HW4

6/4/2020

Question 9.1

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. (Note that to first scale the data, you can include scale. = TRUE to scale as part of the PCA function. Don't forget that, to make a prediction for the new city, you'll need to unscale the coefficients (i.e., do the scaling calculation in reverse)!)

```
cr_data <- read.table("/Users/chintan/Downloads/6501/crimedata.txt",stringsAsFactors = FA
  LSE, header = TRUE)
head(cr_data)
```

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

```
library(DAAG)
```

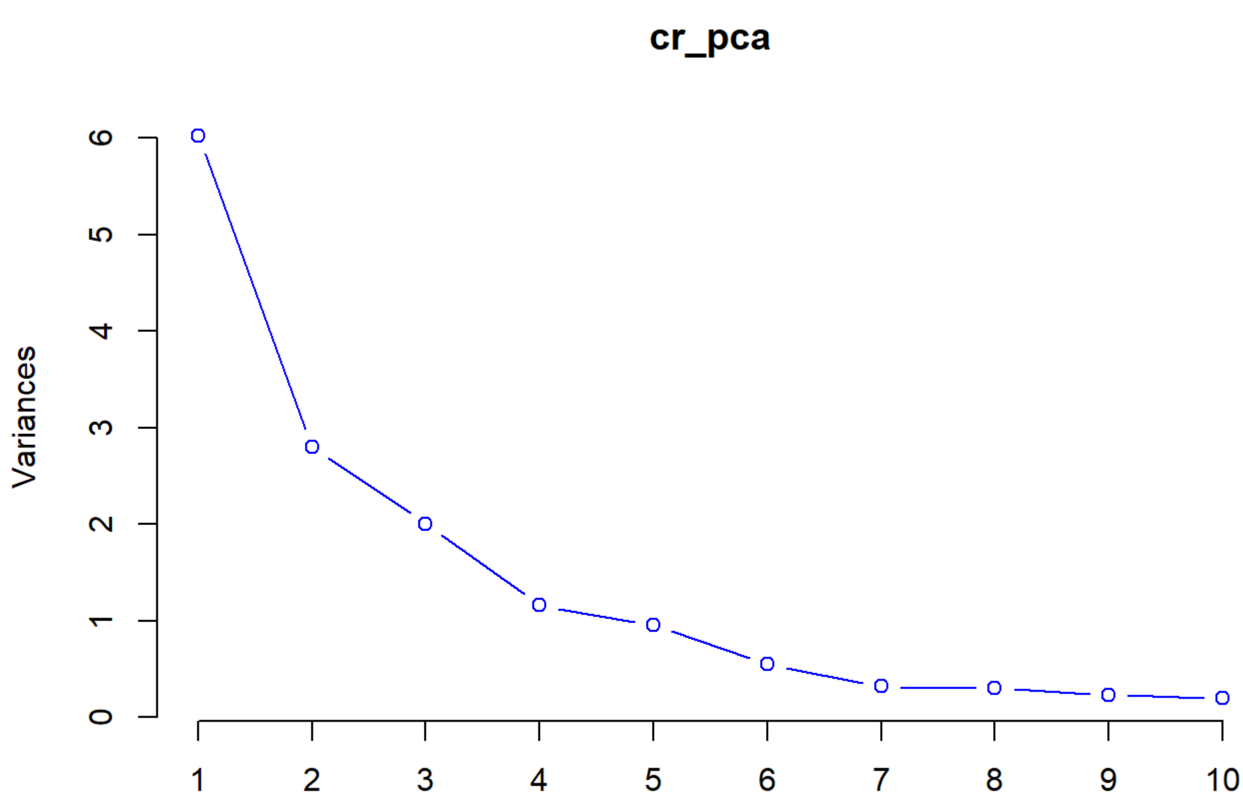
```
## Loading required package: lattice
```

```
cr_pca <- prcomp(cr_data[,-16], center=T, scale=T)
summary(cr_pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.4534 1.6739 1.4160 1.07806 0.97893 0.74377 0.56729
## Proportion of Variance 0.4013 0.1868 0.1337 0.07748 0.06389 0.03688 0.02145
## Cumulative Proportion  0.4013 0.5880 0.7217 0.79920 0.86308 0.89996 0.92142
##                            PC8     PC9    PC10    PC11    PC12    PC13   PC14
## Standard deviation     0.55444 0.48493 0.44708 0.41915 0.35804 0.26333 0.2418
## Proportion of Variance 0.02049 0.01568 0.01333 0.01171 0.00855 0.00462 0.0039
## Cumulative Proportion  0.94191 0.95759 0.97091 0.98263 0.99117 0.99579 0.9997
##                           PC15
## Standard deviation     0.06793
## Proportion of Variance 0.00031
## Cumulative Proportion  1.00000
```

