Week 4 - Homework

Alessio Benedetti

11 june 2018

Question 9.2

Using the same crime data set uscrime.txt as in Question 8.2, apply Principal Component Analysis and then create a regression model using the first few principal components. Specify your new model in terms of the original variables (not the principal components), and compare its quality to that of your solution to Question 8.2. You can use the R function prcomp for PCA.

First we need to load the libraries and the data from the temp *txt* file.

```
raw_data <- read.table('9.1uscrimeSummer2018.txt', stringsAsFactors = FALSE,
header=TRUE)
head(raw_data) #view top  rows of dataset

##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6
##       Prob    Time Crime
## 1 0.084602 26.2011   791
## 2 0.029599 25.2999  1635
## 3 0.083401 24.3006   578
## 4 0.015801 29.9012  1969
## 5 0.041399 21.2998  1234
## 6 0.034201 20.9995   682
```

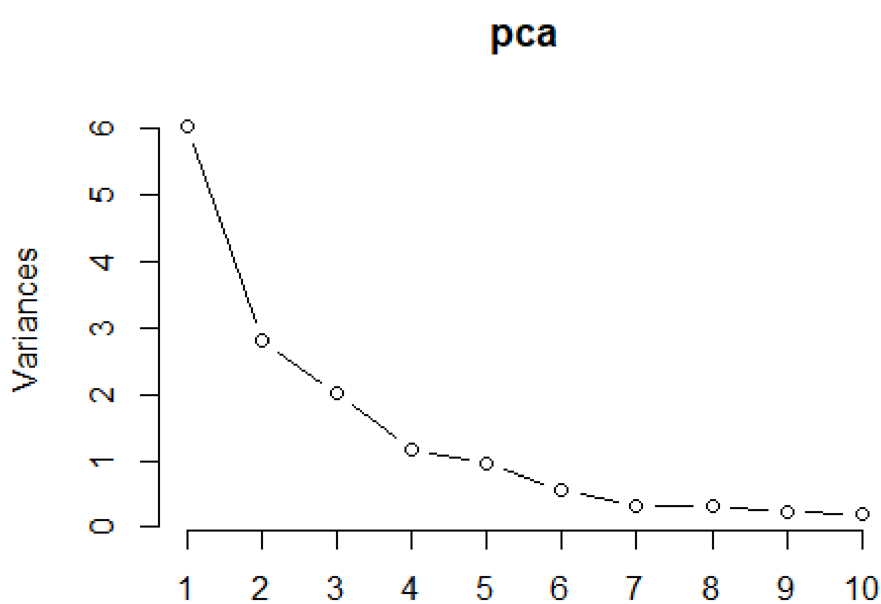
As suggested in the headline, we can use the `prcomp` function to perform PCA with scaled values.

```
pca <- prcomp(raw_data[,1:15], scale=TRUE)
summary(pca)

## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893 0.74377
## Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389 0.03688
## Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308 0.89996
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.56729 0.55444 0.48493 0.44708 0.41915 0.35804
```

```
## Proportion of Variance 0.02145 0.02049 0.01568 0.01333 0.01171 0.00855
## Cumulative Proportion  0.92142 0.94191 0.95759 0.97091 0.98263 0.99117
##                           PC13    PC14     PC15
## Standard deviation     0.26333 0.2418 0.06793
## Proportion of Variance 0.00462 0.0039 0.00031
## Cumulative Proportion  0.99579 0.9997 1.00000
```

