Logit, LDA, QDA, and KNN

Langtao Chen

Feb 25, 2020

Contents

In this example, we'll use the Credit Card Default dataset to demonstrate how to conduct logistic regression, LDA, QDA, and kNN classification.

1. Data

```
# Load ISLR package
library(ISLR)

# Structure of the Default dataset
str(Default)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ default: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ student: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 2 1 1 ...
##  $ balance: num  730 817 1074 529 786 ...
##  $ income : num  44362 12106 31767 35704 38463 ...
```

The dataset contains 10,000 observations of 4 variables including:

- default: A factor with levels No and Yes indicating whether the customer defaulted on their debt
- student: A factor with levels No and Yes indicating whether the customer is a student
- balance: The average balance that the customer has remaining on their credit card after making their monthly payment
- income: Income of customer

```
head(Default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

```
summary(Default)
```

```
##  default    student       balance           income
##  No :9667   No :7056   Min.   :   0.0   Min.   :  772
##  Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
##                        Median : 823.6   Median :34553
##                        Mean   : 835.4   Mean   :33517
##                        3rd Qu.:1166.3   3rd Qu.:43808
##                        Max.   :2654.3   Max.   :73554
```

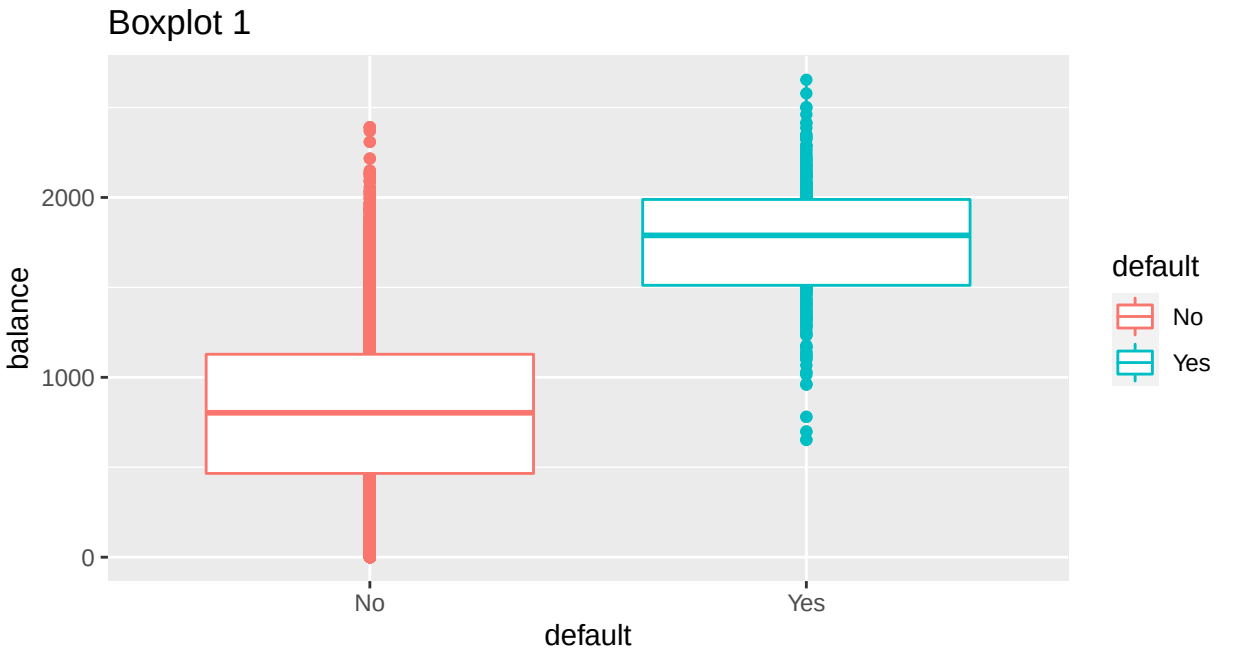
2. Data Exploration

```
# Frequency of default
table(Default$default)
```

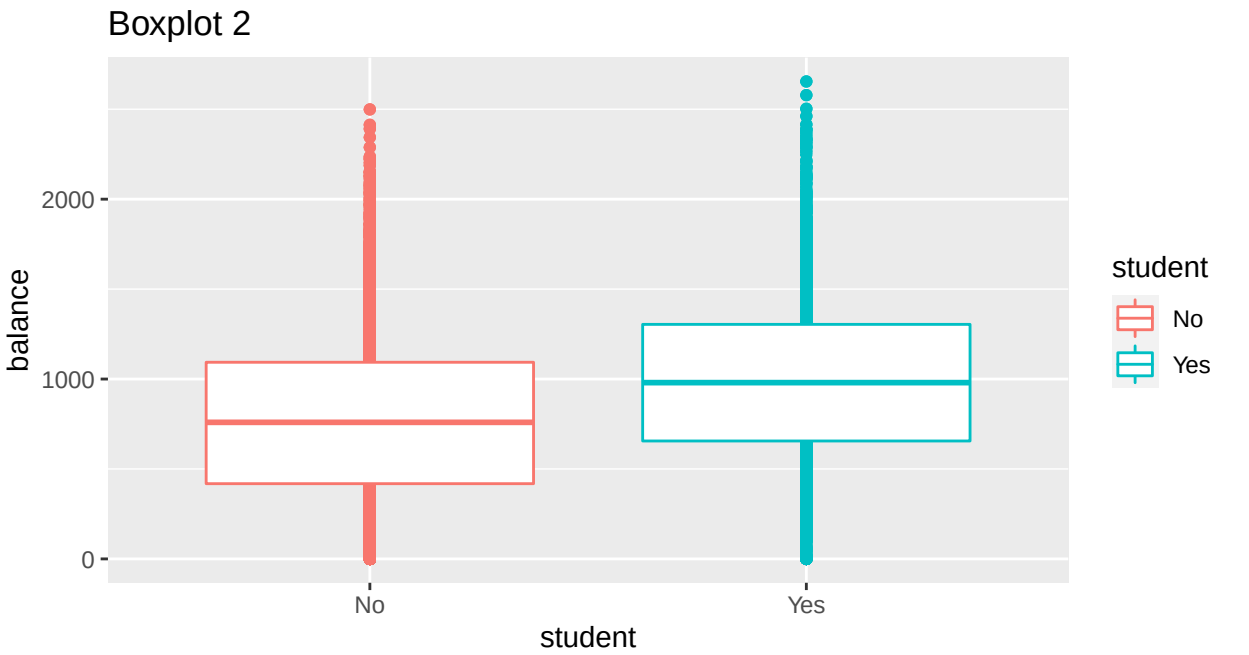
```
##
##   No  Yes
## 9667  333
```

Among the 10,000 customers, 333 customers defaulted their credit card debt.

