HomeWork7

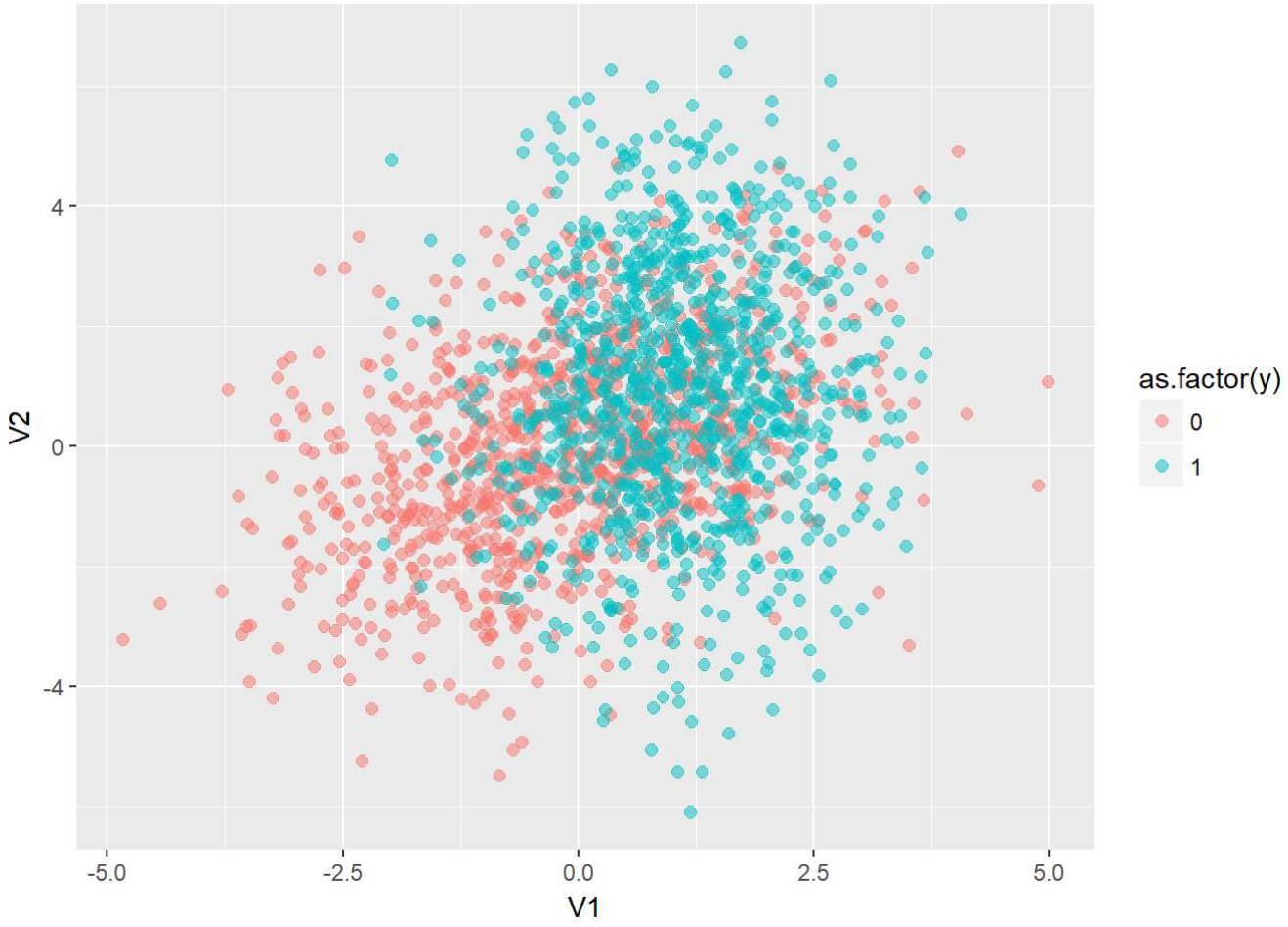
Ivan Ojiambo

November 9, 2017

a

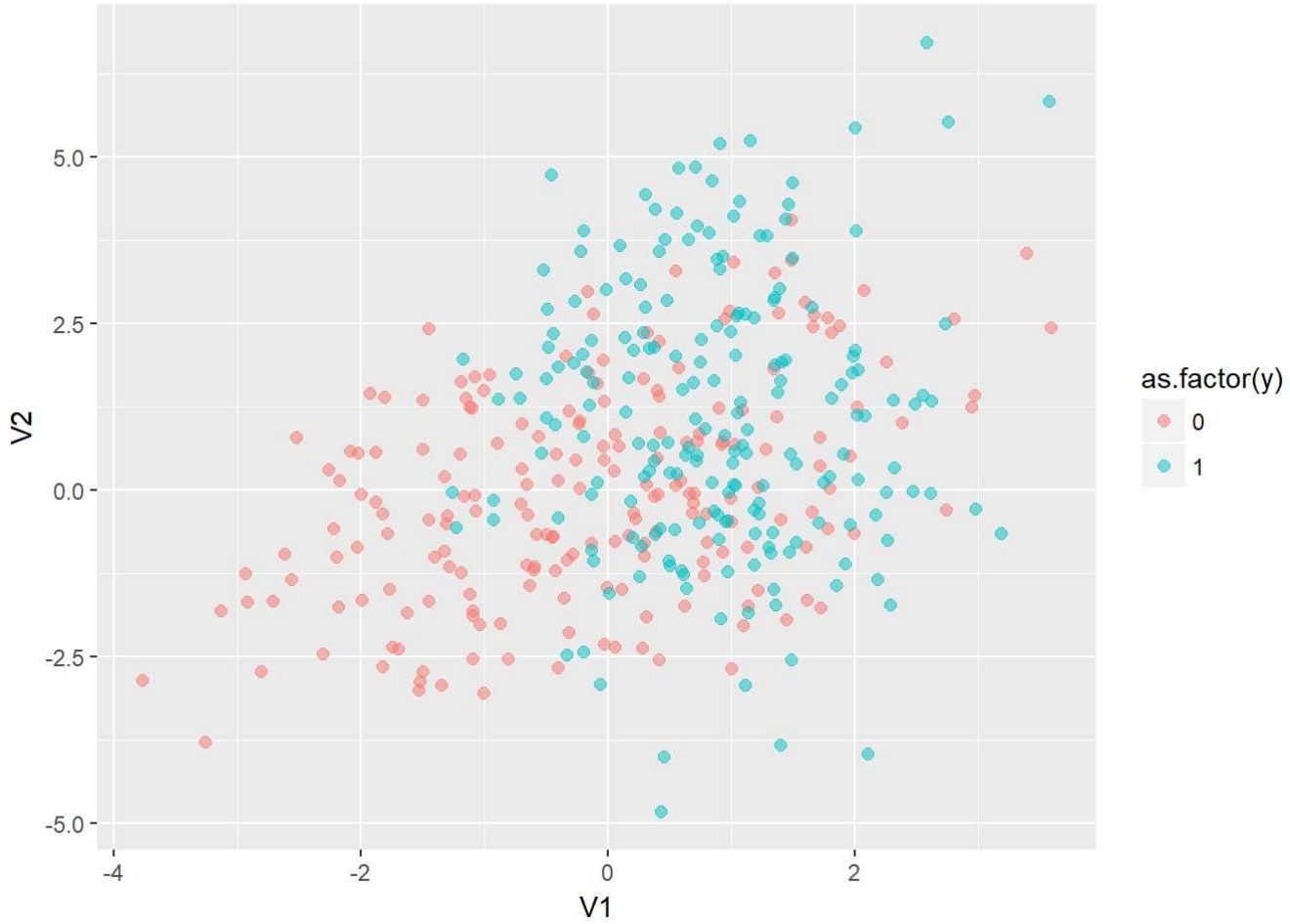
ggplot for train data

```
plot1
```



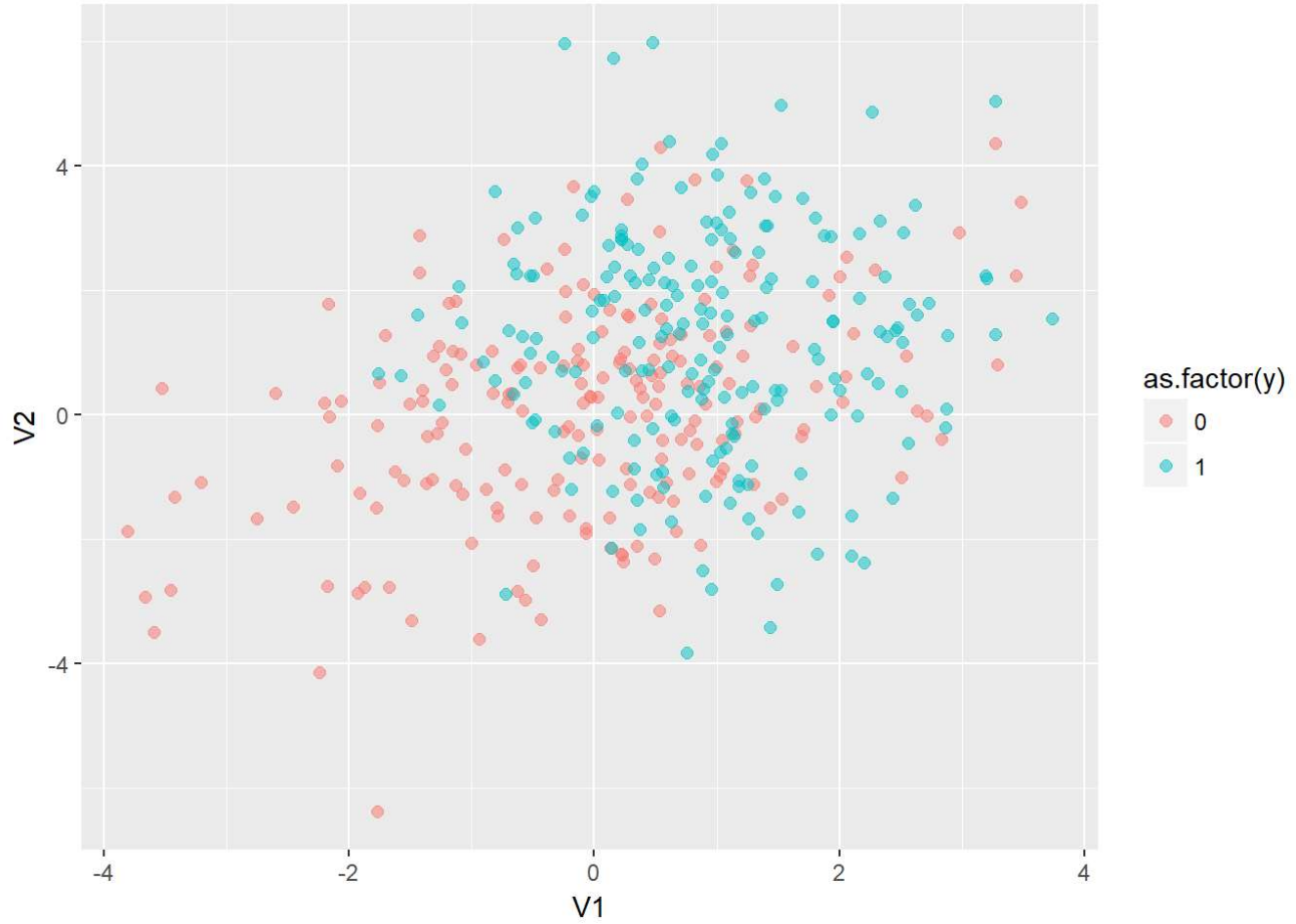
ggplot for test data

```
plot2
```



ggplot for validation data

```
plot3
```



b) possible cutoffs (thresholds) from extracted probabilities. How can you interpret these values?