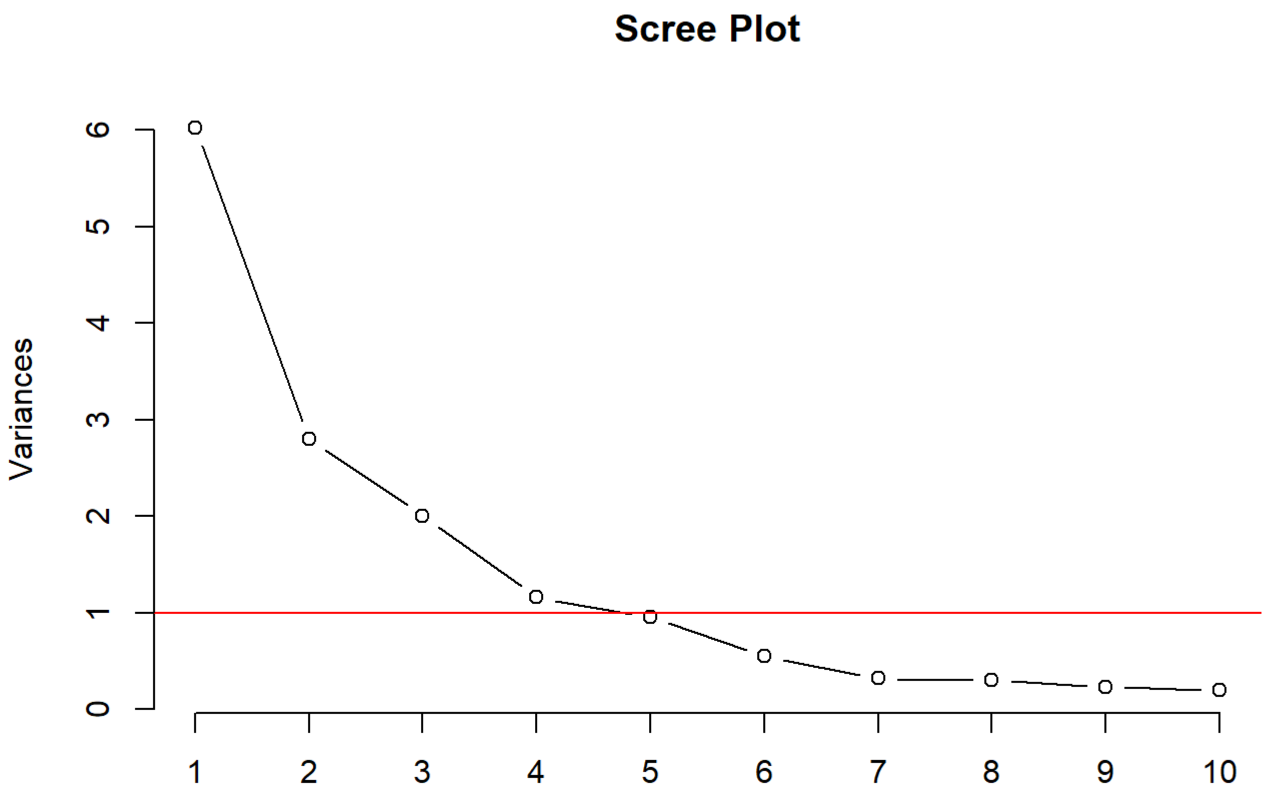
As we know PCA ranks the proportion of each principal component. PCs 1-3 have a significant amount of variation relative to the rest as shown in the figure below.

```
plot(cr_pca,  type  = "l",col= "blue")
```



As per Kaiser Method any variable with stdev greater than one is important. So we will consider top 5 variables to build our linear regression model. While looking at the graph we can say that after 7th predictors the line is almost flat. First we will build our model using top five predictors and later we will increase them to 7.

```
screeplot(cr_pca,main = "Scree Plot", type = "line")
abline(h=1, col="red")
```




```
pca_data <- cbind(cr_pca$x[,1:5], cr_data[,16])

lm_model <- lm(V6~.,data = as.data.frame(pca_data))
summary(lm_model)
```

```
##
## Call:
## lm(formula = V6 ~ ., data = as.data.frame(pca_data))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -420.79 -185.01   12.21  146.24  447.86
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      35.59  25.428  < 2e-16 ***
## PC1            65.22      14.67   4.447 6.51e-05 ***
## PC2           -70.08      21.49  -3.261  0.00224 **
## PC3            25.19      25.41   0.992  0.32725
## PC4            69.45      33.37   2.081  0.04374 *
## PC5          -229.04      36.75  -6.232 2.02e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 244 on 41 degrees of freedom
## Multiple R-squared:  0.6452, Adjusted R-squared:  0.6019
## F-statistic: 14.91 on 5 and 41 DF,  p-value: 2.446e-08
```

Transformation

```
    #from above table let's get our intercept value....
beta <- lm_model$coefficients[1]
    #beta vector with our coefficients......
beta_vec <- lm_model$coefficients[2:6]
    #multply the coefficients by our rotated matrix, A to create alpha vector
alpha <- cr_pca$rotation[,1:5] %*% beta_vec

mu <- sapply(cr_data[,1:15],mean)
sigma <- sapply(cr_data[,1:15],sd)

    #we recover our original alpha values by dividing the alpha vector by sigma
    #and our original beta by subtracting from the intercept the sum of (alpha*mu)/sigma

orig_alpha <- alpha/sigma
orig_beta <- beta - sum(alpha*mu /sigma)

  # Y = aX + b , where a is our scaled alpha and b is our original intercept
estimates <- as.matrix(cr_data[,1:15]) %*% orig_alpha + orig_beta



SSE = sum((estimates - cr_data[,16])^2)
SStot = sum((cr_data[,16] - mean(cr_data[,16]))^2)

  #we can now use our estimates to calculate the R-squared values
R2 <- 1 - SSE/SStot
R2
```

```
## [1] 0.6451941
```