```
library(ggplot2)
qplot(x=default, y=balance, data = Default, col= default, main = 'Boxplot 1') +
  geom_boxplot()
```




```
qplot(x=student, y=balance, data = Default, col= student, main = 'Boxplot 2') +
  geom_boxplot()
```

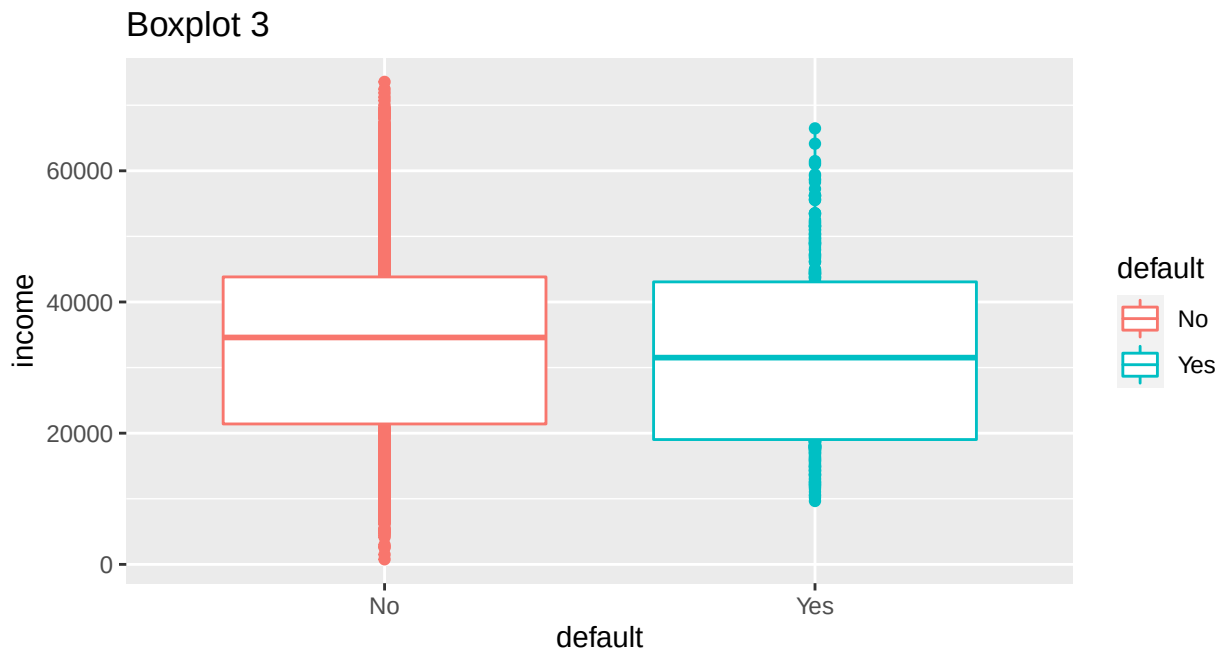


We also notice that student customers have higher balance than non-student customers.

From the above plot, we can notice that the distribution of balance is different across two customer groups

(default = Yes vs default = No). Customers tend to default their debt when the balance is higher.

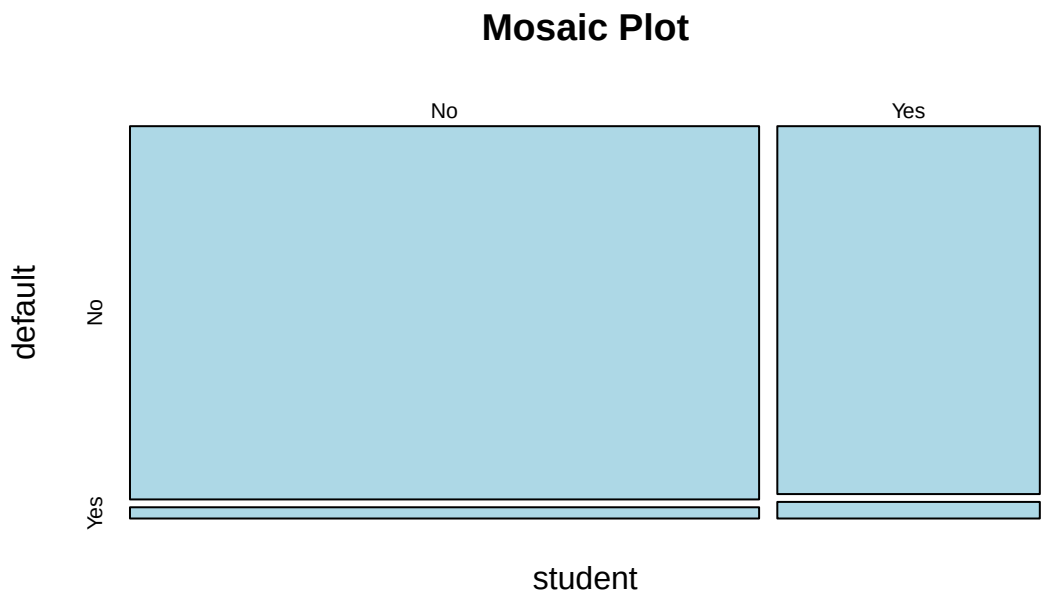
```
qplot(x=default, y=income, data = Default, col= default, main = 'Boxplot 3') +
  geom_boxplot()
```



From the above plot, it seems the income of the customer does not matter whether the customer defaults the debt or not.

As the student and default variables are both qualitative, we can use a mosaic plot to show their relationship.

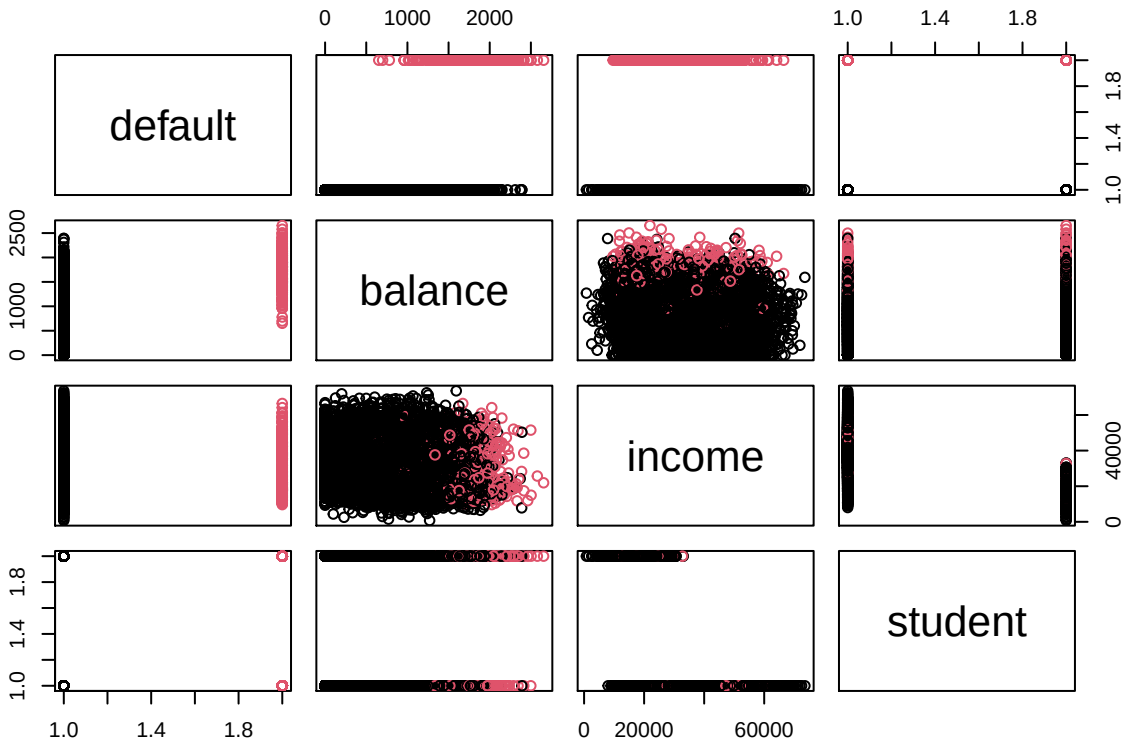
```
mosaicplot(~ student + default, data=Default, main = "Mosaic Plot", col='lightblue')
```



From the above plot, it seems student customers are more likely to default than non-student customers.

Draw a scatterplot matrix of the dataset.