```
cutoffs <- sort(unique(probs_val))
cutoffs_test_data <- sort(unique(probs_test))
```

c) Compute true positive rate (TPR) and false positive rate (FPR) for all possible cutoffs using validation data:

```
tpr  <- sapply(cutoffs, function(cut) sum((probs_val >= cut)  == TRUE &
                                          val_data$y == 1)/(sum(val_data$y == 1)))
fpr  <- sapply(cutoffs, function(cut) sum((probs_val >= cut) == TRUE &
                                          val_data$y == 0)/(sum(val_data$y == 0)))
stats <- data.frame(cutoffs, tpr, fpr)
stats[6,]
```

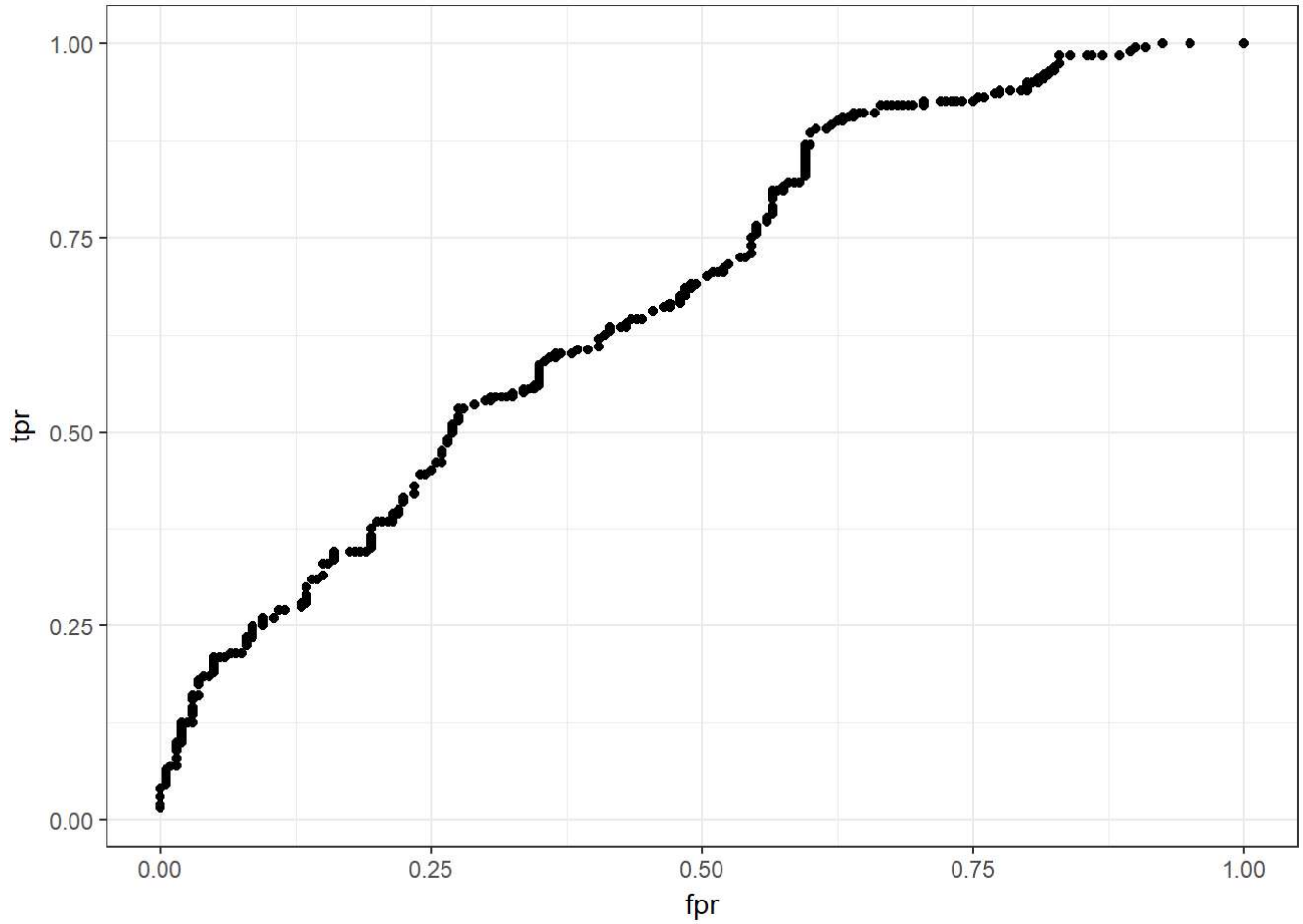
```
##   cutoffs  tpr   fpr
## 6   0.012 0.99 0.895
```