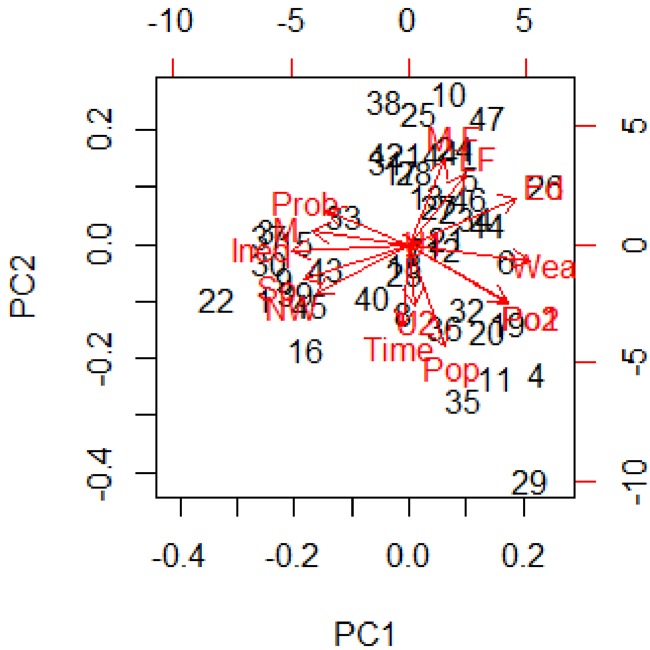
The `summary` method returns the standard deviation of each of the principal components and their rotation. With the `plot` statement we're able to see the variances as a function of the principal components.

```
plot(pca, type = 'l')
```



We can see in the figure that the first 6 principal components explains around 90% of the variance of the data. We also can represent the biplot as shown below:

```
biplot(pca)
```



Now we can build the regression model by using the first 6 principal components. First we build a new dataframe with the 6 principal components and the response variable: Crime.

```
pca_df <- data.frame(cbind(pca$x[,1:6],raw_data$Crime))
names(pca_df) <- c('PC1','PC2','PC3','PC4','PC5','PC6','Crime')
```

Then we fit the regression model:

```
model_pca <- lm(Crime ~ ., pca_df)
summary(model_pca)

##
## Call:
## lm(formula = Crime ~ ., data = pca_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -377.15 -172.23   25.81  132.10  480.38
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      35.35  25.604  < 2e-16 ***
## PC1            65.22      14.56   4.478 6.14e-05 ***
## PC2           -70.08      21.35  -3.283  0.00214 **
## PC3            25.19      25.23   0.998  0.32409
## PC4            69.45      33.14   2.095  0.04252 *
## PC5          -229.04      36.50  -6.275 1.94e-07 ***
```

```
## PC6           -60.21      48.04  -1.253  0.21734
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 242.3 on 40 degrees of freedom
## Multiple R-squared:  0.6586, Adjusted R-squared:  0.6074
## F-statistic: 12.86 on 6 and 40 DF,  p-value: 4.869e-08
```

The R-squared value is about 65%. We should now convert the principal components of the model, back to the original factors by using the rotation matrix.

```
convert_coeff <- (pca$rotation[,1:6] %*%
model_pca$coefficients[2:7])/pca$scale
adjusted_intercept <- model_pca$coefficients[1] - sum(convert_coeff *
pca$center)
```

We the "rotated" model we can now predict the Crime response with the new model and compare it with the value obtained in exercise 8.2

```
new_datapoint <- data.frame(M = 14.0, So = 0, Ed = 10.0, Po1 = 12.0, Po2 =
15.5,
                          LF = 0.640, M.F = 94.0, Pop = 150, NW = 1.1, U1 = 0.120,
                          U2 = 3.6, Wealth = 3200, Ineq = 20.1, Prob = 0.04, Time
= 39.0)

Crime <- sum(
  convert_coeff[1,1] %*% new_datapoint$M,
  convert_coeff[2,1] %*% new_datapoint$So,
  convert_coeff[3,1] %*% new_datapoint$Ed,
  convert_coeff[4,1] %*% new_datapoint$Po1,
  convert_coeff[5,1] %*% new_datapoint$Po2,
  convert_coeff[6,1] %*% new_datapoint$LF,
  convert_coeff[7,1] %*% new_datapoint$M.F,
  convert_coeff[8,1] %*% new_datapoint$Pop,
  convert_coeff[9,1] %*% new_datapoint$NW,
  convert_coeff[10,1] %*% new_datapoint$U1,
  convert_coeff[11,1] %*% new_datapoint$U2,
  convert_coeff[12,1] %*% new_datapoint$Wealth,
  convert_coeff[13,1] %*% new_datapoint$Ineq,
  convert_coeff[14,1] %*% new_datapoint$Prob,
  convert_coeff[15,1] %*% new_datapoint$Time,
  adjusted_intercept
  )

Crime

## [1] 1248.427
```

The Crime value for the new data point is 1248.427 with an R squared value of 65%.