```
pairs(~default + balance + income + student, data = Default, col = Default$default)
```



3. Logistic Regression

3.1. Fit a Logistic Regression Model

With the visualization of the relationships between predictors and the response, now let's formally use logistic regression model to analyze these relationships.

First, let's fit the logistic regression model by using only one predictor balance

```
# Logistic regression
logit1.fit <- glm(default ~ balance,
                  family=binomial(link='logit'),data = Default)

summary(logit1.fit)
```

```
##
## Call:
## glm(formula = default ~ balance, family = binomial(link = "logit"),
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2697  -0.1465  -0.0589  -0.0221   3.7589
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.065e+01  3.612e-01  -29.49   <2e-16 ***
```

```
## balance      5.499e-03  2.204e-04   24.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1596.5  on 9998  degrees of freedom
## AIC: 1600.5
##
## Number of Fisher Scoring iterations: 8
```

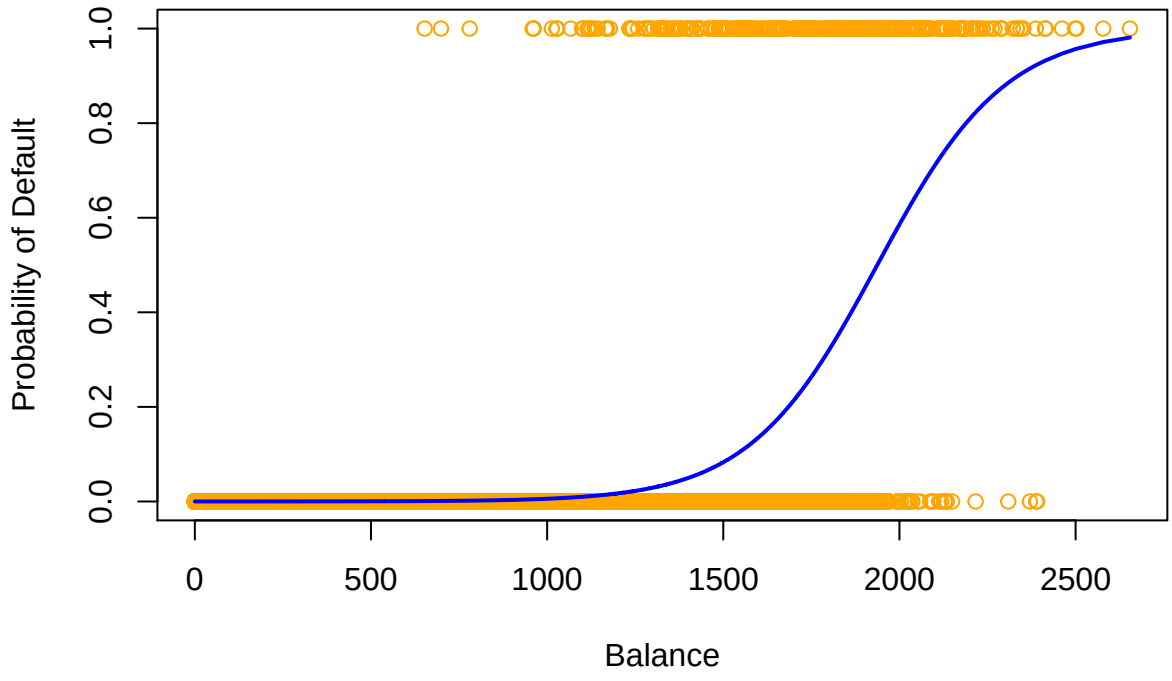
We can see that balance has a significant and positive effect on the probability (or log odds) of default. This is consistent with the conclusion of the box plot 1 above.

We can visualize the default probability predicted by the fitted logistic regression model as follows.

```r
# Calculate predicted probability
logit1.prob <- predict(logit1.fit, type = "response")

plot(x = Default$balanc, y = ifelse(Default$default == "Yes", 1, 0),
     col = "orange", xlab = "Balance", ylab = "Probability of Default")

points(x = Default$balance[order(Default$balance)],
       y = logit1.prob[order(Default$balance)],
       type = "l", col="blue", lwd = 2)
```



From the above plot, we find that if balance is high, the probability of default is high.

First, let's fit the logistic regression model by using only one predictor student.

```
# Logistic regression
logit2.fit <- glm(default ~ student,
                  family=binomial(link='logit'),data = Default)

summary(logit2.fit)
```

```
##
## Call:
## glm(formula = default ~ student, family = binomial(link = "logit"),
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.2970  -0.2970  -0.2434  -0.2434   2.6585
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.50413    0.07071  -49.55  < 2e-16 ***
## studentYes   0.40489    0.11502    3.52 0.000431 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 2908.7  on 9998  degrees of freedom
## AIC: 2912.7
##
## Number of Fisher Scoring iterations: 6
```

We find that being a student significantly increases the probability of default. This is consistent with the conclusion of the mosaic plot above.

Now, let's include balance into the modeling.

```
# Logistic regression
logit3.fit <- glm(default ~ balance + student,
                  family=binomial(link='logit'),data = Default)

summary(logit3.fit)
```

```
##
## Call:
## glm(formula = default ~ balance + student, family = binomial(link = "logit"),
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4578  -0.1422  -0.0559  -0.0203   3.7435
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.075e+01  3.692e-01 -29.116  < 2e-16 ***
## balance      5.738e-03  2.318e-04  24.750  < 2e-16 ***
## studentYes  -7.149e-01  1.475e-01  -4.846 1.26e-06 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1571.7  on 9997  degrees of freedom
## AIC: 1577.7
##
## Number of Fisher Scoring iterations: 8
```

We can find that being a student significantly decreases the probability of default, after controlling for balance.

Balance variable distorts the relationship between student and default. Balance is also called a confounder or confounding variable in this case.

Now, let's regress default on all other variables.

```
# Logistic regression
logit.fit <- glm(default ~ balance + income + student,
                 family=binomial(link='logit'),data = Default)

summary(logit.fit)
```

```
##
## Call:
## glm(formula = default ~ balance + income + student, family = binomial(link = "logit"),
##     data = Default)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4691  -0.1418  -0.0557  -0.0203   3.7383
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
## balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
## income       3.033e-06  8.203e-06   0.370  0.71152
## studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1571.5  on 9996  degrees of freedom
## AIC: 1579.5
##
## Number of Fisher Scoring iterations: 8
```

```
# Get coefficient estimates
coef(logit.fit)
```

```
##   (Intercept)       balance        income    studentYes
## -1.086905e+01  5.736505e-03  3.033450e-06 -6.467758e-01
```

```
# Get coefficient estimates with statistics
summary(logit.fit)$coef
```

```
##                  Estimate   Std. Error    z value      Pr(>|z|)
## (Intercept) -1.086905e+01 4.922555e-01 -22.080088 4.911280e-108
## balance      5.736505e-03 2.318945e-04  24.737563 4.219578e-135
## income       3.033450e-06 8.202615e-06   0.369815  7.115203e-01
## studentYes  -6.467758e-01 2.362525e-01  -2.737646  6.188063e-03
```

```
# Get p-values for all coefficient estimates
summary(logit.fit)$coef[,4]
```

```
##   (Intercept)       balance        income    studentYes
## 4.911280e-108 4.219578e-135  7.115203e-01  6.188063e-03
```

3.2. Interpret Logistic Regression

Let's use the stargazer package to report regression results in a more professional way.

