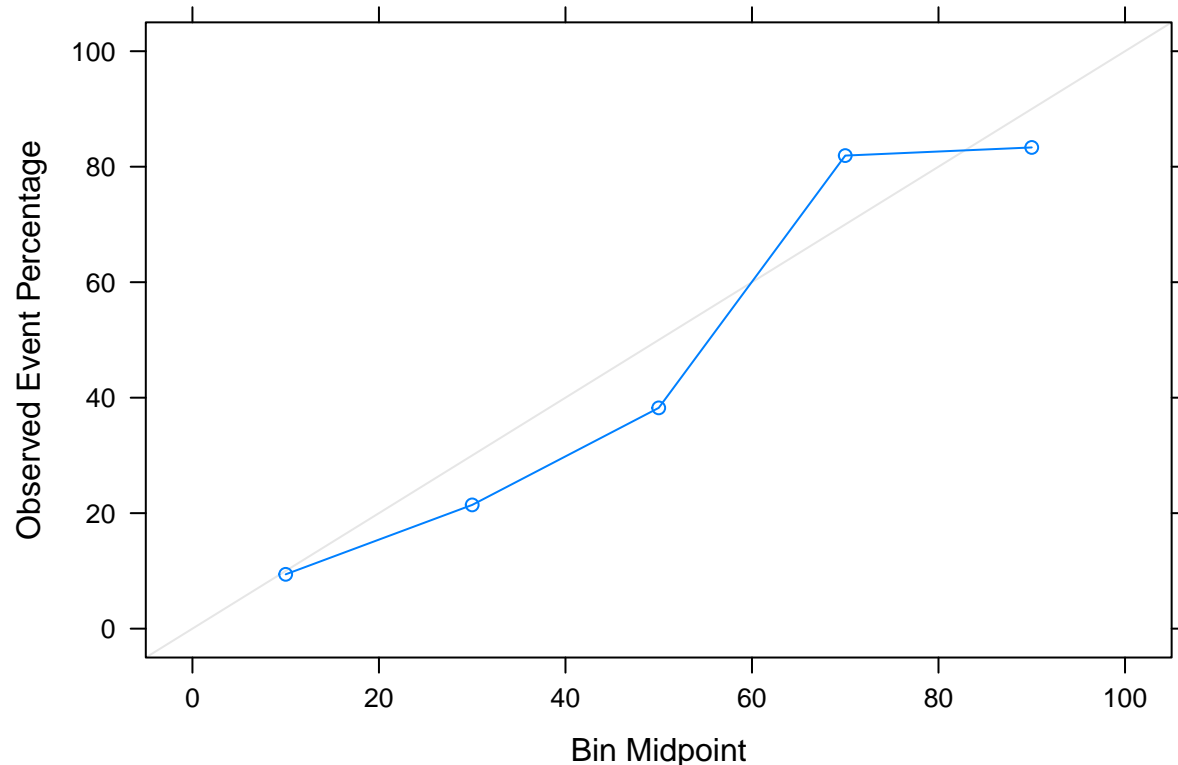
Problem 2.1: Well-calibrated probabilities

Produce a calibration plot using your predictions of the test Data

```

calibProbs = calibration(YtestRelevel ~ YhatTestProb$Yes, cuts = 5)
xyplot(calibProbs)

```



Answer 2.1.1. Are these well calibrated probabilities?

Compared to Logistic regression, at some bins LDA underestimates and in some bins it overestimates the true proportion. But in overall perspective these are well calibrated probabilities.

Problem 2.2: The Confusion matrix

Get the classifications now using the default threshold.

```
YhatTest = predict(outLDA, XtestFull, type = 'raw')
```

Look at the help file for the function ‘confusionMatrix’ function. Instead of producing all the output as in the lectures, produce only the confusion matrix, the accuracy rate, kappa, the sensitivity, and the specificity.

```
confusionMatrixOutLDA = confusionMatrix(reference = YtestRelevel, data = YhatTest)
print(confusionMatrixOutLDA$table)
```

```
##           Reference
## Prediction Yes  No
##           Yes 111 40
##           No  75 477
```

```
print(confusionMatrixOutLDA$overall[1:2])
```

```
## Accuracy      Kappa
## 0.8364154 0.5526975
```

```
print(confusionMatrixOutLDA$byClass[1:2])
```

```
## Sensitivity Specificity
## 0.5967742 0.9226306
```

```
LDAacc = confusionMatrixOutLDA$overall[1]#### Answer 2.2.1
```