with a threshold of 0.012, we see that we get a true positive rate of 99% and false positive rate of 89.5%.

This is not a good a threshold point, even though it correctly predicated a high number (99%) of positives , It also falsly predicated a high number(89.5%) of positives as negatives.

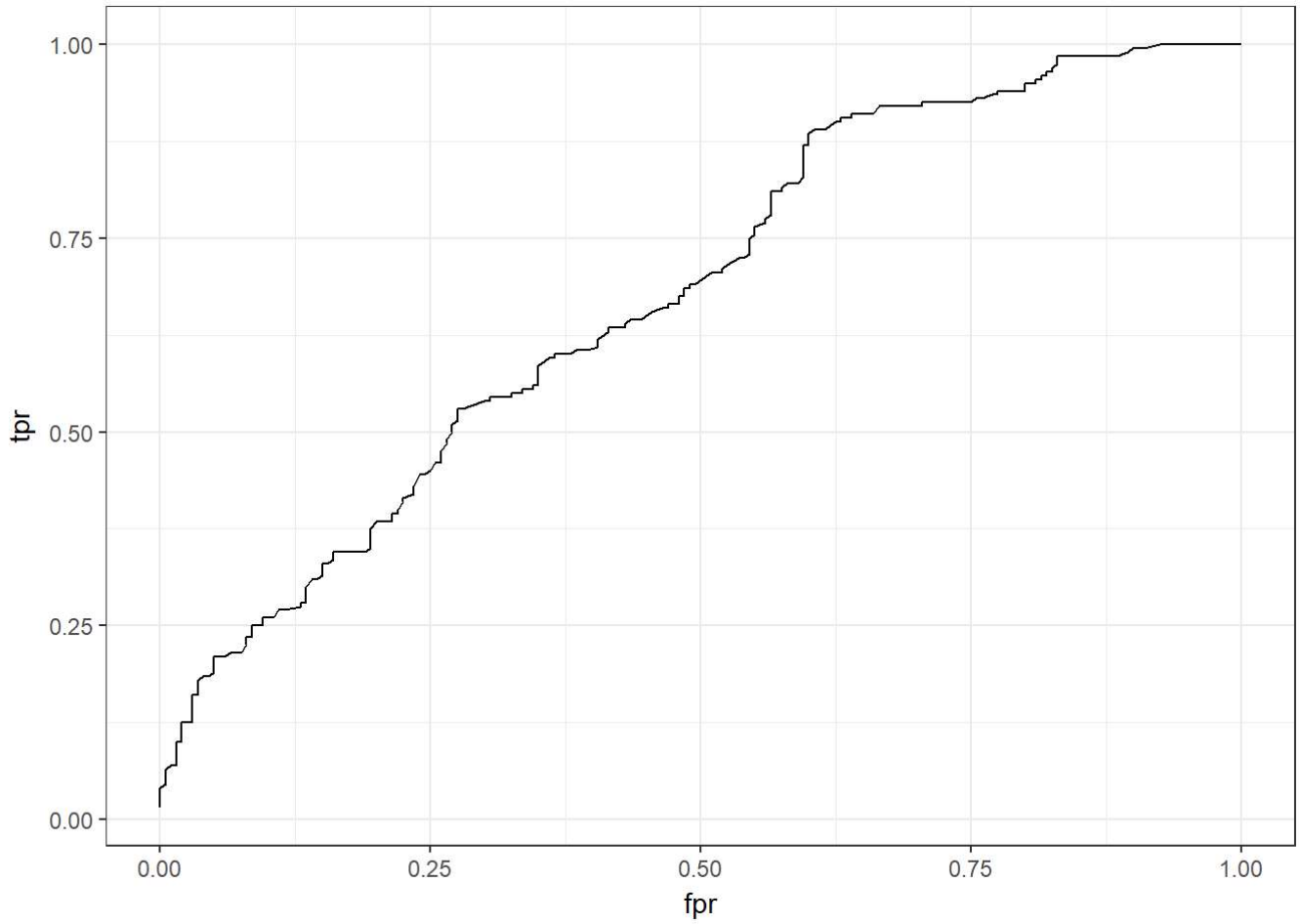
(d) Use ggplot to make a figure where TPR is on y-axis and FPR on x-axis.

```
roc_plot <- ggplot(stats, aes(x = fpr, y = tpr)) + geom_point() + theme_bw()
roc_plot
```



using geom path

```
ggplot(stats, aes(x = fpr, y = tpr)) + geom_path() + theme_bw()
```