In the previous exercise the regression model predicted a value of 1301.432 with an R squared value of 73%.

The PCA regression model performed a bit worse than the other.

Question 10.1

Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using a regression tree model, and a random forest model.In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results

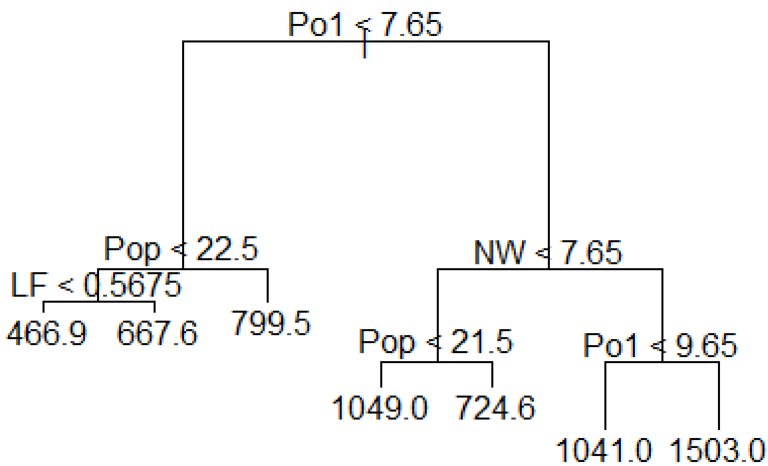
The data is already loaded from the previous question, with only need to load the appropriate libraries to conduct our analysis.

```
#install.packages('tree')
library(tree)
#install.packages('randomForest')
library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.
```

Tree

Let's first create a tree model.

```
model_tree <- tree(Crime ~ ., raw_data)
summary(model_tree)

##
## Regression tree:
## tree(formula = Crime ~ ., data = raw_data)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100

# Visualize tree model
plot(model_tree)
text(model_tree)
```



We can now calculate the R squared value (coefficient of determination) of the model. Let's build our own R squared function at first.

```
R2_calc <- function(yhat, raw_data) {
  SSres <- sum((yhat - raw_data$Crime)^2)
  SStot <- sum((raw_data$Crime - mean(raw_data$Crime))^2)
  R2 <- 1 - SSres/SStot
  return(R2)
}
```

Now we can calculate the coefficent of determination of the model:

```
crime_tree_yhat <- predict(model_tree)
crime_tree_r2 <- R2_calc(crime_tree_yhat, raw_data)
crime_tree_r2

## [1] 0.7244962
```

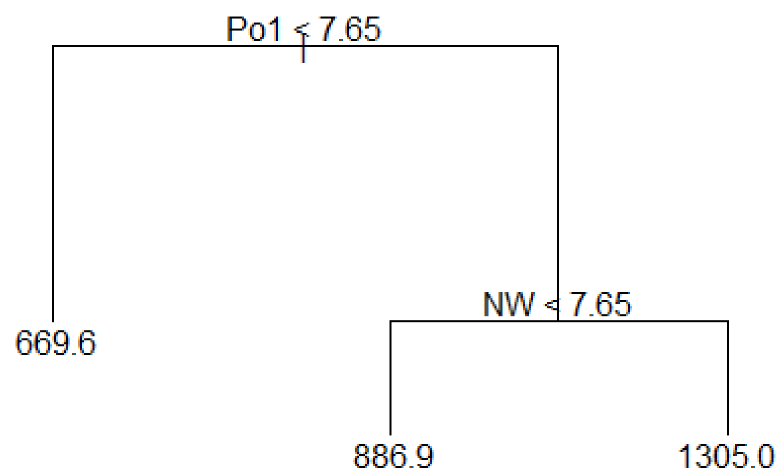
We calculated an R2 of near 72%.