We use this model to predict the crime rate for the given data in the exercise 7.2

```
given_dp<- 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, We
  alth = 3200, Ineq = 20.1, Prob = 0.040,Time = 39.0)



pred_df <- data.frame(predict(cr_pca, given_dp))
pred <- predict(lm_model, pred_df)
pred
```

```
##        1
## 1388.926
```

Whole, idea of this exericse is to reduce the number of predictors and check the models'spredictions accuracy. When I applied the PCA on the crime data, it indicated that top 5 predeictors are important and rest of them will less impact. I build the regression model using top 5 prdictors and it gave me R squared value of 0.64 and Adjusted R-squred of 0.60. It predcits crime rate of 1388.926 which is acceptable. I tried incresing the number of predictors to 15. Below is the R-squred value and number of predictors.

No of Predictors R-Squared Prediction 6 0.6586 1248.427 7 0.6882 1230.418 8 0.6899 1190.455 9 0.692 1136.169 10 0.6963 1110.684 11 0.6974 1100.079 12 0.7693 1581.932 13 0.7724 1433.792 14 0.7911 957.264 15 0.8031 155.4349

Based on this analysys it's look like we can pick any number of predictors between 7 to 11 and it will give us good R-squared value and good predictions of crime.

This is almost same nubmer of predicores as I had used in exercise 7.2 (by looking only at p values). Below are some good links that explains r- squred and Residuals

https://statisticsbyjim.com/regression/interpret-r-squared-regression/ (https://statisticsbyjim.com/regression/interpret-r-squared-regression/)

https://drsimonj.svbtle.com/visualising-residuals (https://drsimonj.svbtle.com/visualising-residuals)

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) a regression tree model, and (b) 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 (i.e., don't just stop when you have a good model, but interpret it too).

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
library(tree)
library(caret)
```

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

```
## Registered S3 method overwritten by 'cli':
##   method     from
##   print.tree tree
```

```
##
## Attaching package: 'ggplot2'
```

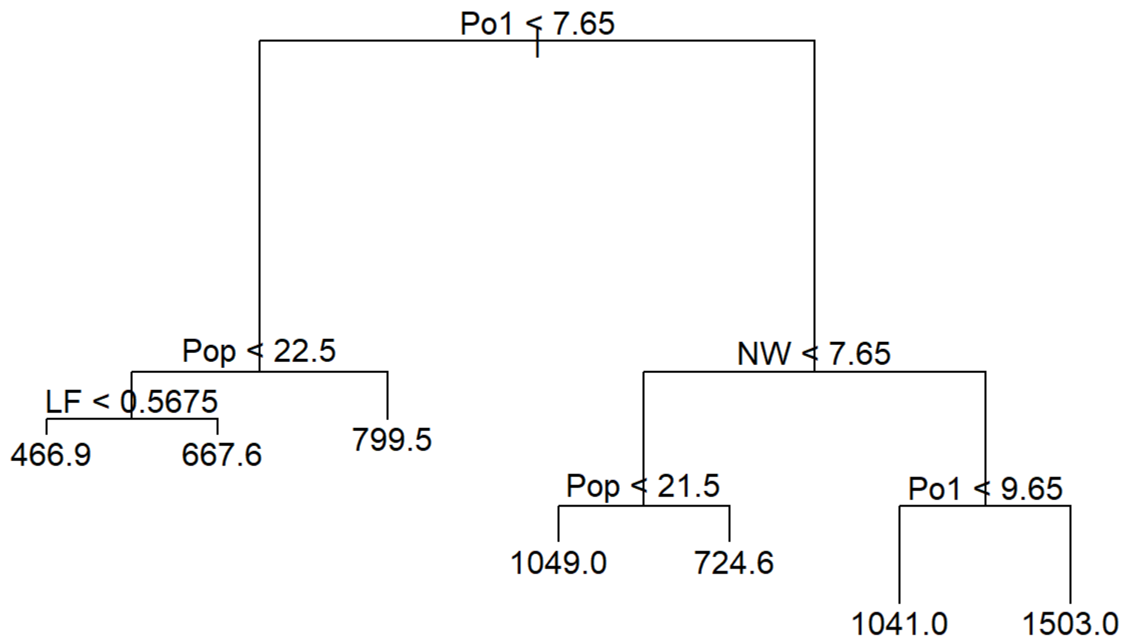
```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
tree_model <- tree(Crime~.,data = cr_data)
summary(tree_model)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = cr_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
```

Good thing about tree is that we can see it was split up.