Is the performance of the RF classifier better than random guess? How ROC would look for the random classifier?

```
AUC <- function(TPR, FPR){
  dTPR <- c(diff(TPR), 0)
  dFPR <- c(diff(FPR), 0)
  sum(dFPR * TPR) + sum(dFPR * dTPR)
}

Area_under_curve <- with(stats, AUC(stats$tpr, stats$fpr))
abs(Area_under_curve)
```

```
## [1] 0.674325
```

Yes the random forest classifier gives a slightly better performance compared to random classifier ie using random classifier we yield area under curve of 0.674 compared to one of random classifier which is 0.5

e

```
rf_dt <- data.frame(class = predict(rf_fit, newdata = val_data, type = 'raw'),
           probs = probs_val)

rf_df <- rf_dt[with(rf_dt, order(probs)),]

rf_df <- rf_df[rf_df$class == "X1",]
rf_df <- rf_df[with(rf_df, order(probs)),]
cutoff <- rf_df$probs[1]

print(paste("The cutoff used by random forest is ", cutoff))
```

```
## [1] "The cutoff used by random forest is  0.508"
```

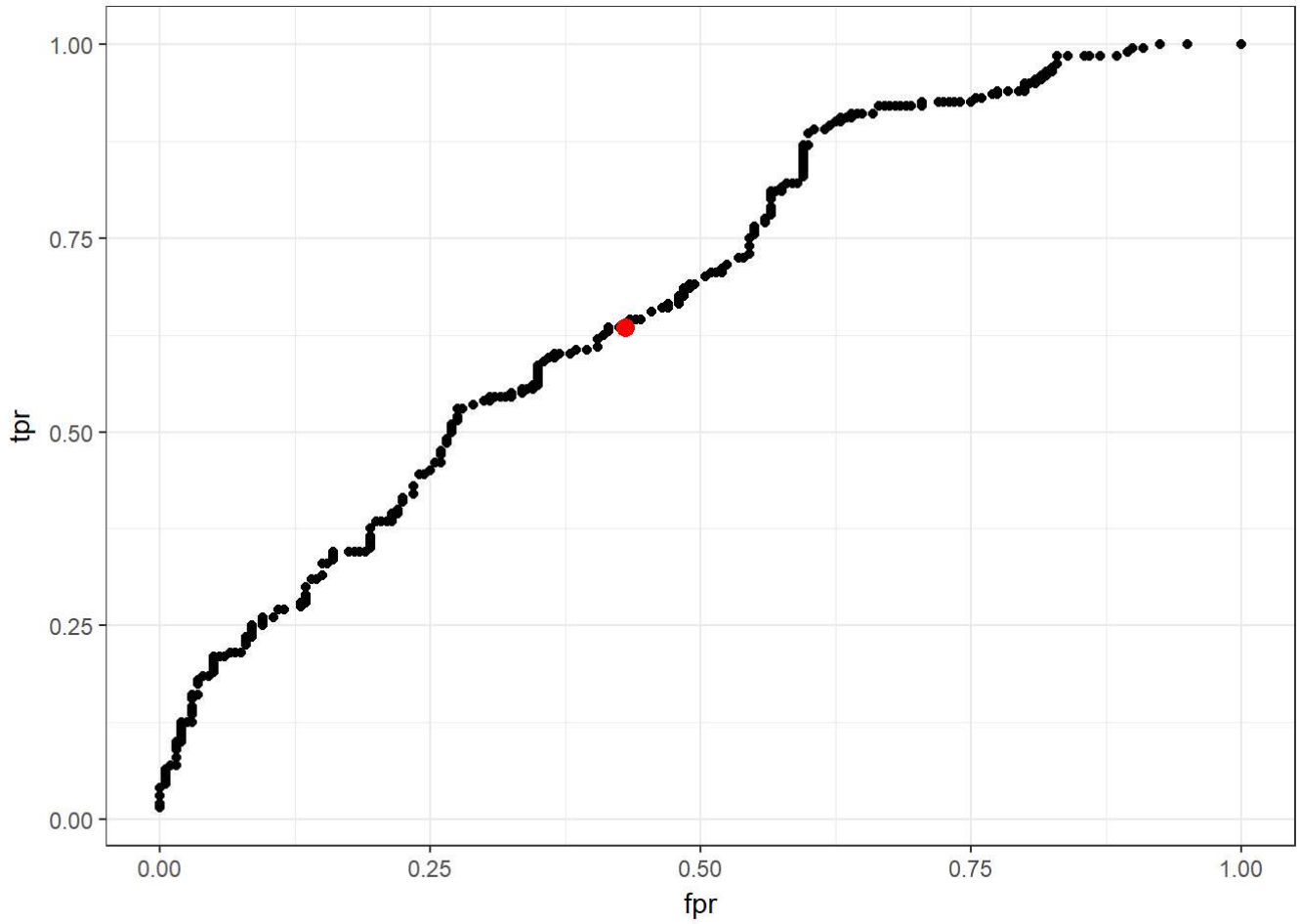
```
TP  <- sum(probs_val >= 0.5  & val_data$y == 1)
TN  <- sum(probs_val <0.5 &  val_data$y == 0)

accuracy <- (TP +TN)/nrow(rf_df)
print(paste("Accuracy when cutoff is 0.5  = ", accuracy))
```

```
## [1] "Accuracy when cutoff is 0.5  =  1.13679245283019"
```

f

The red point on the ggplot reprensent the cutoff point on the ROC curve

```
roc_plot
```