We can prune the tree to prevent overfitting, by using the function `prune.tree()`. We only need to choose how many leafs we want the tree to have. Let's investigate with 3, 4 and 5 leaves.

```
model_tree_pruned_3 <- prune.tree(model_tree , best = 3)
summary(model_tree_pruned_3)
```

```
## 
## Regression tree:
## snip.tree(tree = model_tree, nodes = c(6L, 2L, 7L))
## Variables actually used in tree construction:
## [1] "Po1" "NW"
## Number of terminal nodes:  3
## Residual mean deviance:  76460 = 3364000 / 44
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -550.9  -181.8   -37.9     0.0   158.9   688.1
```

```
plot(model_tree_pruned_3)
text(model_tree_pruned_3)
```




```
pruned_tree_yhat_3 <- predict(model_tree_pruned_3)
pruned_tree_3_r2 <- R2_calc(pruned_tree_yhat_3, raw_data)
pruned_tree_3_r2
```

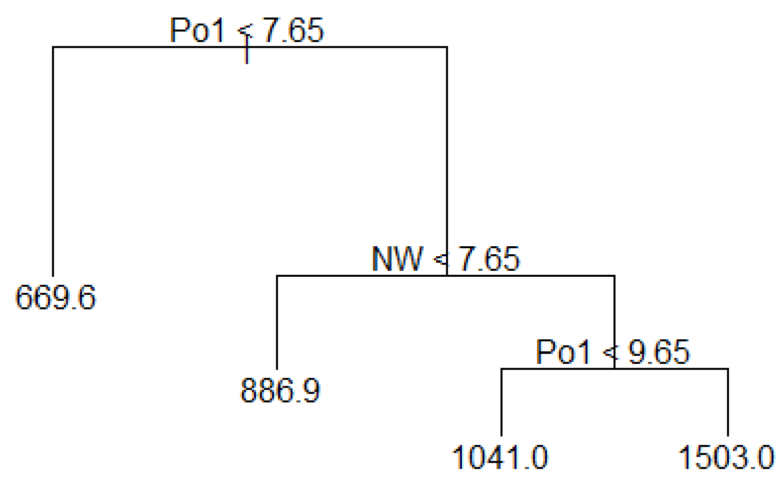
```
## [1] 0.5111061
```

```
model_tree_pruned_4 <- prune.tree(model_tree , best = 4)
summary(model_tree_pruned_4)
```

```
## 
## Regression tree:
## snip.tree(tree = model_tree, nodes = c(6L, 2L))
## Variables actually used in tree construction:
## [1] "Po1" "NW"
```

```
## Number of terminal nodes:  4 
## Residual mean deviance:  61220 = 2633000 / 43 
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## -573.90 -152.60   35.39    0.00  158.90  490.10
```

```
plot(model_tree_pruned_4)
text(model_tree_pruned_4)
```



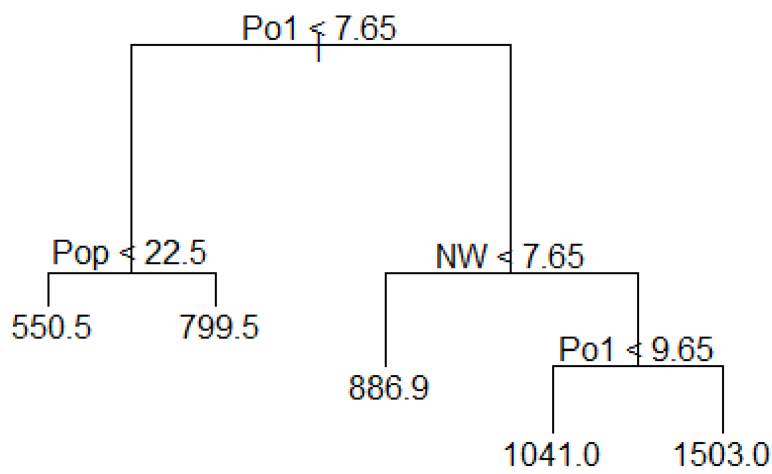

```
pruned_tree_yhat_4 <- predict(model_tree_pruned_4)
pruned_tree_4_r2 <- R2_calc(pruned_tree_yhat_4, raw_data)
pruned_tree_4_r2
```

```
## [1] 0.6174017
```

```
model_tree_pruned_5 <- prune.tree(model_tree , best = 5)
summary(model_tree_pruned_5)
```

```
## 
## Regression tree:
## snip.tree(tree = model_tree, nodes = c(4L, 6L))
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "NW" 
## Number of terminal nodes:  5 
## Residual mean deviance:  54210 = 2277000 / 42 
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  -573.9  -107.5    15.5     0.0   122.8   490.1
```

```
plot(model_tree_pruned_5)
text(model_tree_pruned_5)
```




```
pruned_tree_yhat_5 <- predict(model_tree_pruned_5)
pruned_tree_5_r2 <- R2_calc(pruned_tree_yhat_5, raw_data)
pruned_tree_5_r2

## [1] 0.6691333
```

