GTx: ISYE6501x - Homework 4

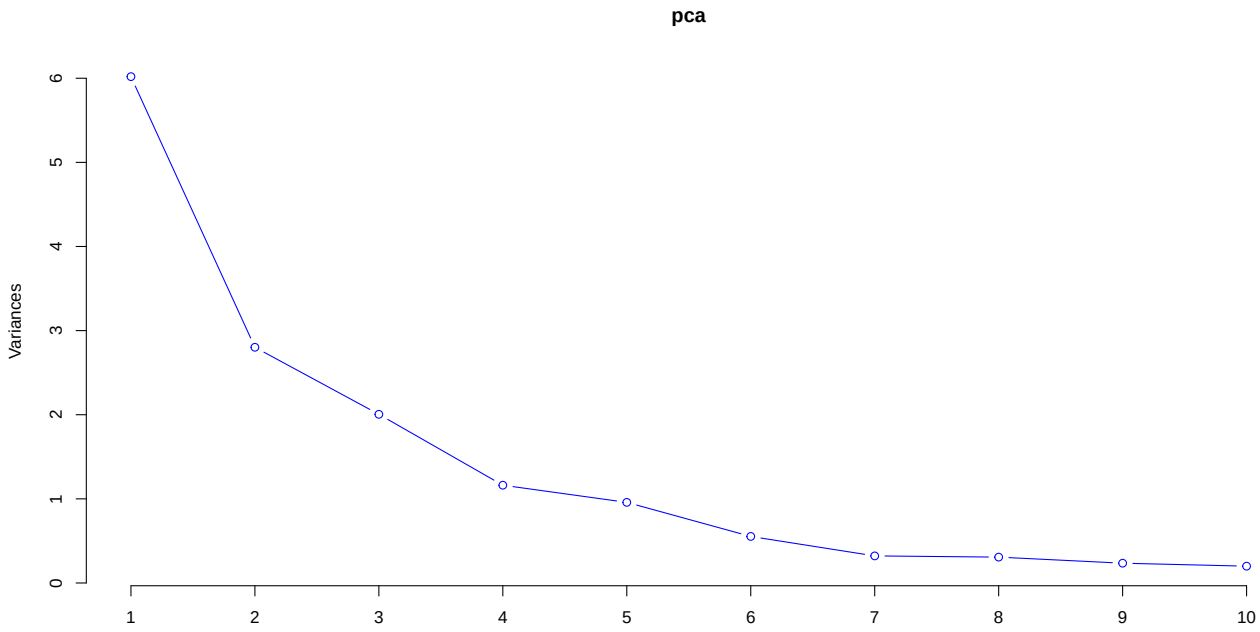
Muh Alif Ahsanul Islam

06/09/2019

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)!)

```
rm(list=ls())
crime_df = read.table('uscrime.txt', header=TRUE)
pca = prcomp(x = crime_df[,1:15], scale=TRUE)
screeplot(pca, type='lines', col='blue')
```