##g

For each of the cutoffs in stats compute number of false positives (FP) and false negatives (FN) on a validation data

```
fp_count <- sapply(cutoffs, function(cut) sum((probs_val >= cut) == TRUE & val_data$y == 0))
fn_count <- sapply(cutoffs, function(cut) sum((probs_val >= cut) == FALSE & val_data$y == 1))
acc <- sapply(cutoffs, function(cut) (sum((probs_val >= cut) == FALSE & val_data$y == 0) +
                                      sum((probs_val >= cut) == TRUE & val_data$y == 1))/leng
th(probs_val))

stats <- data.frame(stats, fp_count, fn_count, acc)
```

h

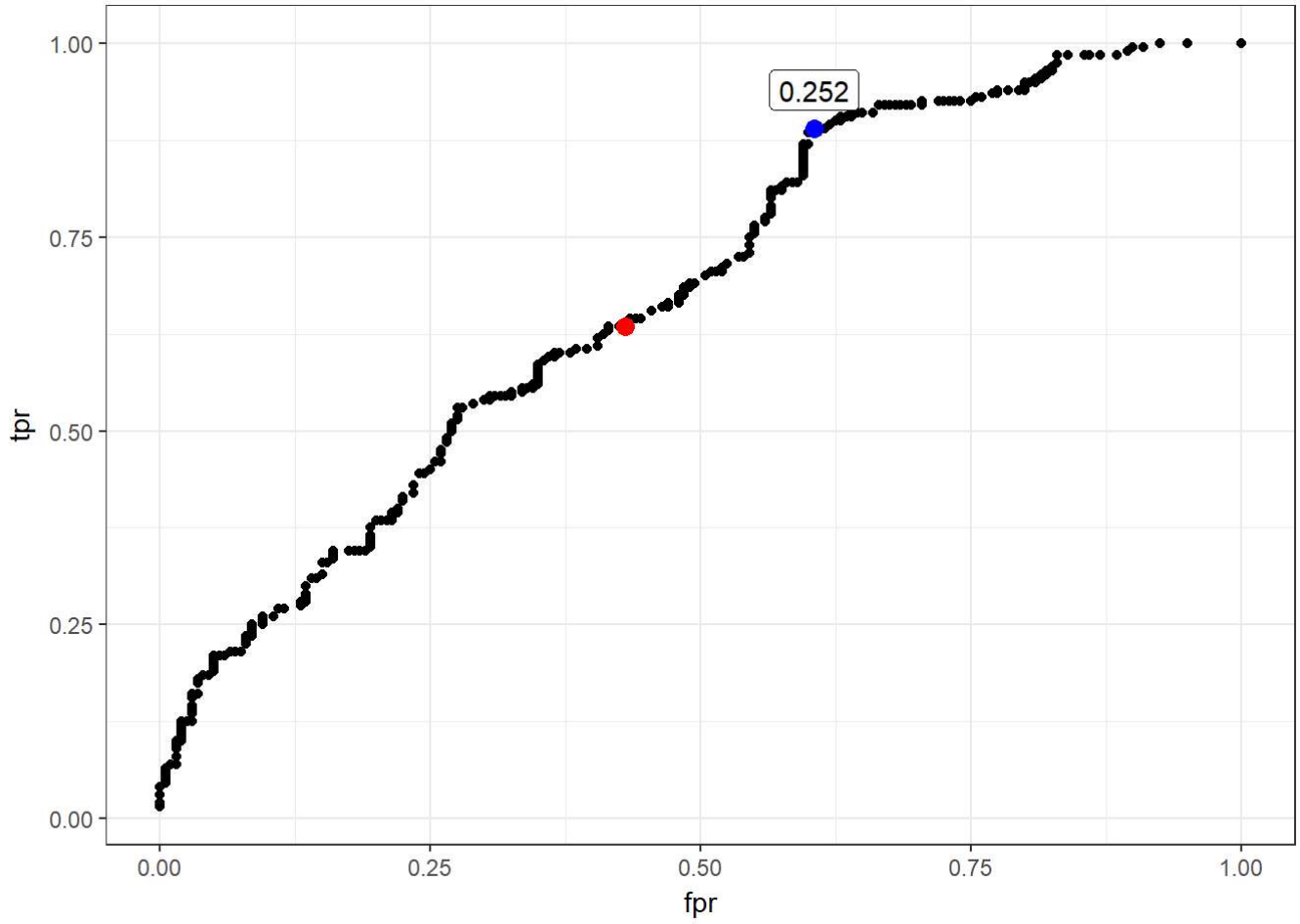
```
stats$cost <- stats$fp_count + stats$fn_count
stats[which.min(stats$cost),]
```

```
##    cutoffs  tpr   fpr fp_count fn_count    acc cost
## 59   0.252 0.89 0.605      121       22 0.6425  143
```

```
roc_plot <- roc_plot +
  geom_point(data=stats[which.min(stats$cost),], aes(x=fpr, y=tpr), colour="blue", size=3) +
  geom_label(data = stats[which.min(stats$cost),], label =
stats$cutoffs[which.min(stats$cost)], nudge_y = 0.05)
```