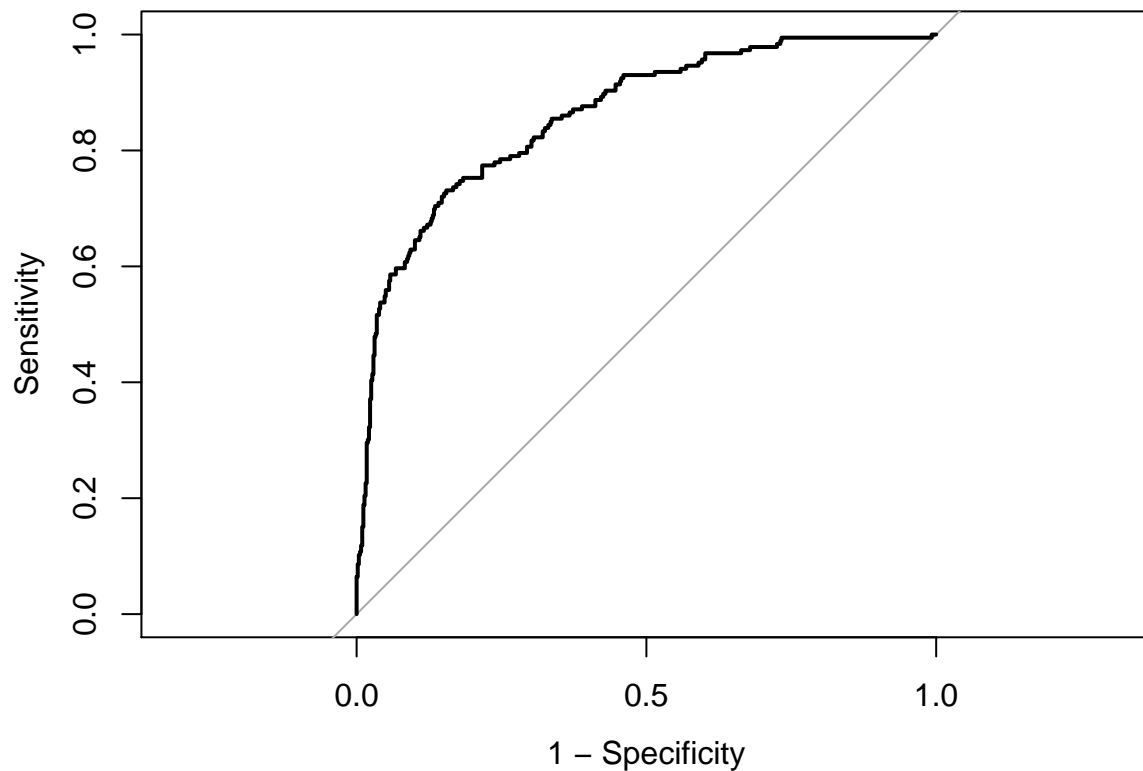
Produce the ROC curve using the test data

```
rocCurveLDA = roc(Ytest, YhatTestProb$Yes)
```

```
## Setting levels: control = No, case = Yes
```

```
## Setting direction: controls < cases
```

```
plot(rocCurveLDA, legacy.axes=TRUE)
```



Problem 2.3: AUC

Get the AUC for LDA

```
LDAauc = rocCurveLDA$auc  
LDAauc#### Answer 2.3.1
```

```
## Area under the curve: 0.8613
```

Problem 2.4: Achieving a particular sensitivity

What is the specificity for a model that has a sensitivity of at least 0.8? What threshold does it occur at?

```
thresholdsLDA = rocCurveLDA$thresholds  
  
pt8b          = which.min(rocCurveLDA$sensitivities >= 0.8)#Answer 1.7.1. Fill in the appropriate value)  
  
thresholdLDA  = thresholdsLDA[pt8b]  
specificityLDA = rocCurveLDA$specificities[pt8b]  
#### Answer 2.4.1.
```

The specificity is 0.7059961. It occurs at a threshold of 0.1984692

Problem 3: Comparison (35 pts)

For this problem, you can either adjust the previous object names so that the correct information is printed here (this would be a great idea) or (as I did in the solutions to make the code look as similar between logistic and lda as possible) you can just write in the relevant information. Report your answers to at least 4 digits after the decimal.

Answer 3.1: What are the models's accuracies? Which model would you prefer based on accuracy? The accuracy for Logistic Regression and LDA is 0.8477952 and 0.8364154. Higher the accuracy of the model, better the performance rate. In this case, Logistic regression has higher accuracy than LDA.

Answer 3.2: What are the models's AUCs? Which model would you prefer based on AUC? The AUC value for both Logistic Regression and LDA is 0.8689191 and 0.8612966. Higher the AUC better the performance of the model, logistic regression has higher AUC than LDA.

Answer 3.3: What are the models's maximum specificity for a sensitivity at least 0.8? Which model would you prefer based on this? The maximum specificity for a sensitivity at least 0.8 for logistic regression and LDA are 0.7330754 and 0.7059961. High value of specificity is preferred for better clasification. Hence, logistic regression is preferred over LDA.