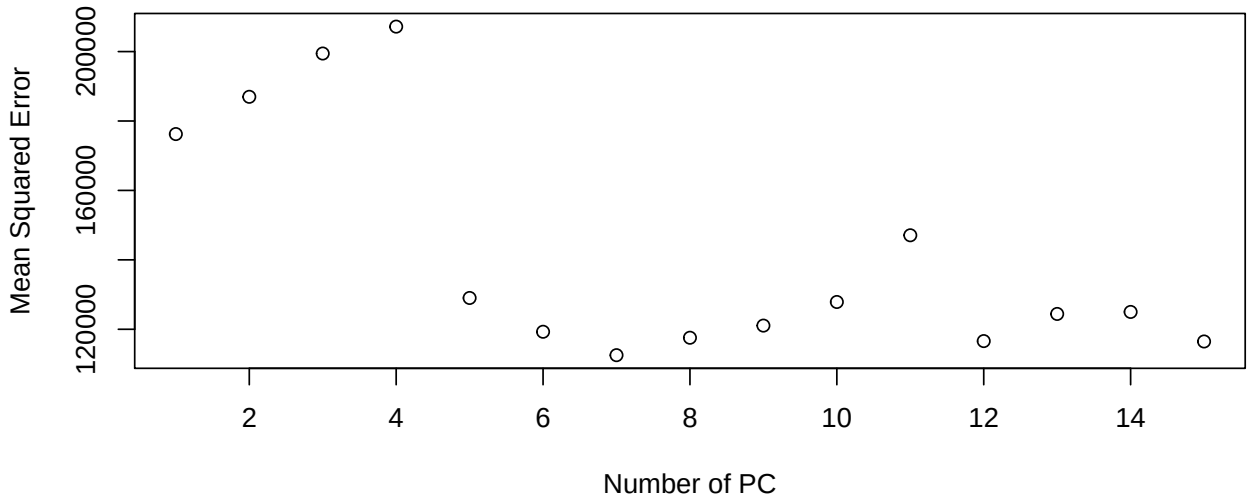


First I will choose how many principal components to use by doing 23-fold cross validation on linear regression model and choose number of principal component to use.

The code is: (i don't show it in R because it will output many images)

```
cv_res = NULL
for (n_pca_comp in 1:ncol(pca$x)){
  pca_df = as.data.frame(cbind(pca$x[,1:n_pca_comp], Crime=crime_df[,16]))
  cv = cv.lm(data=pca_df, form.lm=Crime~., m=23, printit=FALSE)
  mean_squared_error = attr(cv, 'ms')
  cv_res = rbind(cv_res,
                 data.frame(n_pca_comp, mean_squared_error))
}
```

```r
plot(x=cv_res$n_pca_comp, y=cv_res$mean_squared_error,
     xlab='Number of PC', ylab='Mean Squared Error')
```



Lowest mean squared error happens on number of PC = 7. When number of PC is 12 and 15, the mean squared error is also small, but I will argue that simple model is often better (Occam's razor principle).

```r
n_pc = 7
pca_df = as.data.frame(cbind(pca$x[,1:n_pc], Crime=crime_df[,16]))
model = lm(Crime~., data=pca_df)
summary(model)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = pca_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -475.41 -141.65   34.73  137.25  412.32
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   905.09      34.21  26.454  < 2e-16 ***
## PC1            65.22      14.10   4.626 4.04e-05 ***
## PC2           -70.08      20.66  -3.392   0.0016 **
## PC3            25.19      24.42   1.032   0.3086
## PC4            69.45      32.08   2.165   0.0366 *
## PC5          -229.04      35.33  -6.483 1.11e-07 ***
## PC6           -60.21      46.50  -1.295   0.2029
## PC7           117.26      60.96   1.923   0.0617 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 234.6 on 39 degrees of freedom
## Multiple R-squared:  0.6882, Adjusted R-squared:  0.6322
## F-statistic:  12.3 on 7 and 39 DF,  p-value: 3.513e-08
```

```r
actual = crime_df$Crime
predicted = model$fitted.values
rss <- sum((predicted - actual) ^ 2)  ## residual sum of squares
```

```
tss <- sum((actual - mean(actual)) ^ 2)  ## total sum of squares
rsq <- 1 - rss/tss
sprintf('R squared is: %.2f', rsq)
```

```
## [1] "R squared is: 0.69"
```

Model coefficient in terms of original predictos:

```
model_coef = model$coefficients[2:length(model$coefficients)]%*%t(pca$rotation[,1:(length(model$coeffic
model_coef
```

```
##             M       So        Ed      Po1      Po2       LF      M.F
## [1,] 69.42028 66.94019 -7.611451 132.5061 129.8085 27.21254 130.8437
##           Pop       NW        U1       U2   Wealth     Ineq      Prob
## [1,] 36.54482 58.45756 -18.52881 20.62032 27.82379 49.67512 -117.5631
##           Time
## [1,] -15.69815
```

Answer to Question 9.1

In Homework 3, I created a regression model using only the following predictors: Ed, Ineq, M, Prob, U2, Po1. The R2 is 0.76 and adjusted R2 is 0.73. The model I got from using first seven principal components has R2 of 0.69.
This shows using PCA for linear regression in this problem doesn't really help increasing R2 of the model.

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).