```
plot(tree_model)
text(tree_model)
```




```
yhat <- predict(tree_model)
ssres <- sum((yhat-cr_data$Crime)^2)
sstot <- sum((cr_data$Crime - mean(cr_data$Crime))^2)
R2 <- 1-(ssres/sstot)
R2
```

```
## [1] 0.7244962
```

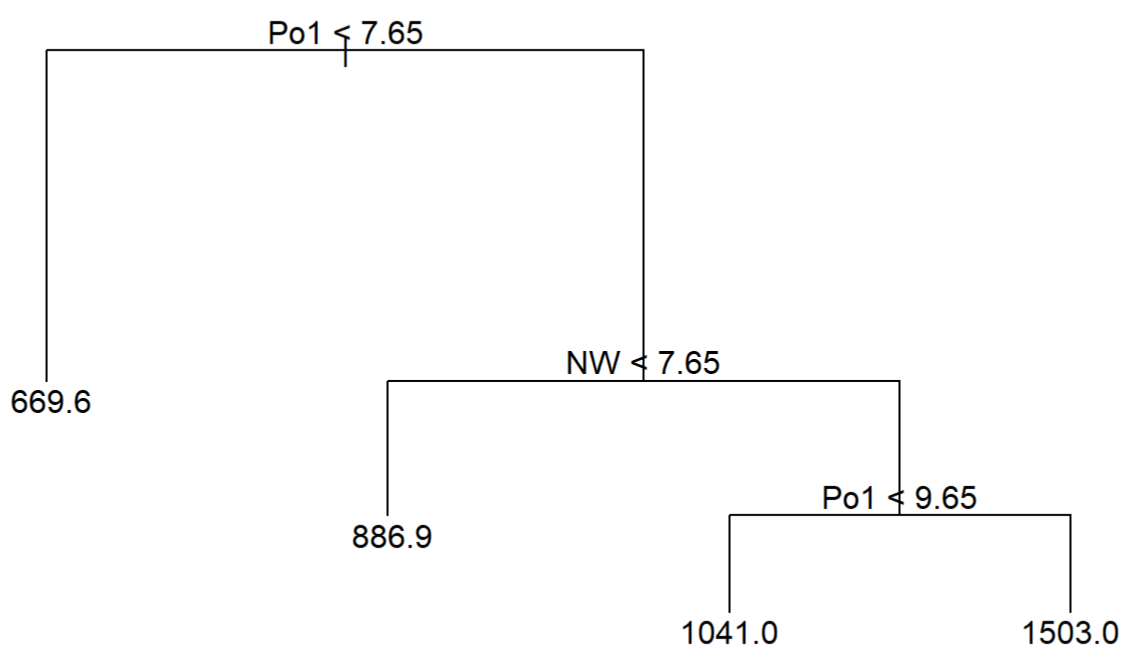
This is our basic regression model's R2 value. We will built other regression model and compare R2 values.

Let's remove some of the leaves from our tree.

```
prune_tree <- prune.tree(tree_model, best =4)
summary(prune_tree)
```

```
##
## Regression tree:
## snip.tree(tree = tree_model, 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(prune_tree)
text(prune_tree)
```




```
yhat <- predict(prune_tree)
ssres <- sum((yhat-cr_data$Crime)^2)
sstot <- sum((cr_data$Crime - mean(cr_data$Crime))^2)
R2_pr <- 1-(ssres/sstot)
R2_pr
```

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

Once we remove the some leaves from our model, it only used two predictors. IF you look closely the prune model has the same leaves and branches as the basic model. Without doubt the basic model has better R-squared value and we should use that model. But before that we need to validate the models and for that we need to divide our data into training and test.

```
set.seed(42)
sample <- round(sample(nrow(cr_data)*0.7))
train_data <- cr_data[sample,]
test_data <- cr_data[-sample,]
```

Let's validate our model using test data.

```
cr <- test_data$Crime
pred_basic <- predict(tree_model, test_data[,1:15])
pred_prune <- predict(prune_tree, test_data[,1:15])
rmse_basic <- sqrt(mean((pred_basic-cr)^2))
rmse_prune <- sqrt(mean((pred_prune-cr)^2))
rmse_basic
```

```
## [1] 163.4527
```

```
rmse_prune
```

```
## [1] 221.3876
```

Even though the basic model as better R2 value the prune model gives better results.

B) Randomforest

```
rf_crime <- randomForest(Crime~., data = cr_data, importance = TRUE , nodsize  = 4)

rf_yhat <- predict(rf_crime)
rf_ssres <- sum((rf_yhat-cr_data$Crime)^2)
rf_sstot <- sum((cr_data$Crime - mean(cr_data$Crime))^2)
rf_R2 <- 1-(rf_ssres/rf_sstot)
rf_R2
```

```
## [1] 0.4101854
```

This randomforest model gives the worst R2 value than standard tree. I ran this model for different node sizes and for node size 4 I got the best R2 value. Let's create new model using test data…