```
library(stargazer)
```

```
stargazer(logit1.fit, logit2.fit, logit3.fit, logit.fit,
          type = "text",star.cutoffs = c(0.05, 0.01, 0.001),
          title="Logistic Regression", digits=4)
```

```
##
## Logistic Regression
## ===============================================================
##                                 Dependent variable:
##                     ------------------------------------------
##                                       default
##                         (1)         (2)         (3)         (4)
## ---------------------------------------------------------------
## balance             0.0055***               0.0057***   0.0057***
##                     (0.0002)                (0.0002)    (0.0002)
##
## income                                                  0.000003
##                                                         (0.00001)
##
## studentYes                      0.4049***  -0.7149***   -0.6468**
##                                 (0.1150)    (0.1475)    (0.2363)
##
## Constant           -10.6513*** -3.5041***  -10.7495*** -10.8690***
##                     (0.3612)    (0.0707)    (0.3692)    (0.4923)
##
## ---------------------------------------------------------------
## Observations         10,000      10,000      10,000      10,000
## Log Likelihood     -798.2258  -1,454.3410  -785.8408   -785.7724
## Akaike Inf. Crit. 1,600.4520  2,912.6830  1,577.6820  1,579.5450
## ===============================================================
## Note:                               *p<0.05; **p<0.01; ***p<0.001
```

Comparing the simple logistic regression (including only one predictor student) and the full model (including all predictors), we can find that there is a confounding due to the high correlation between student and balance.

Interpretation of the full model of logistic regression is:

- Balance has a positive and significant effect on default (p-value < 0.001). A unit increase in balance increases the log odds by 0.0057 after controlling for other factors.
- Income does not have a statistically significant on default.
- Being a student has a negative and significant effect on default (p-value < 0.001). Being student reduces the log odds by 0.6468 after controlling for other factors.

Note: The effect of student on default is different from what we found from the mosaic plot. This is called confouding due to correlation among predictors.

3.3. Calculate Pseudo R Squared

We notice that logistic regression does not report R squared. We can calculate the McFadden pseudo R squared by sung the pscl package.

```
# McFadden R2
# install.packages("pscl")
library(pscl)
pR2(logit.fit)
```

```
## fitting null model for pseudo-r2
```

```
##           llh       llhNull            G2      McFadden          r2ML
##  -785.7724138 -1460.3248557  1349.1048838     0.4619194     0.1262059
##          r2CU
##     0.4982860
```

Or you can manually calculate the pseudo R squared by using the following formula:

$R^2_{McFadden} = 1 - \frac{log(L_m)}{log(L_{null})}$

where $log(L_m)$ is the log likelihood of the model of interest, $log(L_{null})$ is the likelihood of the null model, that has only intercept without any independent variables.

```
# Fit the null model
logit.null.fit <- glm(default ~1,family=binomial(link='logit'),data=Default)

# Show the log likelihood of the null model
logLik(logit.null.fit)
```

```
## 'log Lik.' -1460.325 (df=1)
```

```
# Manually calculate McFadden pseudo R squared
cat("McFadden pseudo R2 = ", 1-logLik(logit.fit)/logLik(logit.null.fit))
```

```
## McFadden pseudo R2 =  0.4619194
```

3.4. Use Logistic Regression Model to Predict

3.4.1. In-Sample Prediction

We can use the predict() function to calculate the predicted outcome. In the following code, the parameter type = "response" is specified to calculate the predicted possibility. By default, the prediction of a logit model is the log odds.

```
# Calculate probability of default
logit.probs <- predict(logit.fit,type="response")

# Show the first 10 values
logit.probs[1:10]
```

```
##            1            2            3            4            5            6
## 1.428724e-03 1.122204e-03 9.812272e-03 4.415893e-04 1.935506e-03 1.989518e-03
##            7            8            9           10
## 2.333767e-03 1.086718e-03 1.638333e-02 2.080617e-05
```

```
# Calculate predicted default
logit.pred <- ifelse(logit.probs >.5, "Yes", "No")

# Show confusion matrix
table(Prediction=logit.pred, Truth=Default$default)
```

```
##           Truth
## Prediction   No  Yes
##        No  9627  228
##        Yes   40  105
```

```
# Calculate in-sample prediction accuracy
mean(logit.pred == Default$default)
```

```
## [1] 0.9732
```

Another way to report the prediction performance is to use the confusionMatrix function in caret package. This function provides other important measures such as kappa coefficient, sensitivity, and specificity.

```
library(caret)
```

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

```
confusionMatrix(factor(logit.pred),Default$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  9627  228
##        Yes   40  105
##
##                Accuracy : 0.9732
##                  95% CI : (0.9698, 0.9763)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.0001044
##
##                   Kappa : 0.4278
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.3153
##             Specificity : 0.9959
##          Pos Pred Value : 0.7241
##          Neg Pred Value : 0.9769
##              Prevalence : 0.0333
##          Detection Rate : 0.0105
##    Detection Prevalence : 0.0145
##       Balanced Accuracy : 0.6556
##
##        'Positive' Class : Yes
##
```

The in-sample prediction accuracy tends to overestimate the true accuracy. Let's calculate out-of-sample prediction performance.

We can call the sensitivity() and specificity() function in caret package to directly calculate sensitivity and specificity.

```
sensitivity(factor(logit.pred),Default$default, positive = "Yes")
```

```
## [1] 0.3153153
```

```
specificity(factor(logit.pred),Default$default, negative = "No")
```

```
## [1] 0.9958622
```

3.4.2. Out-of-Sample Prediction

To calculate out-of-sampel prediction performance, we need to split the dataset into training set and testing set. Use the training data to fit the logistic regression model. Use the testing data to test the performance of the model.

We use a single 70/30% split. We can use the sample() method supported by base R system to do the random sampling.

```
# Data partition: randomly split the dataset into a train (70%) and a test set (30%)
index <- 1:nrow(Default)
set.seed(123)
train_index <- sample(index, round(length(index)*0.7))
train_set <- Default[train_index,]
test_set <- Default[-train_index,]
```

```
# Train the logistic regression model
logit.fit.train <- glm(default ~ balance + income + student,
                   family=binomial(link='logit'),data = train_set)

summary(logit.fit.train)
```

```
##
## Call:
## glm(formula = default ~ balance + income + student, family = binomial(link = "logit"),
##     data = train_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1583  -0.1409  -0.0567  -0.0205   3.7368
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.109e+01  5.923e-01 -18.723   <2e-16 ***
## balance      5.760e-03  2.772e-04  20.777   <2e-16 ***
## income       7.592e-06  9.799e-06   0.775   0.4385
## studentYes  -5.222e-01  2.828e-01  -1.847   0.0648 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2043.8  on 6999  degrees of freedom
## Residual deviance: 1094.9  on 6996  degrees of freedom
```

```
## AIC: 1102.9
##
## Number of Fisher Scoring iterations: 8
```

Compare the full model and the model trained using the training dataset.