Regression tree model

```
rm(list=ls())
library(rpart)
library(caret)
crime_df = read.table('uscrime.txt', header=TRUE)
set.seed(0)
caret_control = trainControl(method='repeatedcv', number=3, repeats = 2)
caret_cv = train(Crime~., data=crime_df, method='rpart',
                 trControl=caret_control, tuneLength=15)
caret_cv
```

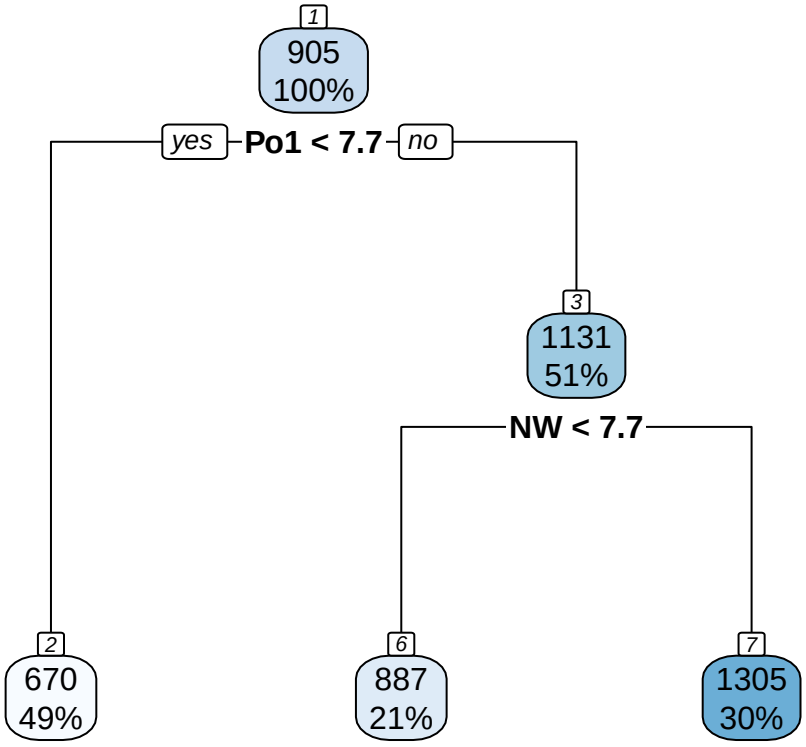
```
## CART
##
## 47 samples
## 15 predictors
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 2 times)
## Summary of sample sizes: 31, 32, 31, 32, 31, 31, ...
## Resampling results across tuning parameters:
##
##   cp          RMSE      Rsquared   MAE
##   0.00000000  348.3207  0.2920491  271.6141
##   0.02592592  348.3207  0.2920491  271.6141
##   0.05185185  348.3207  0.2920491  271.6141
##   0.07777777  353.9545  0.2689376  278.4507
##   0.10370369  358.8027  0.2442745  281.5299
##   0.12962962  368.0270  0.2589726  291.5660
##   0.15555554  368.0270  0.2589726  291.5660
##   0.18148147  368.0270  0.2589726  291.5660
##   0.20740739  368.0270  0.2589726  291.5660
##   0.23333331  368.0270  0.2589726  291.5660
##   0.25925924  368.0270  0.2589726  291.5660
##   0.28518516  368.0270  0.2589726  291.5660
##   0.31111108  368.0270  0.2589726  291.5660
##   0.33703701  384.8229  0.2176655  306.2172
##   0.36296293  402.1866  0.1459624  326.0105
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.05185185.
```

```
caret_cv[["bestTune"]][["cp"]]
```

```
## [1] 0.05185185
```

One of the parameter for rpart model is cp. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted. It means if cp is set to be close to 0 it will try to overfit the data. So that is the reason why I did k fold cross validation to select best cp.

```
best_cp = caret_cv[["bestTune"]][["cp"]]
model = rpart(Crime~., data=crime_df, cp=best_cp)
library(rpart.plot)
rpart.plot(model, type=2, nn=TRUE)
```



Answer to Question 10.1.a

Model interpretation Based on tree model picture above, we can see that Po1, Pop, and NW are the important variables for predicting crime. First level of tree will ask if the Po1 is smaller than 7.7, if yes go to the left branch and else go to right branch. And same method is applied for all brances below.
Insde the box in the leaf, the predicted crime value and percentage of observation is shown.

Random Forest Model

Because random forest model doesn't overfit data, we don't have to do cross validation. Out of curiosity I will try making several random forest models with different number of trees.