As calculated the best model seems the tree pruned with 5 leaves: it gaves an R squared of around 67 %.

Random forest

Now we can use a random forest with 500 trees.

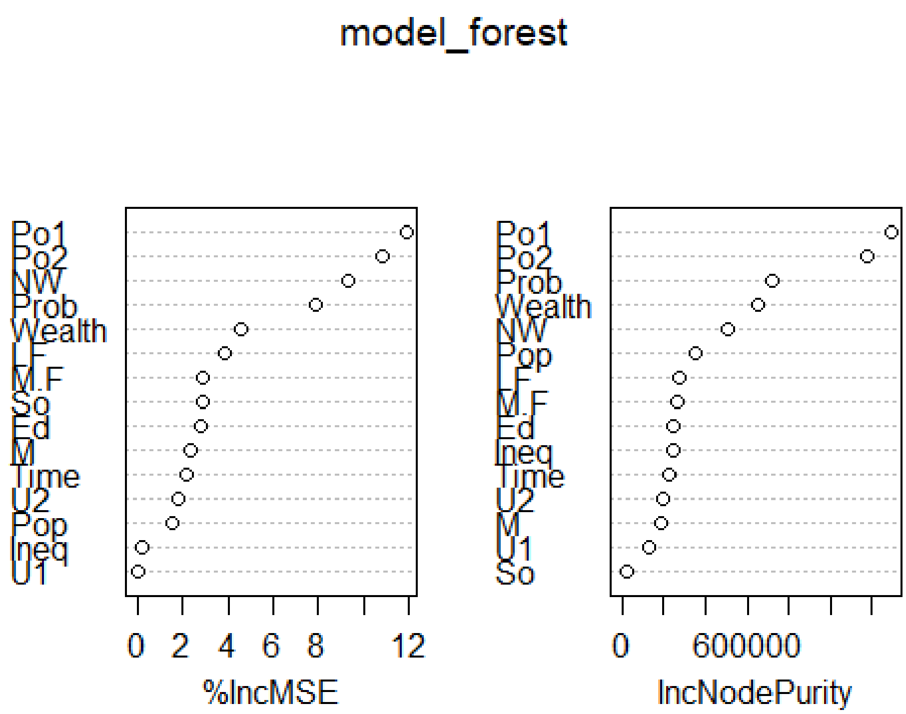
```
model_forest <- randomForest(Crime ~., raw_data, importance = TRUE, ntree =
500)
model_forest

##
## Call:
##  randomForest(formula = Crime ~ ., data = raw_data, importance = TRUE,
ntree = 500)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
```

```
##           Mean of squared residuals: 82617.37
##                     % Var explained: 43.57
```

```r
importance(model_forest)
```

```
##            %IncMSE IncNodePurity
## M       2.30706715     188669.79
## So      2.83705092      24824.93
## Ed      2.79500019     250519.88
## Po1    11.90074201    1292193.32
## Po2    10.84382663    1180597.40
## LF      3.90271783     274743.85
## M.F     2.86053480     263434.36
## Pop     1.55079227     353037.31
## NW      9.27667389     511844.38
## U1     -0.01202225     127143.07
## U2      1.84552751     196721.04
## Wealth  4.60664609     651559.62
## Ineq    0.23348414     244471.22
## Prob    7.88770708     725489.29
## Time    2.16628457     222730.25
```

```r
varImpPlot(model_forest)
```



Once the model is created we can again evaluate it’s R squared value

```r
crime_forest_yhat <- predict(model_forest)
R2_calc(crime_forest_yhat, raw_data)
```

```
## [1] 0.4356842
```

The R squared value of the model is around 40%. The random forest has a lower R squared value, but this may come from the fact that the tree model (even pruned) has an overfitting component.