```
stargazer(logit.fit, logit.fit.train, type = "text",star.cutoffs = c(0.05, 0.01, 0.001),
          title="Logistic Regression", digits=4)
```

```
##
## Logistic Regression
## ================================================
##                         Dependent variable:
##                     ---------------------------
##                               default
##                          (1)            (2)
## ------------------------------------------------
## balance                0.0057***      0.0058***
##                        (0.0002)        (0.0003)
##
## income                 0.000003        0.00001
##                        (0.00001)      (0.00001)
##
## studentYes             -0.6468**       -0.5222
##                        (0.2363)        (0.2828)
##
## Constant              -10.8690***    -11.0890***
##                        (0.4923)        (0.5923)
##
## ------------------------------------------------
## Observations            10,000          7,000
## Log Likelihood         -785.7724       -547.4288
## Akaike Inf. Crit.    1,579.5450      1,102.8580
## ================================================
## Note:                *p<0.05; **p<0.01; ***p<0.001
```

```
# Calculate probability of default
test_probs <- predict(logit.fit.train, newdata = test_set, type="response")

# Show the first 10 values
test_probs[1:10]
```

```
##            6           19           22           23           24           29
## 0.0019123268 0.0004004065 0.0047440436 0.0097839448 0.0007765207 0.0007141964
##           33           34           35           37
## 0.0003121061 0.0041906672 0.0377764368 0.0011812639
```

```
# Calculate predicted default
test_pred <- ifelse(test_probs >.5, "Yes", "No")

# Show confusion matrix
confusionMatrix(factor(test_pred),test_set$default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
```

```
##        No  2889   70
##        Yes   11   30
##
##               Accuracy : 0.973
##                 95% CI : (0.9666, 0.9785)
##    No Information Rate : 0.9667
##    P-Value [Acc > NIR] : 0.02705
##
##                  Kappa : 0.4142
##
##  Mcnemar's Test P-Value : 1.16e-10
##
##             Sensitivity : 0.9962
##             Specificity : 0.3000
##          Pos Pred Value : 0.9763
##          Neg Pred Value : 0.7317
##              Prevalence : 0.9667
##          Detection Rate : 0.9630
##    Detection Prevalence : 0.9863
##       Balanced Accuracy : 0.6481
##
##        'Positive' Class : No
##
```

Among all 2913 non-default customers, 2898 customers are correctly predicted as non-default by the model. However, the prediction of default customer is not that good: only 22 out of 87 default customers are correctly predicted by the algorithm.

In the next, let's explore other algorithms including LDA and KNN.

4. Linear Discriminant Analysis (LDA)

Conduct a linear discriminant analysis on the whole set of Default data.

```
library(MASS)
lda.fit=lda(default ~ balance + income + student, data = Default)
lda.fit
```

```
## Call:
## lda(default ~ balance + income + student, data = Default)
##
## Prior probabilities of groups:
##     No    Yes
## 0.9667 0.0333
##
## Group means:
##       balance   income studentYes
## No   803.9438 33566.17  0.2914037
## Yes 1747.8217 32089.15  0.3813814
##
## Coefficients of linear discriminants:
##                      LD1
## balance     2.243541e-03
## income      3.367310e-06
## studentYes -1.746631e-01
```

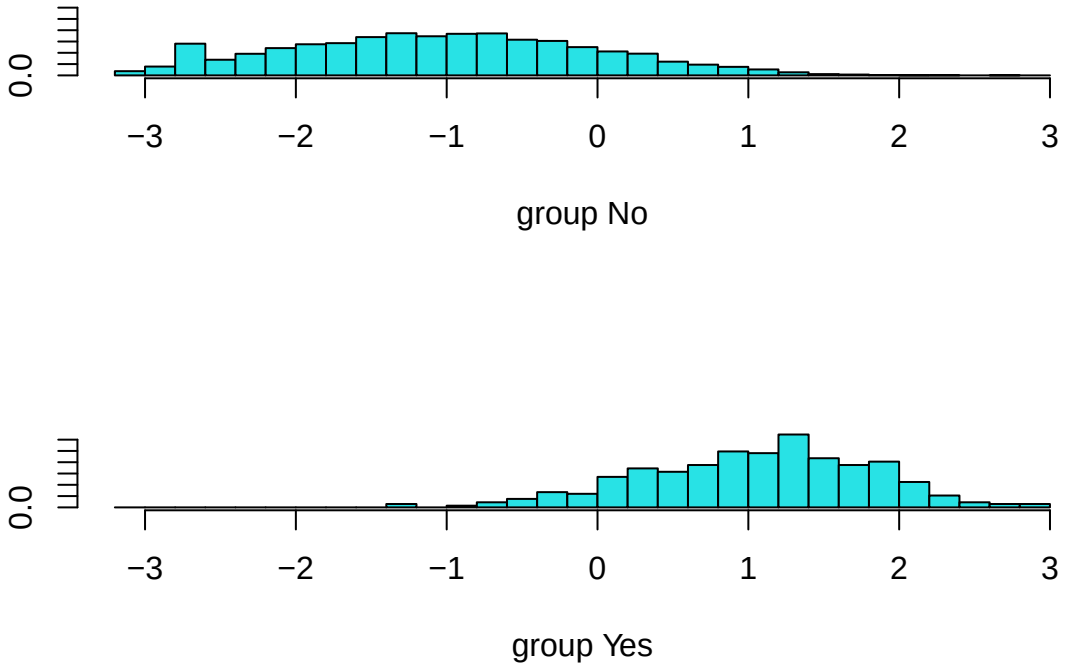
In this example, we got one linear discriminant (LD1).

```
# Show coefficients of linear discriminants
lda.fit$scaling
```

```
##                      LD1
## balance     2.243541e-03
## income      3.367310e-06
## studentYes -1.746631e-01
```

The following plot shows how the response is classified by the LDA classifier. The X-axis is the value of line defined by the coefficients of linear discriminants.

```
plot(lda.fit)
```



Calculate the in-sample prediction.

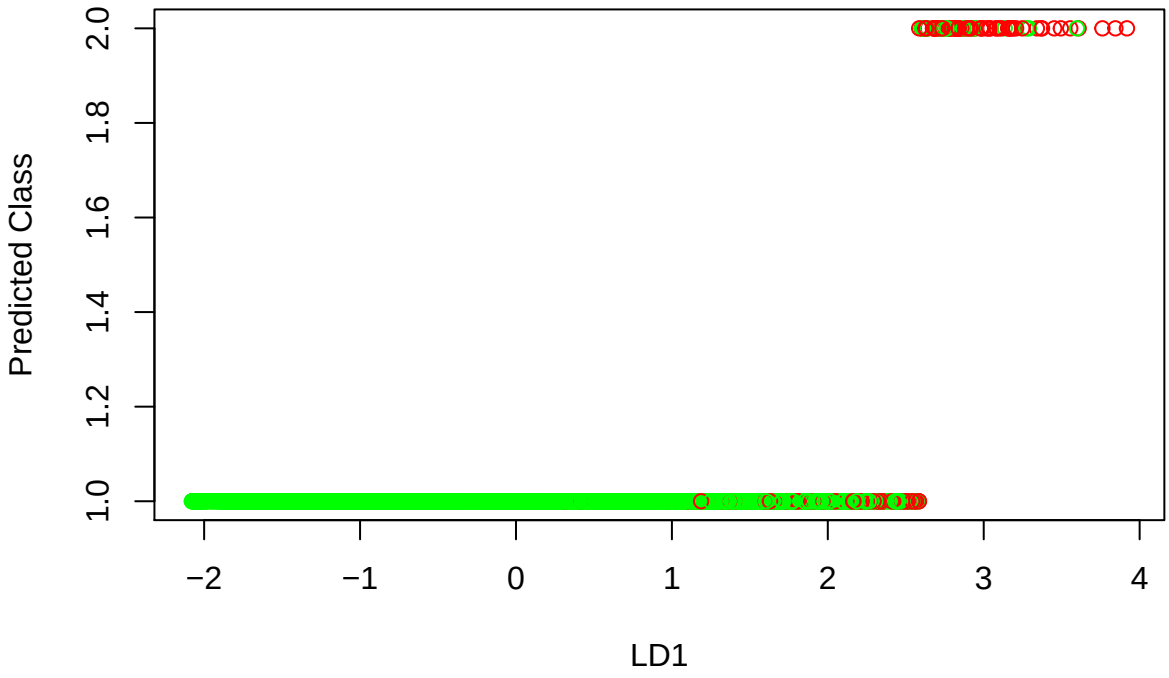
```
lda.pred=predict(lda.fit)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