##most optimal cutoff when Cost of FP equals to cost of FN

```
roc_plot
```



The optimal cutoff point when FP equals to cost of FN is 0.252 and yields an accuracy of 64.25% . TPR = 0.89 FPs = 121 FPR = 0.60 FNs = 22

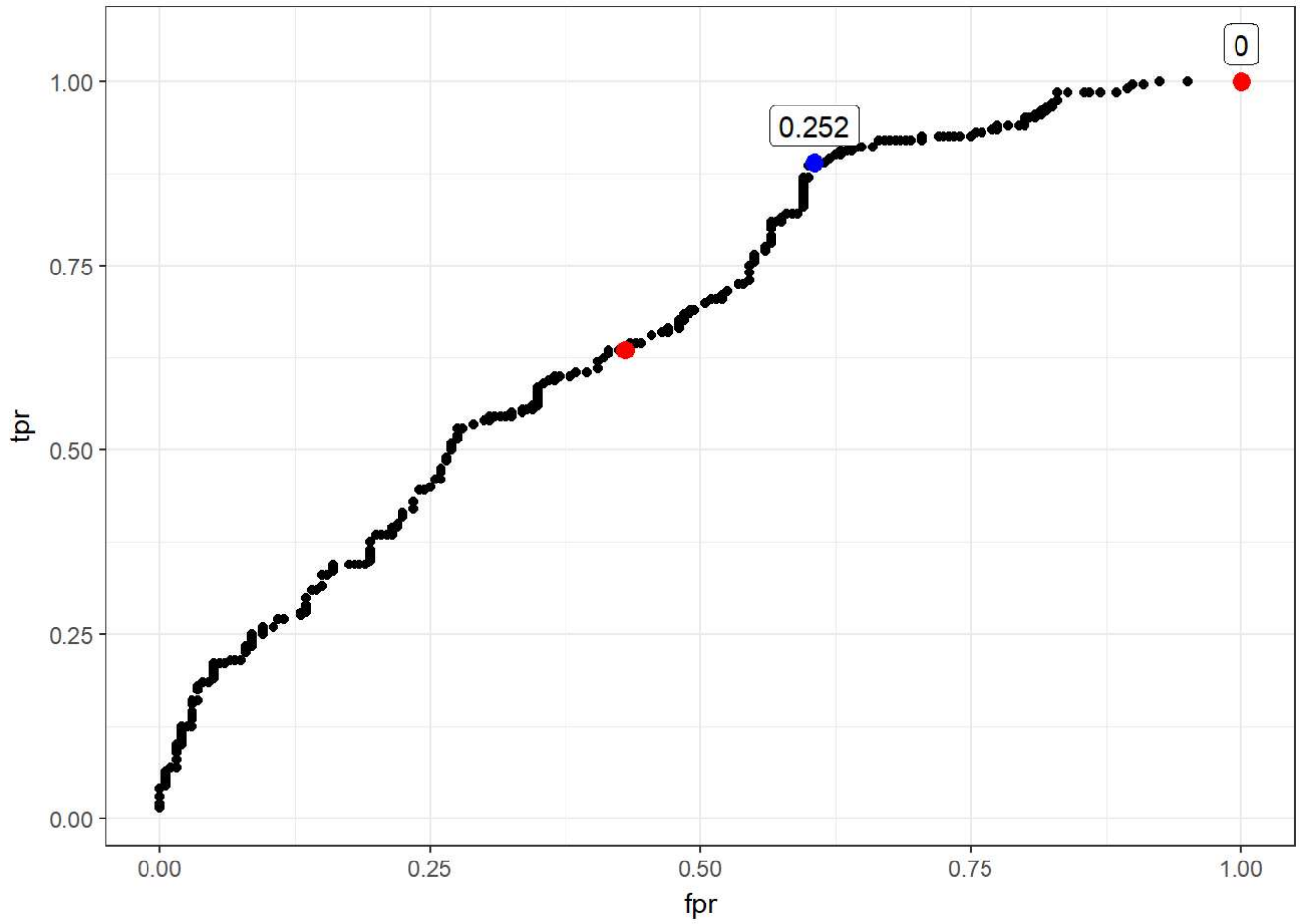
```
stats$cost <-  stats$fn_count*3 + stats$fn_count
stats[which.min(stats$cost),]
```

```
##   cutoffs tpr fpr fp_count fn_count acc cost
## 1       0   1   1      200        0 0.5    0
```

```
roc_plot <- roc_plot +
  geom_point(data=stats[which.min(stats$cost),], aes(x=fpr, y=tpr), colour="red", size=3) +
  geom_label(data = stats[which.min(stats$cost),], label =
stats$cutoffs[which.min(stats$cost)], nudge_y = 0.05)
```

most optimal cutoff when Cost of FP is 3 times larger than cost of FN

```
roc_plot
```