Question 10.2

Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

Since the footbal world cup is near, an example of logistic regression model may be the fact that a penalty succeds (1) or fails (0). As predictors we can consider the level of fatigue of a player, power of kick, precision of kick, number of penalties he succeded in his career and level of strees.

Question 10.3.1

Using the GermanCredit data set germancredit.txt, use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.

First we need to load the libraries and the data from the temp *txt* file.

```
raw_data <- read.table('10.3germancreditSummer2018.txt', stringsAsFactors =
FALSE, header=FALSE)
head(raw_data) #view top  rows of dataset

##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172
##   V18  V19  V20 V21
## 1   1 A192 A201   1
## 2   1 A191 A201   2
## 3   2 A191 A201   1
## 4   2 A191 A201   1
## 5   2 A191 A201   2
## 6   2 A192 A201   1
```

We can initially change our response value in order to have zeroes (instead of 1) and ones (instead of 2)

```r
raw_data$V21[raw_data$V21 == 1] <- 1
raw_data$V21[raw_data$V21 == 2] <- 0
```

Once done, we can fit a logistic model with all factors in order to determine the more significants (we split 80 / 20)

```r
split <- sample(1:nrow(raw_data), size = round(0.8*(nrow(raw_data))))
train_df <- raw_data[split,]
test_df <- raw_data[-split,]

full_model <- glm(V21 ~ ., family = binomial(link="logit"), train_df)
summary(full_model)

##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train_df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8198  -0.7129   0.3540   0.7101   2.2647
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.547e-01  1.205e+00  -0.294 0.768431
## V1A12        2.155e-01  2.471e-01   0.872 0.383135
## V1A13        3.242e-01  4.326e-01   0.749 0.453571
## V1A14        1.595e+00  2.637e-01   6.048 1.47e-09 ***
## V2          -3.224e-02  1.045e-02  -3.084 0.002039 **
## V3A31        2.275e-01  6.026e-01   0.378 0.705765
## V3A32        7.260e-01  4.757e-01   1.526 0.126960
## V3A33        1.300e+00  5.299e-01   2.453 0.014179 *
## V3A34        1.712e+00  4.939e-01   3.466 0.000528 ***
## V4A41        1.108e+00  4.144e-01   2.674 0.007495 **
## V4A410       1.880e+00  9.552e-01   1.968 0.049079 *
## V4A42        5.932e-01  2.939e-01   2.018 0.043554 *
## V4A43        7.756e-01  2.822e-01   2.749 0.005984 **
## V4A44        7.749e-01  8.608e-01   0.900 0.368017
## V4A45       -1.147e-01  6.246e-01  -0.184 0.854295
## V4A46       -5.749e-02  4.890e-01  -0.118 0.906408
## V4A48        1.710e+00  1.206e+00   1.418 0.156067
## V4A49        5.165e-01  3.753e-01   1.376 0.168821
## V5          -1.579e-04  5.004e-05  -3.156 0.001598 **
## V6A62        2.706e-01  3.097e-01   0.874 0.382388
## V6A63        7.410e-01  4.939e-01   1.500 0.133495
## V6A64        1.979e+00  6.744e-01   2.934 0.003346 **
## V6A65        1.002e+00  2.977e-01   3.366 0.000762 ***
## V7A72        4.700e-01  4.689e-01   1.002 0.316196
## V7A73        4.959e-01  4.537e-01   1.093 0.274324
## V7A74        1.352e+00  5.012e-01   2.696 0.007010 **
## V7A75        6.447e-01  4.577e-01   1.409 0.158895
```

```
## V8           -3.324e-01  1.021e-01  -3.256 0.001130 **
## V9A92         1.229e-01  4.407e-01   0.279 0.780341
## V9A93         6.406e-01  4.295e-01   1.492 0.135826
## V9A94         1.836e-01  5.133e-01   0.358 0.720603
## V10A102      -3.716e-01  4.291e-01  -0.866 0.386567
## V10A103       1.022e+00  4.673e-01   2.187 0.028768 *
## V11          -6.749e-02  9.899e-02  -0.682 0.495360
## V12A122      -3.294e-01  2.831e-01  -1.163 0.244735
## V12A123      -1.577e-01  2.630e-01  -0.600 0.548793
## V12A124      -6.843e-01  4.564e-01  -1.499 0.133806
## V13           1.449e-02  1.039e-02   1.395 0.163103
## V14A142       5.954e-01  4.673e-01   1.274 0.202565
## V14A143       7.995e-01  2.661e-01   3.005 0.002658 **
## V15A152       3.235e-01  2.694e-01   1.201 0.229809
## V15A153       9.699e-01  5.302e-01   1.829 0.067342 .
## V16          -2.349e-01  2.127e-01  -1.104 0.269394
## V17A172      -4.371e-01  7.294e-01  -0.599 0.549004
## V17A173      -5.854e-01  6.961e-01  -0.841 0.400402
## V17A174      -2.247e-01  7.037e-01  -0.319 0.749513
## V18          -2.707e-01  2.827e-01  -0.958 0.338292
## V19A192       3.493e-01  2.262e-01   1.544 0.122540
## V20A202       1.130e+00  6.601e-01   1.711 0.087043 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 970.51  on 799  degrees of freedom
## Residual deviance: 707.66  on 751  degrees of freedom
## AIC: 805.66
##
## Number of Fisher Scoring iterations: 5
```