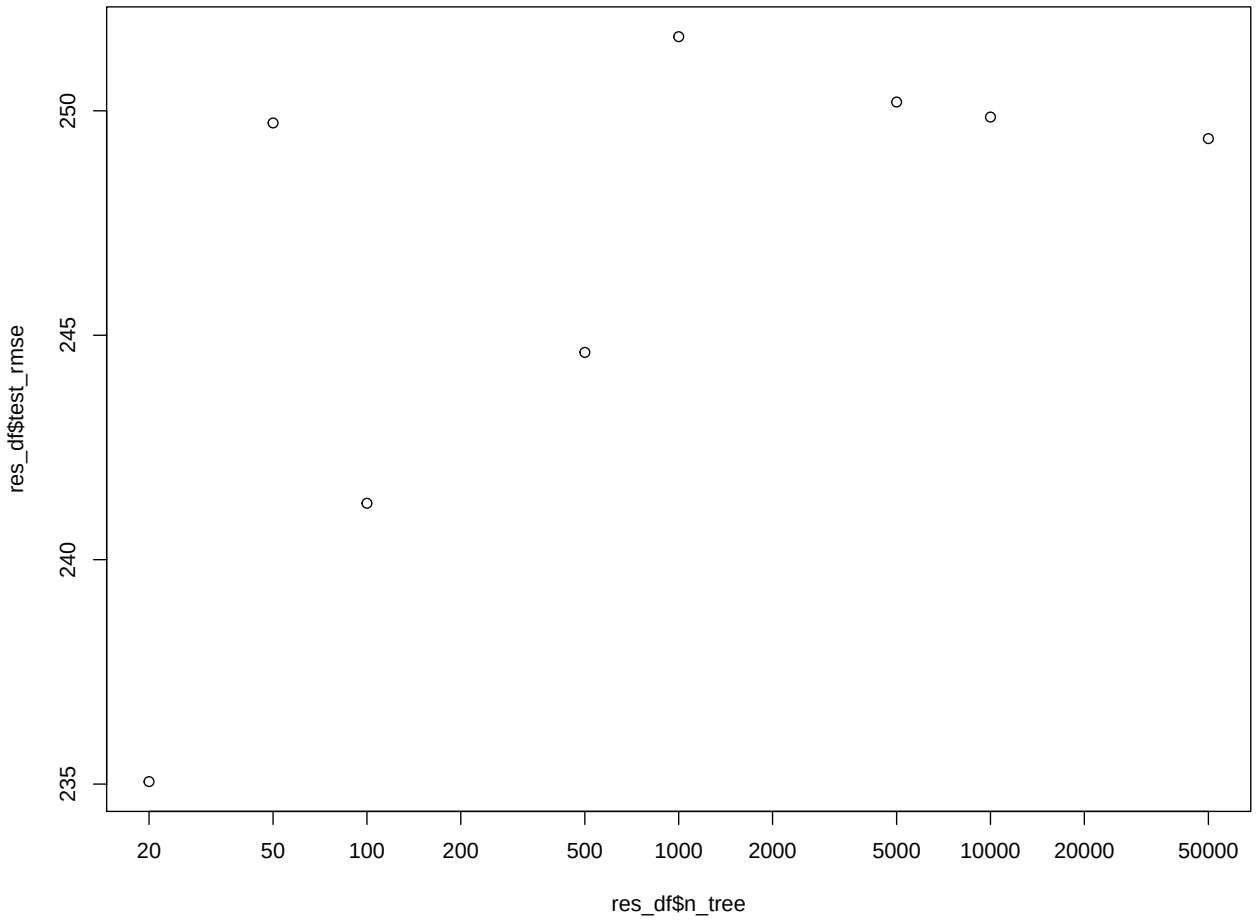
```
rm(list=ls())
crime_df = read.table('uscrime.txt', header=TRUE)
library(caTools)
set.seed(0)
sample = sample.split(crime_df, SplitRatio=0.8)
train = subset(crime_df, sample == TRUE)
test  = subset(crime_df, sample == FALSE)
library(randomForest)
```

```r
library(Metrics)
n_tree_list = c(20, 50, 100, 500, 1000, 5000, 10000, 50000)
model_list = c()
res_df = NULL
for (n_tree in n_tree_list){
  model = randomForest(Crime~., data=train, ntree=n_tree)
  y_train_hat = model$predicted
  y_train = train$Crime
  y_test_hat = predict(model, test)
  y_test = test$Crime
  train_rmse = rmse(y_train, y_train_hat)
  test_rmse = rmse(y_test, y_test_hat)
  model_list = c(model_list, model)
  res_df = rbind(res_df,
                 data.frame(n_tree, train_rmse, test_rmse))
}
res_df
```

```
##   n_tree train_rmse test_rmse
## 1     20   334.3121  235.0551
## 2     50   343.9744  249.7297
## 3    100   315.1818  241.2564
## 4    500   324.8410  244.6179
## 5   1000   319.1512  251.6518
## 6   5000   320.8800  250.1959
## 7  10000   320.9252  249.8611
## 8  50000   319.8543  249.3805
```

```r
best_res = res_df[which.min(res_df$test_rmse),]
```

```r
plot(x=res_df$n_tree, y=res_df$test_rmse, log='x')
```