The following figure shows how the data are classified. Class “Yes” and “No” are colored as red and green correspondingly.

The green points in class 2 (“Yes”) and red points in class 1 (“No”) represent the misclassified response.

```
plot(lda.pred$x, lda.pred$class,
     col=c("green","red")[Default$default],
     xlab = 'LD1', ylab = 'Predicted Class')
```




```
confusionMatrix(lda.pred$class,Default$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  9645  254
##        Yes   22   79
##
##                Accuracy : 0.9724
##                  95% CI : (0.969, 0.9755)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.0006128
##
##                   Kappa : 0.354
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.2372
##             Specificity : 0.9977
##          Pos Pred Value : 0.7822
##          Neg Pred Value : 0.9743
##              Prevalence : 0.0333
##          Detection Rate : 0.0079
##    Detection Prevalence : 0.0101
##       Balanced Accuracy : 0.6175
```

```
##
##        'Positive' Class : Yes
##
```

```
# Calculate in-sample prediction accuracy
mean(lda.pred$class == Default$default)
```

```
## [1] 0.9724
```

```
sum(lda.pred$posterior[,2]>=.5)
```

```
## [1] 101
```

```
sum(lda.pred$posterior[,2]<.5)
```

```
## [1] 9899
```

```
lda.pred$posterior[1:20,2]
```

```
##            1            2            3            4            5            6
## 3.223517e-03 2.689531e-03 1.470860e-02 1.184329e-03 4.023242e-03 4.208244e-03
##            7            8            9           10           11           12
## 4.408251e-03 2.686170e-03 2.292005e-02 9.394683e-05 6.181968e-05 1.781659e-02
##           13           14           15           16           17           18
## 2.844354e-04 1.819757e-03 1.670438e-02 4.035712e-04 1.089540e-04 7.173153e-04
##           19           20
## 1.159558e-03 1.566269e-02
```

```
lda.pred$class[1:20]
```

```
##  [1] No No No No No No No No No No No No No No No No No No No No
## Levels: No Yes
```

If we use 0.2 as threshold, the number of default prediction can be calculated as:

```
sum(lda.pred$posterior[,2]>.2)
```

```
## [1] 425
```

Redo the prediction by using 0.2 as threshold.

```
lda.pred2 <- ifelse(lda.pred$posterior[,2] > 0.2, "Yes", "No")
table(lda.pred2)
```

```
## lda.pred2
##   No  Yes
## 9575  425
```

```
confusionMatrix(factor(lda.pred2),Default$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  9435  140
##        Yes  232  193
##
##                Accuracy : 0.9628
##                  95% CI : (0.9589, 0.9664)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.985
```

```
##
##                   Kappa : 0.4902
##
##  Mcnemar's Test P-Value : 2.38e-06
##
##             Sensitivity : 0.5796
##             Specificity : 0.9760
##          Pos Pred Value : 0.4541
##          Neg Pred Value : 0.9854
##              Prevalence : 0.0333
##          Detection Rate : 0.0193
##    Detection Prevalence : 0.0425
##       Balanced Accuracy : 0.7778
##
##        'Positive' Class : Yes
##
```

Compared with the default threshold 0.5, using 0.2 as the threshold increases sensitivity but reduces specificity.

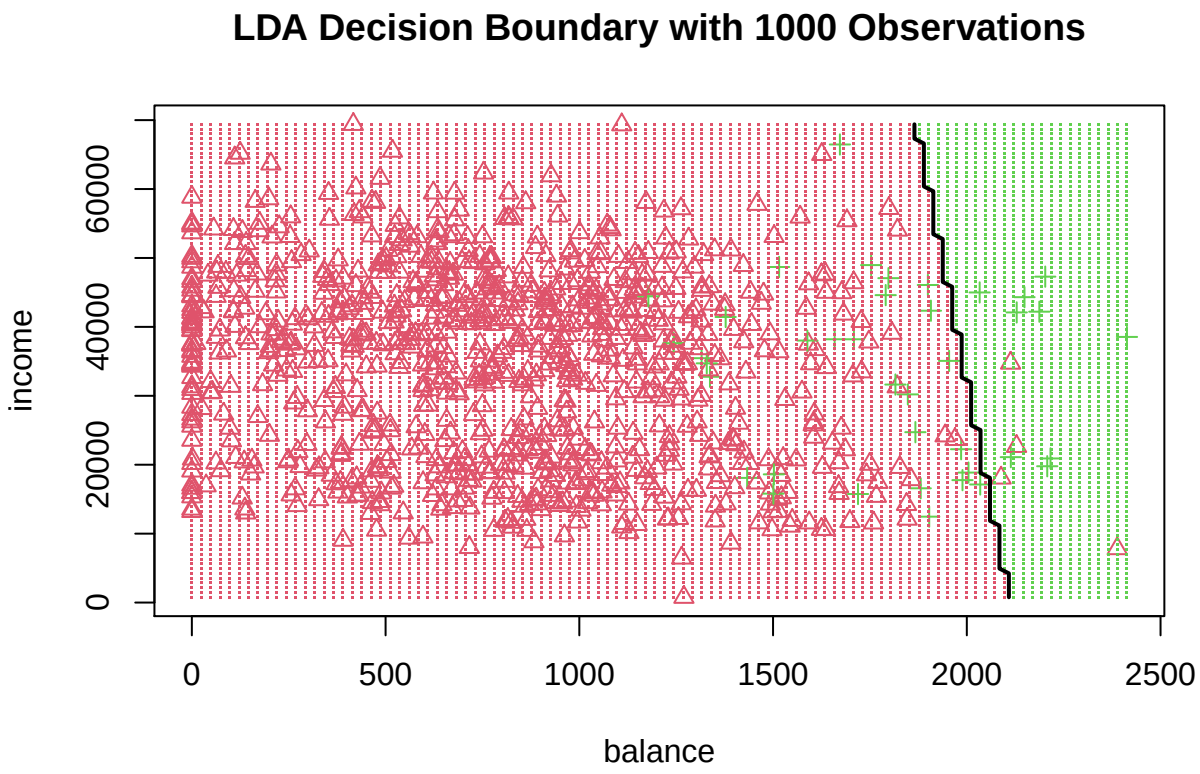
In order to further increase sensitivity (or reduce the false negative rate), we may want to reduce the threshold to 0.1 or less.