```
set.seed(40)
rf_crime_test <- randomForest(Crime~., data = train_data, nodsize  = 4,importance= TRUE)
rf_yhat_test <- predict(rf_crime_test)
rf_ssres_test <- sum((rf_yhat_test - train_data$Crime)^2)
rf_sstot_test <- sum((train_data$Crime - mean(train_data$Crime))^2)
rf_R2_test <- 1-(rf_ssres_test/rf_sstot_test)
rf_R2_test
```

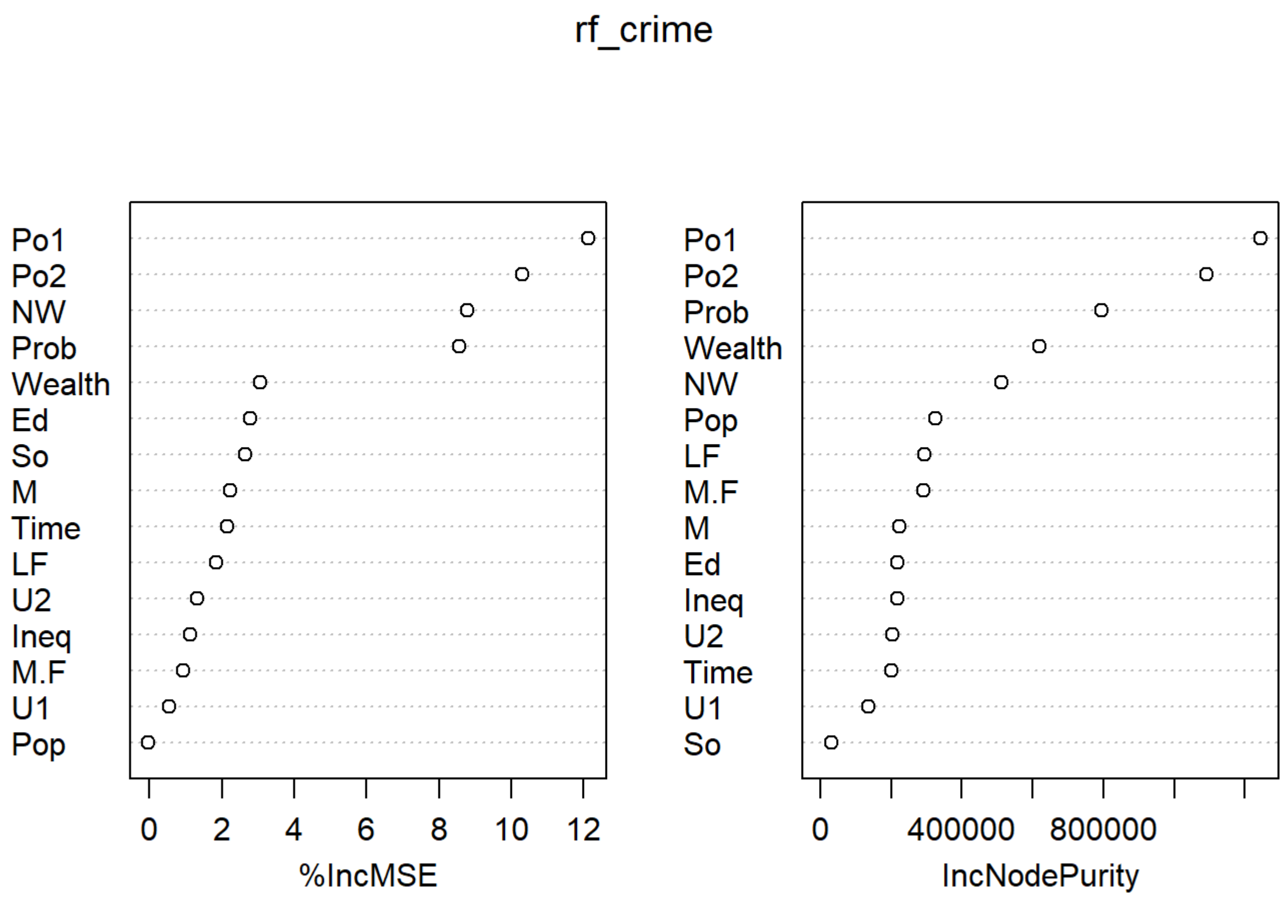
```
## [1] 0.4544708
```

Below function shows the most important predictors according to randomforest model.