Answer to Question 10.1.b

From graph above, we can conclude that very big random forest model with number of tree 50000 doesn't produce very different result with random forest with n_tree=50. We can infer that number of tree doesn't cause the model to overtfit. With this conclusion I will use the number of tree that produce lowest test error:

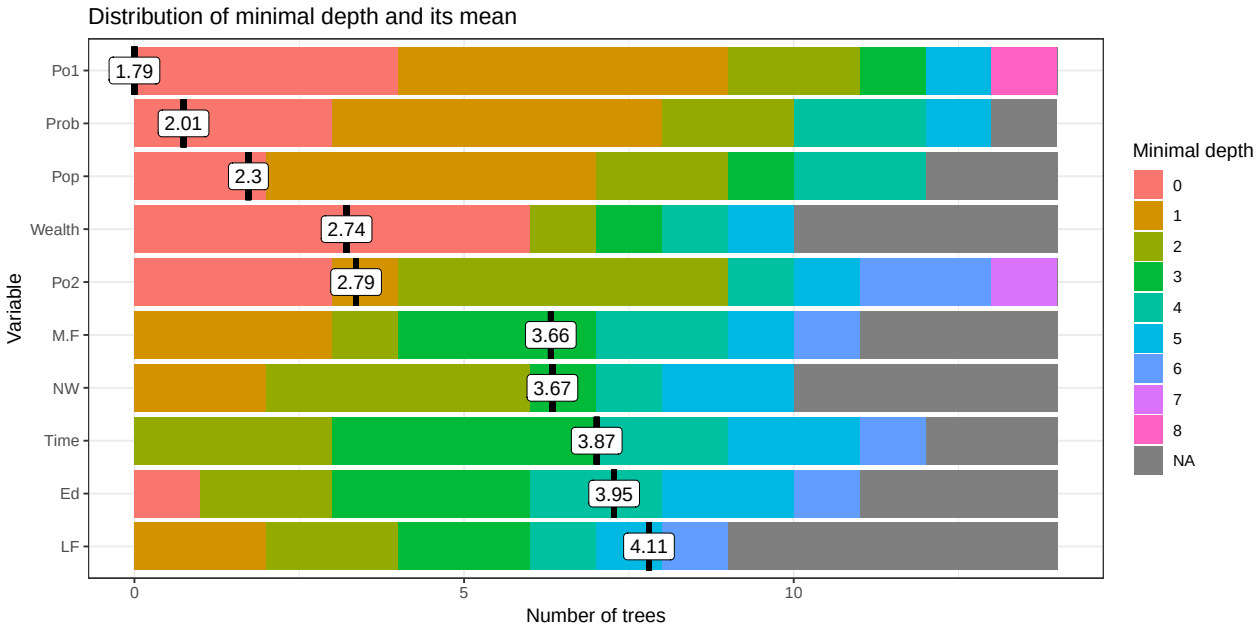
```
best_res
```

```
##   n_tree train_rmse test_rmse
## 1     20   334.3121  235.0551
```

```
model = randomForest(Crime~., data=train, ntree=best_res$n_tree)
y_train_hat = model$predicted
y_train = train$Crime
y_test_hat = predict(model, test)
y_test = test$Crime
train_rmse = rmse(y_train, y_train_hat)
test_rmse = rmse(y_test, y_test_hat)
```

Visualizing random forest model is not easy. So I will try to see the distribution of minimal depth in a random forest model. Attributes/predictors with low minimal depth has bigger importance for predictor than large minimal depth predictor.

```
library(randomForestExplainer)
plot_min_depth_distribution(model)
```


Distribution of minimal depth and its mean
Variable
Po1
Prob
Pop
Wealth
Po2
M.F
NW
Time
Ed
LF
1.79
2.01
2.3
2.74
2.79
3.66
3.67
3.87
3.95
4.11
0
5
10
Number of trees
Minimal depth
0
1
2
3
4
5
6
7
8
NA

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.