We can also visualize the LDA decision boundary as follows, ignoring if a customer is a student or not.

```
x <- Default[c("balance", "income", "default")]

lda.fit2=lda(default ~ balance + income, data = x)

set.seed(3)
decisionplot(lda.fit2, x[sample(nrow(x), 1000),], class = "default",
             main = "LDA Decision Boundary with 1000 Observations")
```



5. Quadratic Discriminant Analysis (QDA)

5.1. Fit QDA on the Training Dataset

```
qda.fit=qda(default ~ balance + income + student, data = train_set)
qda.fit
```

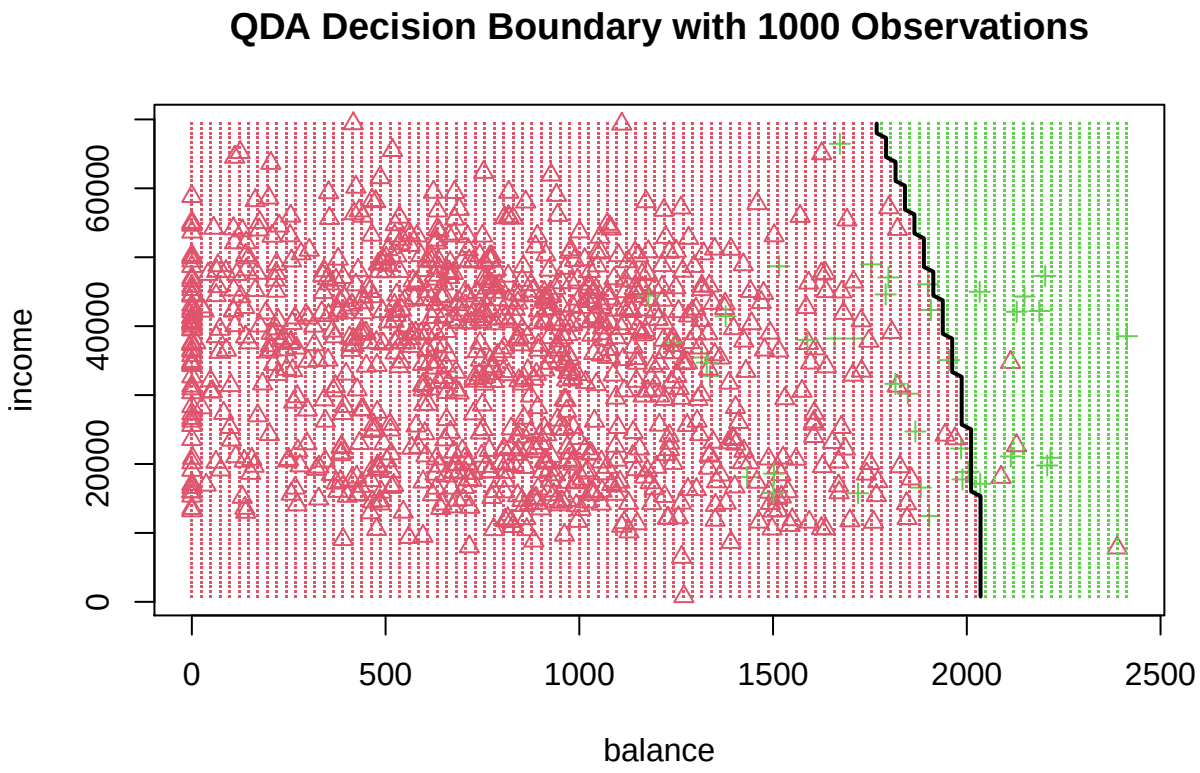
```
## Call:
## qda(default ~ balance + income + student, data = train_set)
##
## Prior probabilities of groups:
##         No        Yes
## 0.96671429 0.03328571
##
## Group means:
##       balance   income studentYes
## No   806.0212 33561.71  0.2883109
## Yes 1747.6158 32574.86  0.3733906
```

We can also visualize the QDA decision boundary as follows, ignoring if a customer is a student or not. Compared to the LDA, the QDA has a non-linear decision boundary.

```
x <- Default[c("balance", "income", "default")]

qda.fit2=qda(default ~ balance + income, data = x)
```

```
set.seed(3)
decisionplot(qda.fit2, x[sample(nrow(x), 1000),], class = "default",
             main = "QDA Decision Boundary with 1000 Observations")
```



5.2. Evaluate QDA on Test Dataset

```
qda.pred=predict(qda.fit,test_set)

confusionMatrix(qda.pred$class, test_set$default, positive = "Yes")
```

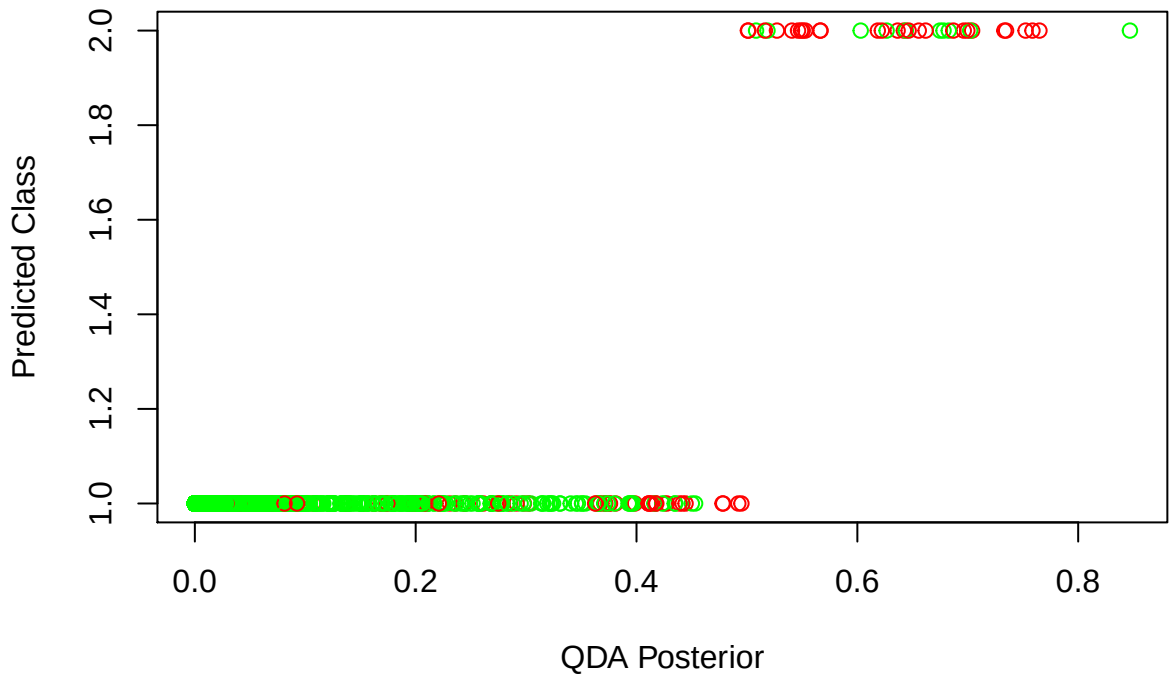
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2890   73
##        Yes   10   27
##
##                Accuracy : 0.9723
##                  95% CI : (0.9658, 0.9779)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.04365
##
##                   Kappa : 0.3831
##
##  Mcnemar's Test P-Value : 1.008e-11
```

```
##
##             Sensitivity : 0.27000
##             Specificity : 0.99655
##          Pos Pred Value : 0.72973
##          Neg Pred Value : 0.97536
##              Prevalence : 0.03333
##          Detection Rate : 0.00900
##    Detection Prevalence : 0.01233
##       Balanced Accuracy : 0.63328
##
##        'Positive' Class : Yes
##
```

The following figure shows how the test data are classified by the QDA model. Class "Yes" and "No" are colored as red and green correspondingly.

The green points in class 2 ("Yes") and red points in class 1 ("No") represent the misclassified response in the test dataset.