```
importance(rf_crime)
```

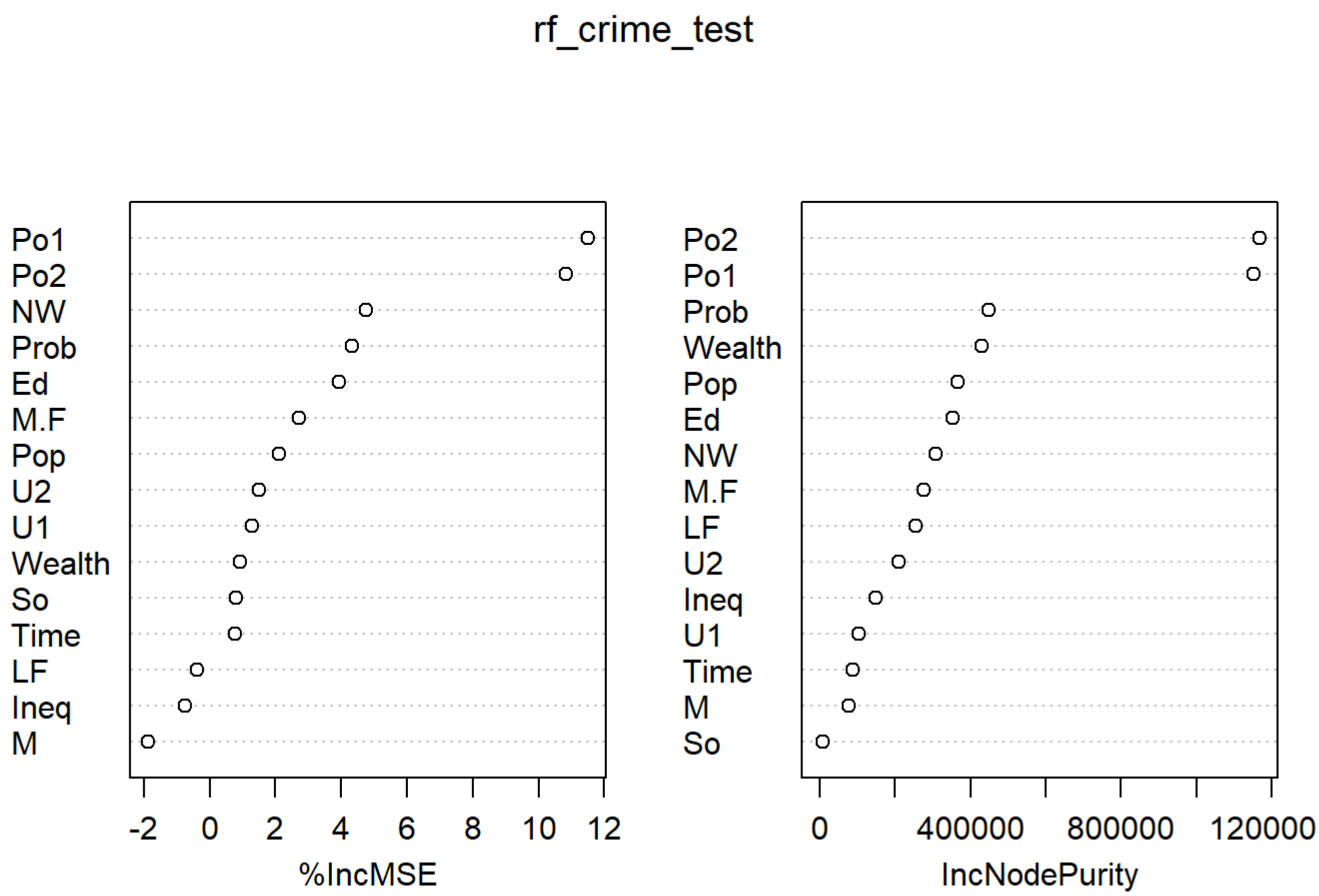
```
##           %IncMSE IncNodePurity
## M       2.2268668     223817.53
## So      2.6349415      31976.75
## Ed      2.7845561     219654.38
## Po1    12.1449851    1245413.47
## Po2    10.3006240    1092076.98
## LF      1.8343147     295151.93
## M.F     0.9286452     291505.77
## Pop    -0.0474878     325602.86
## NW      8.7893430     511995.96
## U1      0.5359285     137436.39
## U2      1.3274800     203333.21
## Wealth  3.0704952     618646.34
## Ineq    1.1136153     218674.08
## Prob    8.5633758     795535.65
## Time    2.1401844     200822.78
```

```
varImpPlot(rf_crime)
```



According to our basic “rf_crime” model the Po2, Po1 and Prob are the main important predictors. When I ran the rf_crime_test model it also shows that Po1 and Po2 are the most important predictors.

```
varImpPlot(rf_crime_test)
```



Based on this small dataset it seems that increasing the number of variables it actually decreases the accuracy of this model.

Q 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.

We could use logistic regression to predict whether a person will buy a car or not. It has limited outcome either person will buy the car or not. The predictors we can use are age, gender, salary, previous car type and race. Using these predictors we can build a model which gives us an outcome of 0 or 1. 1 means person will buy the car and 0 means he will not. This is the example of binary logistic regression model. There are other kind of logistic models as well. Like “Multinomial Logistic regression” where instead of getting 0 or 1 as output we can get three or more categories as output like which food veg, non-veg, vegan is preferred more on the flight. The other one is “Ordinal Logistic” regression where the outcome is three or more categories with order.

Question 10.3 (1 & 2 combine)

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german (http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german) / (description at

http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29 (http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29)), 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.

```
ger_data <- read.table("/Users/chintan/Downloads/6501/germancredit.txt",stringsAsFactors
  = FALSE, header = TRUE)
head(ger_data)
```

```
##   A11 X6 A34 A43 X1169 A65 A75 X4 A93 A101 X4.1 A121 X67 A143 A152 X2 A173 X1
## 1 A12 48 A32 A43  5951 A61 A73  2 A92 A101    2 A121  22 A143 A152  1 A173  1
## 2 A14 12 A34 A46  2096 A61 A74  2 A93 A101    3 A121  49 A143 A152  1 A172  2
## 3 A11 42 A32 A42  7882 A61 A74  2 A93 A103    4 A122  45 A143 A153  1 A173  2
## 4 A11 24 A33 A40  4870 A61 A73  3 A93 A101    4 A124  53 A143 A153  2 A173  2
## 5 A14 36 A32 A46  9055 A65 A73  2 A93 A101    4 A124  35 A143 A153  1 A172  2
## 6 A14 24 A32 A42  2835 A63 A75  3 A93 A101    4 A122  53 A143 A152  1 A173  1
##   A192 A201 X1.1
## 1 A191 A201    2
## 2 A191 A201    1
## 3 A191 A201    1
## 4 A191 A201    2
## 5 A192 A201    1
## 6 A191 A201    1
```

Let's look at the response column in the data set. The response values are 1 and 2. The glm recognize 0 and 1 as response/ Let convert them to 0 and 1.