Answer
Situation: I want to predict what is my chance to be admitted to Georgia Tech OMSA program.
By collecting data from many people who applied to OMSA program, I will use regression to predict the probability of a person accepted or not. I will need data from person who is accepted and from person who is rejected.
Predictors:
1. Undergraduate GPA
2. Academic speciality
3. TOEFL score
4. GRE score
5. Numbers of reference letter

Question 10.3.1

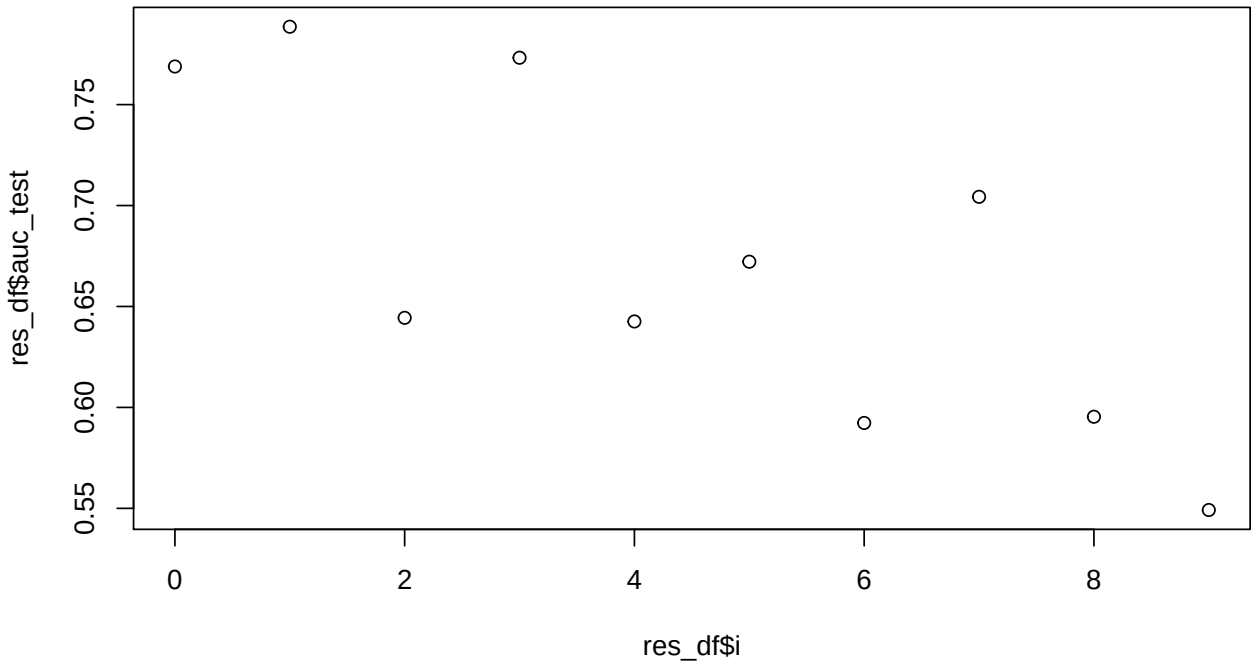
Using the GermanCredit data set, use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic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

```
rm(list=ls())
credit_df = read.table('germancredit.txt', header=FALSE)
credit_df$V21[credit_df$V21==1] = 0 # bad credit
credit_df$V21[credit_df$V21==2] = 1 # good credit
credit_df$V21 = as.factor(credit_df$V21)
str(credit_df)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ V1 : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...
##  $ V2 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ V3 : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...
##  $ V4 : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...
##  $ V5 : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ V6 : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...
##  $ V7 : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...
##  $ V8 : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ V9 : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...
##  $ V10: Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...
##  $ V11: int  4 2 3 4 4 4 4 2 4 2 ...
##  $ V12: Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...
##  $ V13: int  67 22 49 45 53 35 53 35 61 28 ...
##  $ V14: Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ V15: Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...
##  $ V16: int  2 1 1 1 2 1 1 1 1 2 ...
##  $ V17: Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...
##  $ V18: int  1 1 2 2 2 2 1 1 1 1 ...
##  $ V19: Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
##  $ V20: Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
##  $ V21: Factor w/ 2 levels "0","1": 1 2 1 1 2 1 1 1 1 2 ...
```

```r
library(caTools)
library(pROC)
set.seed(0)
sample = sample.split(credit_df, SplitRatio=0.8)
train = subset(credit_df, sample == TRUE)
test = subset(credit_df, sample == FALSE)
res_df = NULL
i = 0
while (i < 10){
  if (i==0){
    rand_col_list = colnames(credit_df)
  }
  else{
    rand_col_list = c(sample(colnames(credit_df)[-21], sample(1:10, 1)), 'V21')
  }
  rand_col_str = paste(rand_col_list, collapse=' ')
  train_filtered = train[rand_col_list]
  test_filtered = test[rand_col_list]
  model = glm(V21~., data=train_filtered, family=binomial(link='logit'))
  train_filtered$prob = predict(model, train_filtered, type='response')
  test_filtered$prob = predict(model, test_filtered, type='response')
  roc_train = roc(V21 ~ prob, data=train_filtered)
  roc_test = roc(V21 ~ prob, data=test_filtered)
  auc_train = roc_train$auc
  auc_test = roc_test$auc
  res_df = rbind(res_df,
  data.frame(i, rand_col_str, auc_train, auc_test))
  i = i + 1
}
# res_df
best_res = res_df[which.max(res_df$auc_test), ]
```

```r
plot(res_df$i, res_df$auc_test)
```