We can now refine the model by selecting only the significant attributes, based on their p-values.

```
refined_model <- glm(V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V10 + V14 +
V20,
                     family = binomial(link="logit"), train_df
                    )
summary(refined_model)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V10 +
##     V14 + V20, family = binomial(link = "logit"), data = train_df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6911  -0.7620   0.3929   0.7365   2.1340
```

```
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.702e-01  6.069e-01  -0.775 0.438444
## V1A12        2.150e-01  2.328e-01   0.923 0.355851
## V1A13        5.250e-01  4.025e-01   1.304 0.192087
## V1A14        1.612e+00  2.512e-01   6.417 1.39e-10 ***
## V2          -3.023e-02  9.743e-03  -3.103 0.001915 **
## V3A31        5.441e-01  5.736e-01   0.949 0.342828
## V3A32        9.476e-01  4.454e-01   2.127 0.033401 *
## V3A33        1.457e+00  5.132e-01   2.839 0.004526 **
## V3A34        1.837e+00  4.710e-01   3.900 9.62e-05 ***
## V4A41        1.070e+00  3.922e-01   2.728 0.006381 **
## V4A410       1.759e+00  8.660e-01   2.031 0.042278 *
## V4A42        3.857e-01  2.748e-01   1.404 0.160431
## V4A43        7.255e-01  2.676e-01   2.711 0.006706 **
## V4A44        6.400e-01  8.023e-01   0.798 0.425041
## V4A45       -2.228e-01  6.117e-01  -0.364 0.715701
## V4A46       -1.280e-01  4.662e-01  -0.275 0.783624
## V4A48        1.744e+00  1.188e+00   1.467 0.142289
## V4A49        6.071e-01  3.571e-01   1.700 0.089119 .
## V5          -1.048e-04  4.441e-05  -2.360 0.018282 *
## V6A62        1.808e-01  2.920e-01   0.619 0.535647
## V6A63        8.661e-01  4.769e-01   1.816 0.069338 .
## V6A64        1.889e+00  6.617e-01   2.855 0.004301 **
## V6A65        9.844e-01  2.805e-01   3.509 0.000449 ***
## V8          -2.483e-01  9.486e-02  -2.617 0.008867 **
## V10A102     -4.342e-01  4.180e-01  -1.039 0.298847
## V10A103      9.917e-01  4.379e-01   2.265 0.023524 *
## V14A142      4.907e-01  4.484e-01   1.094 0.273845
## V14A143      7.118e-01  2.548e-01   2.793 0.005224 **
## V20A202      1.026e+00  6.299e-01   1.629 0.103319
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 970.51  on 799  degrees of freedom
## Residual deviance: 744.46  on 771  degrees of freedom
## AIC: 802.46
##
## Number of Fisher Scoring iterations: 5
```