```
table(ger_data$X1.1)
```

```
##
##   1   2
## 699 300
```

```
ger_data$X1.1[ger_data$X1.1==1] <- 0
ger_data$X1.1[ger_data$X1.1==2] <- 1
```

Let's devide that data into training and test.

```r
set.seed(40)

sample <- sample(nrow(ger_data)*0.7)
train <- ger_data[sample,]
test <- ger_data[-sample,]

table(train$X1.1)
```

```
##
##   0   1
## 492 207
```

```r
table(test$X1.1)
```

```
##
##   0   1
## 207  93
```

Let’s build the model using train dataset.

```r
train_model <- glm(X1.1~., family = binomial(link = "logit"), data = train)
summary(train_model)
```

```
##
## Call:
## glm(formula = X1.1 ~ ., family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7424  -0.6878  -0.3552   0.6638   2.7025
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  8.074e-01  1.317e+00   0.613 0.539988
## A11A12      -1.647e-01  2.627e-01  -0.627 0.530652
## A11A13      -1.041e+00  4.368e-01  -2.383 0.017185 *
## A11A14      -1.744e+00  2.909e-01  -5.994 2.05e-09 ***
## X6           2.883e-02  1.109e-02   2.600 0.009322 **
## A34A31       6.536e-01  6.926e-01   0.944 0.345315
## A34A32      -6.825e-01  5.092e-01  -1.340 0.180162
## A34A33      -9.092e-01  5.558e-01  -1.636 0.101893
## A34A34      -1.526e+00  5.305e-01  -2.876 0.004024 **
## A43A41      -1.872e+00  4.936e-01  -3.792 0.000150 ***
## A43A410     -1.555e+00  8.472e-01  -1.836 0.066405 .
## A43A42      -8.266e-01  3.200e-01  -2.583 0.009788 **
## A43A43      -8.695e-01  3.039e-01  -2.861 0.004219 **
## A43A44      -1.274e-01  9.259e-01  -0.138 0.890540
## A43A45      -5.633e-01  6.639e-01  -0.849 0.396139
## A43A46       4.304e-02  4.544e-01   0.095 0.924543
## A43A48      -2.362e+00  1.318e+00  -1.792 0.073103 .
## A43A49      -7.909e-01  4.184e-01  -1.890 0.058718 .
## X1169        1.153e-04  5.571e-05   2.069 0.038541 *
## A65A62      -2.387e-01  3.460e-01  -0.690 0.490245
## A65A63      -4.463e-01  5.256e-01  -0.849 0.395763
## A65A64      -1.641e+00  6.363e-01  -2.579 0.009917 **
## A65A65      -8.163e-01  3.166e-01  -2.578 0.009928 **
## A75A72      -2.950e-01  5.417e-01  -0.545 0.586084
## A75A73      -4.285e-01  5.130e-01  -0.835 0.403553
## A75A74      -1.198e+00  5.564e-01  -2.153 0.031348 *
## A75A75      -5.057e-01  5.144e-01  -0.983 0.325502
## X4           3.564e-01  1.079e-01   3.304 0.000954 ***
## A93A92      -4.949e-01  4.593e-01  -1.077 0.281291
## A93A93      -1.297e+00  4.492e-01  -2.887 0.003885 **
## A93A94      -3.874e-01  5.378e-01  -0.720 0.471292
## A101A102     7.793e-01  4.842e-01   1.610 0.107463
## A101A103    -1.100e+00  5.260e-01  -2.091 0.036573 *
## X4.1         1.401e-02  1.047e-01   0.134 0.893571
## A121A122     4.032e-01  3.136e-01   1.286 0.198569
## A121A123     1.558e-01  2.837e-01   0.549 0.582925
## A121A124     8.670e-01  5.126e-01   1.691 0.090758 .
## X67         -1.367e-02  1.140e-02  -1.199 0.230573
```

```
## A143A142     6.375e-02  5.019e-01   0.127 0.898919
## A143A143    -6.353e-01  2.917e-01  -2.178 0.029390 *
## A152A152    -2.119e-01  2.939e-01  -0.721 0.470852
## A152A153    -9.053e-01  5.846e-01  -1.549 0.121436
## X2           3.424e-01  2.252e-01   1.520 0.128418
## A173A172    -7.122e-02  8.638e-01  -0.082 0.934296
## A173A173     1.301e-02  8.325e-01   0.016 0.987527
## A173A174     1.434e-01  8.210e-01   0.175 0.861328
## X1           4.910e-01  3.096e-01   1.586 0.112729
## A192A192    -3.046e-01  2.500e-01  -1.218 0.223213
## A201A202    -1.419e+00  8.185e-01  -1.733 0.083020 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 849.36  on 698  degrees of freedom
## Residual deviance: 612.57  on 650  degrees of freedom
## AIC: 710.57
##
## Number of Fisher Scoring iterations: 5
```

Let's check how well this model predicts ….