Answer to Question 10.3.1

At first I will set the threshold to be 0.5.

```
best_feature_comb = strsplit(as.vector(best_res$rand_col_str), ' ')[[1]]
train_filtered = train[best_feature_comb]
test_filtered = test[best_feature_comb]
model = glm(V21~., data=train_filtered, family=binomial(link='logit'))
train_filtered$prob = predict(model, train_filtered, type='response')
test_filtered$prob = predict(model, test_filtered, type='response')
train_filtered$pred[train_filtered$prob < 0.5] = 0 #bad credit
train_filtered$pred[train_filtered$prob >= 0.5] = 1 #good credit
test_filtered$pred[test_filtered$prob < 0.5] = 0 #bad credit
test_filtered$pred[test_filtered$prob >= 0.5] = 1 #good credit
roc_train = roc(V21 ~ prob, data=train_filtered)
roc_test = roc(V21 ~ prob, data=test_filtered)
auc_train = roc_train$auc
auc_test = roc_test$auc
auc_train
```

```
## Area under the curve: 0.7904
```

```
auc_test
```

```
## Area under the curve: 0.7886
```

```
conf_matrix = table(test_filtered$V21, test_filtered$pred)
print('Row is actual value, column is predicted')
```

```
## [1] "Row is actual value, column is predicted"
```

```
conf_matrix
```

```
##
##       0   1
```

```
##   0 150  23
##   1  35  31
```

```
tp = conf_matrix[2, 2] #good customer classified as good
tn = conf_matrix[1, 1] #bad customer classified as bad
fp = conf_matrix[1, 2] #bad customer classified as good
fn = conf_matrix[2, 1] #good customer classified as bad
```