```
plot(qda.pred$posterior[,2], qda.pred$class,
     col=c("green","red")[test_set$default],
     xlab = 'QDA Posterior', ylab = 'Predicted Class')
```



6. K-Nearest Neighbors

6.1. Fit k-NN Models on the Training Dataset

We use the knn() method in "class" package to fit the k-NN model.

Because the k-NN algorithm needs to calculate distance between observations, all predictors need to be represented as numbers. Let's dummy code the student variable.

```
train_set$student <- ifelse(train_set$student=='Yes', 1, 0)
test_set$student <- ifelse(test_set$student=='Yes', 1, 0)
```

Then, fit the k-NN model to the training dataset.

```
library(class)
# Select the true values of the response in training set
cl <- train_set[,"default"]
# Use knn for k = 5, 20
knn5 <- knn(train_set[,-1],test_set[,-1], cl, k = 5)
knn20 <- knn(train_set[,-1],test_set[,-1], cl, k = 20)
```

6.2. Evaluate k-NN Models on Test Dataset

```
# Confusion matrix and statistics, k = 5
confusionMatrix(knn5,test_set$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2885   84
##        Yes   15   16
##
##                Accuracy : 0.967
##                  95% CI : (0.96, 0.9731)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.486
##
##                   Kappa : 0.2322
##
##  Mcnemar's Test P-Value : 8.243e-12
##
##             Sensitivity : 0.160000
##             Specificity : 0.994828
##          Pos Pred Value : 0.516129
##          Neg Pred Value : 0.971708
##              Prevalence : 0.033333
##          Detection Rate : 0.005333
##    Detection Prevalence : 0.010333
##       Balanced Accuracy : 0.577414
##
##        'Positive' Class : Yes
##
```

```
# Confusion matrix and statistics, k = 20
confusionMatrix(knn20,test_set$default, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2899   98
##        Yes    1    2
##
##                Accuracy : 0.967
##                  95% CI : (0.96, 0.9731)
##     No Information Rate : 0.9667
##     P-Value [Acc > NIR] : 0.486
##
##                   Kappa : 0.037
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0200000
##             Specificity : 0.9996552
##          Pos Pred Value : 0.6666667
##          Neg Pred Value : 0.9673006
##              Prevalence : 0.0333333
##          Detection Rate : 0.0006667
##    Detection Prevalence : 0.0010000
##       Balanced Accuracy : 0.5098276
##
##        'Positive' Class : Yes
##
```

Tune the hyperparameter k for KNN.

```
accuracy <- NULL
sensitivity <-NULL
specificity <- NULL

for(i in 1:50) {
  knn.fit <- knn(train_set[,-1],test_set[,-1], cl, k = i)
  accuracy <- c(accuracy, mean(knn.fit == test_set$default))
  sensitivity <- c(sensitivity, sensitivity(knn.fit,test_set$default, positive = "Yes"))
  specificity <- c(specificity, specificity(knn.fit,test_set$default, negative = "No"))
}

balanced_accuracy = (sensitivity + specificity)/2
```

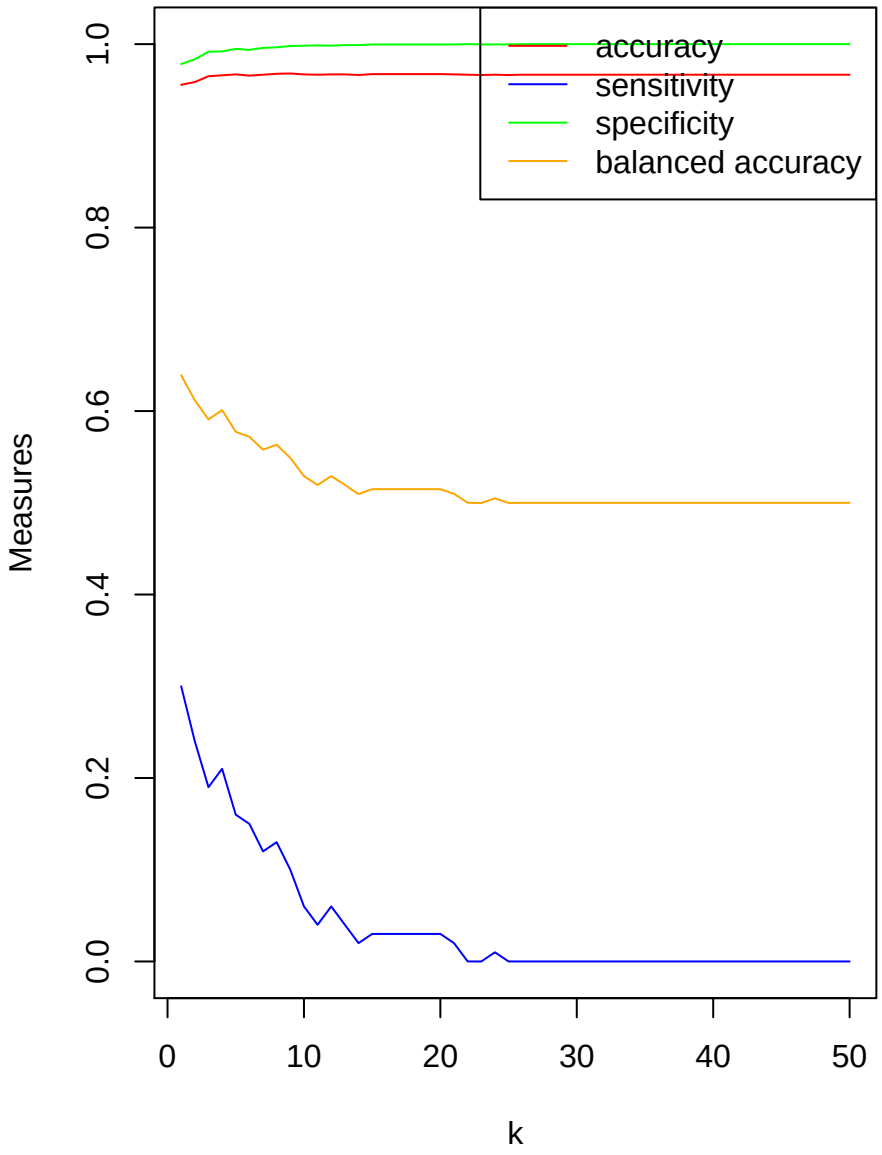
```
plot(1:50, accuracy, type = "l" ,col = "red",
     ylab = "Measures", xlab = "k",ylim = c(0.0, 1.0))

lines(1:50, sensitivity, type = "l", col = "blue")

lines(1:50, specificity, type = "l", col = "green")

lines(1:50, balanced_accuracy, type = "l", col = "orange")

legend("topright", legend = c("accuracy","sensitivity","specificity", "balanced accuracy"),
       col = c("red","blue","green","orange"), lty = 1)
```



If balanced accuracy is used to evaluate the performance, k = 1 gives the best balanced accuracy (0.639).