```
test_predict <- predict(train_model, newdata = test[,-21], type = "response")

test1 <- table(test$X1.1, test_predict>0.5)
test1
```

```
##
##     FALSE TRUE
##   0   174   33
##   1    40   53
```

Our model classifies 33 customers as false positive with the threshold of 0.5. Since the cost of the falsely identifying bad customer as good customer is significantly high we can increase our threshold to be 0.7 And also we can remove some of the unimportant predictors as we did in previous examples and observe the results.

```
test2 <- table(test$X1.1, test_predict>0.7)
test2
```

```
##
##     FALSE TRUE
##   0   196   11
##   1    59   34
```

We can see that when we increase the threshold to 0.7 the number of false positive went down to 11 and number of false negative went up to 59 as expected. Now, let's try to reduce the number of predictors by only considering the ones which has P<= 0.1 value.

```
new_trainmodel <- glm(X1.1~ A11+X6+A34+A43+X1169+A65+A75+A93+A101+A121+A173+A201,
                      family = binomial(link = "logit"), data = train)
summary(new_trainmodel)
```

```
##
## Call:
## glm(formula = X1.1 ~ A11 + X6 + A34 + A43 + X1169 + A65 + A75 +
##     A93 + A101 + A121 + A173 + A201, family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8392  -0.7155  -0.3886   0.7262   2.6162
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.317e+00  9.821e-01   1.341 0.180082
## A11A12      -2.253e-01  2.524e-01  -0.893 0.371974
## A11A13      -1.143e+00  4.224e-01  -2.706 0.006808 **
## A11A14      -1.712e+00  2.795e-01  -6.127 8.97e-10 ***
## X6           3.698e-02  1.044e-02   3.543 0.000395 ***
## A34A31       8.205e-01  6.630e-01   1.238 0.215859
## A34A32      -8.656e-01  4.878e-01  -1.775 0.075945 .
## A34A33      -9.193e-01  5.454e-01  -1.685 0.091900 .
## A34A34      -1.469e+00  5.164e-01  -2.845 0.004444 **
## A43A41      -1.768e+00  4.690e-01  -3.770 0.000163 ***
## A43A410     -1.376e+00  8.142e-01  -1.691 0.090928 .
## A43A42      -7.865e-01  3.042e-01  -2.585 0.009730 **
## A43A43      -7.710e-01  2.914e-01  -2.646 0.008146 **
## A43A44      -1.351e-01  8.653e-01  -0.156 0.875911
## A43A45      -3.537e-01  6.498e-01  -0.544 0.586207
## A43A46       8.340e-02  4.384e-01   0.190 0.849110
## A43A48      -1.976e+00  1.336e+00  -1.479 0.139133
## A43A49      -6.739e-01  4.055e-01  -1.662 0.096483 .
## X1169        2.115e-05  4.832e-05   0.438 0.661571
## A65A62      -1.893e-01  3.349e-01  -0.565 0.571906
## A65A63      -6.047e-01  5.081e-01  -1.190 0.233945
## A65A64      -1.350e+00  6.052e-01  -2.231 0.025704 *
## A65A65      -6.804e-01  3.037e-01  -2.240 0.025078 *
## A75A72      -1.785e-01  5.161e-01  -0.346 0.729406
## A75A73      -4.735e-01  4.963e-01  -0.954 0.340092
## A75A74      -1.168e+00  5.348e-01  -2.184 0.028984 *
## A75A75      -5.236e-01  4.942e-01  -1.060 0.289289
## A93A92      -4.605e-01  4.410e-01  -1.044 0.296376
## A93A93      -1.021e+00  4.269e-01  -2.393 0.016733 *
## A93A94      -3.512e-01  5.178e-01  -0.678 0.497631
## A101A102     9.667e-01  4.743e-01   2.038 0.041539 *
## A101A103    -1.039e+00  5.067e-01  -2.051 0.040292 *
## A121A122     3.647e-01  3.023e-01   1.206 0.227673
## A121A123     2.571e-01  2.729e-01   0.942 0.346278
## A121A124     3.908e-01  3.622e-01   1.079 0.280630
## A173A172     3.245e-01  8.097e-01   0.401 0.688556
```

```
## A173A173     3.400e-01  7.853e-01   0.433 0.665064
## A173A174     4.128e-01  7.664e-01   0.539 0.590176
## A201A202    -1.528e+00  8.167e-01  -1.871 0.061295 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 849.36  on 698  degrees of freedom
## Residual deviance: 638.96  on 660  degrees of freedom
## AIC: 716.96
##
## Number of Fisher Scoring iterations: 5
```

```
test_predict_new <- predict(new_trainmodel, newdata = test[,-21], type = "response")
test3 <- table(test$X1.1, test_predict_new>0.7)
test3
```

```
##
##     FALSE TRUE
##   0   197   10
##   1    65   28
```

We can see that there is slight improvement. The true positive number went up by 1 and false negative went up to 65. Thus we can say that there is very less chance that this model will classify the bad customer as good customer.