Now we can evaluate the refined model by calculating the confusion matrix. We added a step for converting the continuous output in a 0/1 response.

```
refined_model_yhat <- predict(refined_model, test_df, type = "response")
norm_refined_model_yhat <- as.integer(refined_model_yhat > 0.5)
table(test_df$V21, norm_refined_model_yhat)
```

```
##    norm_refined_model_yhat
##       0   1
##   0  29  35
##   1  16 120
```

Based on the resulting confusion matrix (row = true classification & col = model's classification) we got 29 false negatives and 20 false positives.

```
sensivity = 28/(28+29)
specificity = 123/(20+123)
```

With a sensivity = 0.4912281 and a specificity = 0.8601399

Question 10.3.2

Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

In other terms false positives (incorrectly identifying a bad customer as good) are 5 times worse than false negatives (incorrectly classifying a good customer as bad).

We may work on the 50% level used before to convert the continuous output in a 0/1 response. By looping from 8% to 99% we can calculate each time a cost function as FN + 5*FP.

```
for (level in 8:99) {
  norm_refined_model_yhat <- as.integer(refined_model_yhat > level/100)
  table_level <- table(test_df$V21, norm_refined_model_yhat)
  cost_level <- table_level[1,2] + 5*table_level[2,1]
  print (paste(level,cost_level))
}
```

```
## [1] "8 63"
## [1] "9 63"
## [1] "10 62"
## [1] "11 62"
## [1] "12 61"
## [1] "13 60"
## [1] "14 60"
## [1] "15 60"
## [1] "16 59"
## [1] "17 59"
## [1] "18 59"
## [1] "19 59"
## [1] "20 59"
## [1] "21 58"
## [1] "22 58"
## [1] "23 58"
```

```
## [1] "24 57"
## [1] "25 60"
## [1] "26 59"
## [1] "27 64"
## [1] "28 62"
## [1] "29 61"
## [1] "30 59"
## [1] "31 58"
## [1] "32 57"
## [1] "33 60"
## [1] "34 59"
## [1] "35 59"
## [1] "36 63"
## [1] "37 63"
## [1] "38 68"
## [1] "39 68"
## [1] "40 68"
## [1] "41 72"
## [1] "42 72"
## [1] "43 76"
## [1] "44 75"
## [1] "45 74"
## [1] "46 79"
## [1] "47 84"
## [1] "48 99"
## [1] "49 117"
## [1] "50 115"
## [1] "51 119"
## [1] "52 119"
## [1] "53 138"
## [1] "54 137"
## [1] "55 135"
## [1] "56 135"
## [1] "57 135"
## [1] "58 144"
## [1] "59 148"
## [1] "60 152"
## [1] "61 161"
## [1] "62 166"
## [1] "63 164"
## [1] "64 164"
## [1] "65 163"
## [1] "66 173"
## [1] "67 175"
## [1] "68 180"
## [1] "69 184"
## [1] "70 183"
## [1] "71 183"
## [1] "72 192"
## [1] "73 202"
```

```
## [1] "74 212"
## [1] "75 227"
## [1] "76 247"
## [1] "77 262"
## [1] "78 280"
## [1] "79 285"
## [1] "80 294"
## [1] "81 307"
## [1] "82 316"
## [1] "83 316"
## [1] "84 329"
## [1] "85 353"
## [1] "86 387"
## [1] "87 402"
## [1] "88 401"
## [1] "89 420"
## [1] "90 435"
## [1] "91 455"
## [1] "92 490"
## [1] "93 528"
## [1] "94 557"
## [1] "95 572"
## [1] "96 607"
## [1] "97 625"
## [1] "98 635"
## [1] "99 670"
```

By observing the minimum (53) and maximum values (695) of the cost value we can suggest a threshold at 29% to be under the 10% of the maximum cost value.