Answer to Question 10.3.2

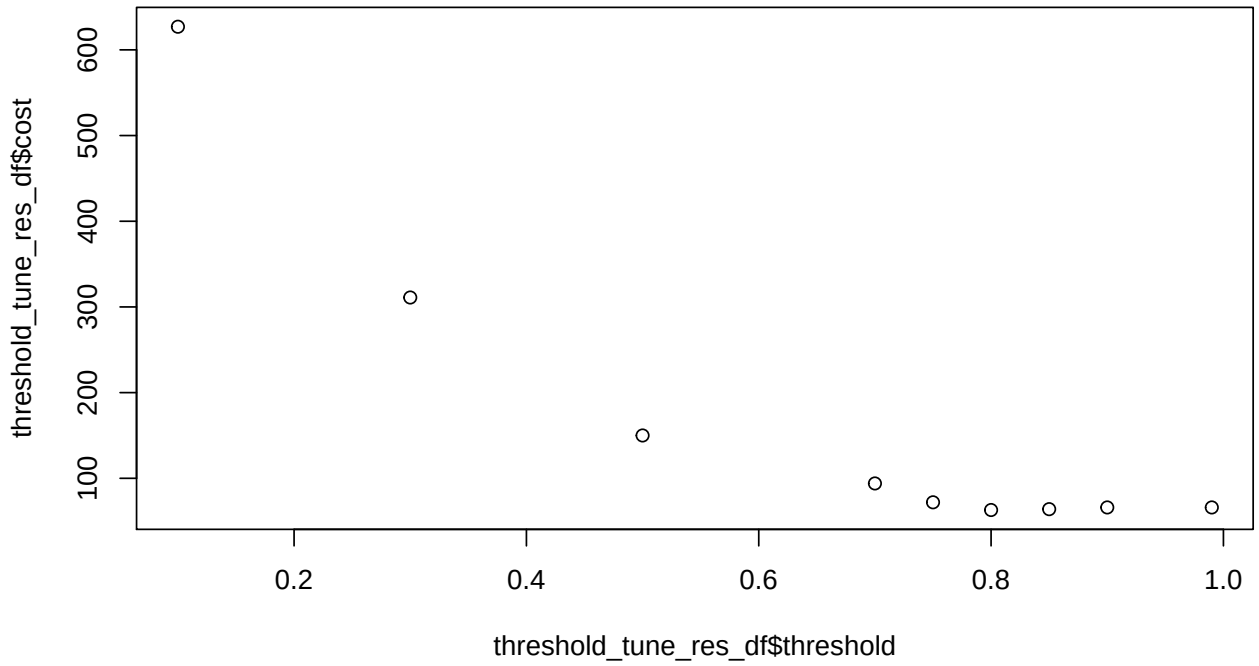
Because the cost of incorrectly classifying bad customer as good is 5 times the cost of classifying good customer as bad, the we want to minimize the following value: 5 * False positive + False negative.

```
library(caret)
threshold_list = c(0.1, 0.3, 0.5, 0.7, 0.75, 0.8, 0.85, 0.9, 0.99)
threshold_tune_res_df = NULL
for (threshold in threshold_list){
  train_filtered$pred[train_filtered$prob < threshold] = 0 #bad credit
  train_filtered$pred[train_filtered$prob >= threshold] = 1 #good credit
  test_filtered$pred[test_filtered$prob < threshold] = 0 #bad credit
  test_filtered$pred[test_filtered$prob >= threshold] = 1 #good credit
  train_filtered$pred = as.factor(train_filtered$pred)
  test_filtered$pred = as.factor(test_filtered$pred)
  conf_matrix = confusionMatrix(test_filtered$pred, test_filtered$V21)[['table']]
  tp = conf_matrix[2, 2] #good customer classified as good
  tn = conf_matrix[1, 1] #bad customer classified as bad
  fp = conf_matrix[2, 1] #bad customer classified as good
  fn = conf_matrix[1, 2] #good customer classified as bad
  cost = 5 * fp + fn
  threshold_tune_res_df = rbind(threshold_tune_res_df,
                                data.frame(threshold, cost, tp, tn, fp, fn))
}
threshold_tune_res_df
```

```
##   threshold cost tp  tn  fp fn
## 1      0.10  627 64  48 125  2
## 2      0.30  311 50 114  59 16
## 3      0.50  150 31 150  23 35
## 4      0.70   94  7 166   7 59
## 5      0.75   72  4 171   2 62
## 6      0.80   63  3 173   0 63
## 7      0.85   64  2 173   0 64
## 8      0.90   66  0 173   0 66
## 9      0.99   66  0 173   0 66
```

```
best_res = threshold_tune_res_df[which.min(threshold_tune_res_df$cost), ]
```

```
plot(x=threshold_tune_res_df$threshold, y=threshold_tune_res_df$cost)
```




```
best_res
```

```
##   threshold cost tp  tn fp fn
## 6       0.8   63  3 173  0 63
```

```
sprintf('Best threshold is %.2f', best_res$threshold)
```

```
## [1] "Best threshold is 0.80"
```