The optimal cutoff point when FP is 3 times larger than cost of FN is 0 and yields an accuracy of 50% . TPR = 1 FPs = 200 FPR = 1 FNs = 0

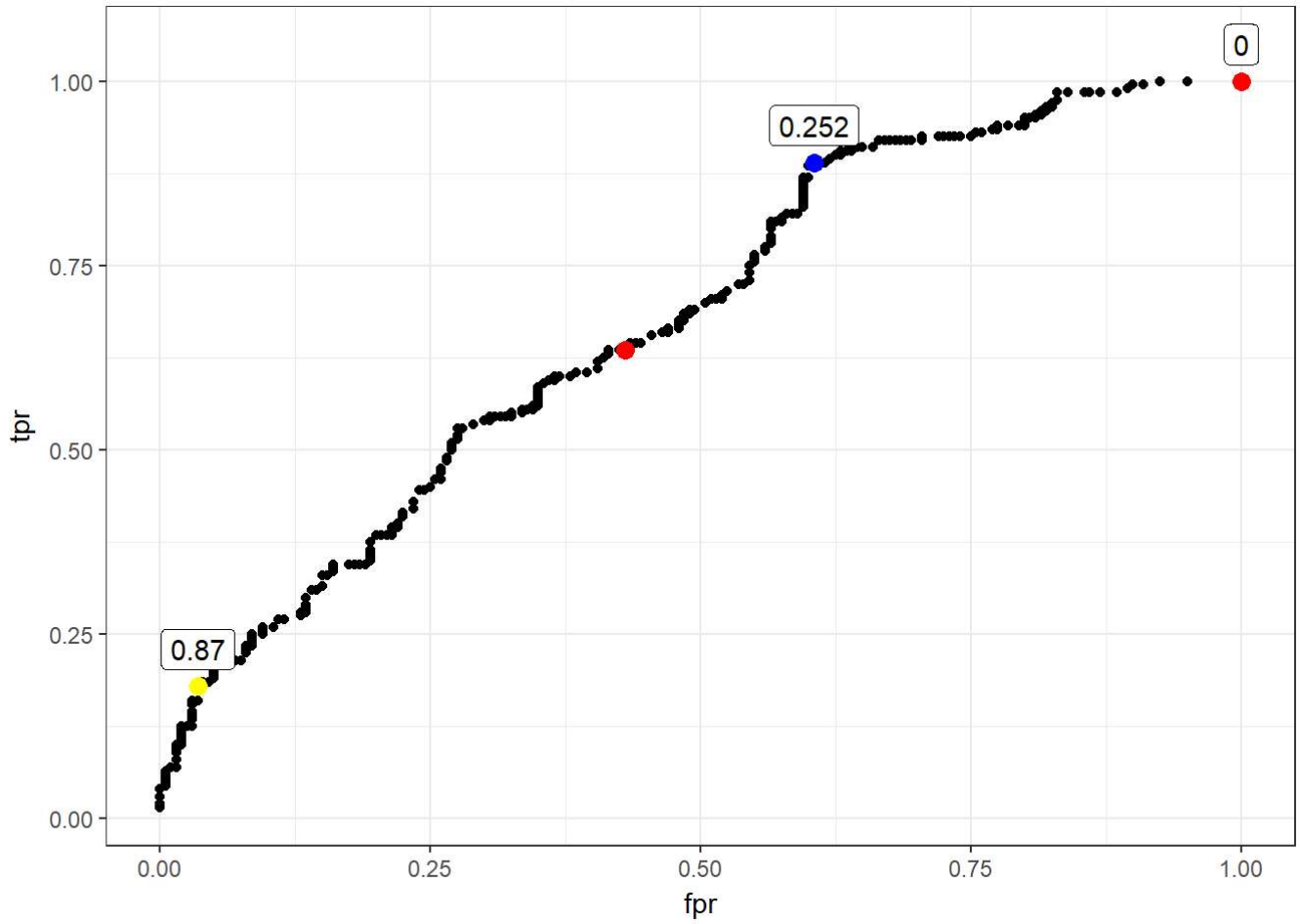
```
stats$cost <-  stats$fp_count*3 + stats$fn_count
stats[which.min(stats$cost),]
```

```
##     cutoffs  tpr   fpr fp_count fn_count    acc cost
## 227    0.87 0.18 0.035        7      164 0.5725  185
```

```
roc_plot <- roc_plot +
  geom_point(data=stats[which.min(stats$cost),], aes(x=fpr, y=tpr), colour="yellow", size=3)
+
  geom_label(data = stats[which.min(stats$cost),], label =
stats$cutoffs[which.min(stats$cost)], nudge_y = 0.05)
```

Scenario: Cost of FN is 3 times larger than cost of FP

```
roc_plot
```



cuttoff = 0.87 accuracy = 57.25% TPR = 0.18 FPs = 7 FPR = 0.035 FNs = 164

Which of the points on the ROC (cutoffs) corresponds to the maximal possible accuracy of our random forest classifier? Store this cutoff.

- cutt off of 0.252 corresponds to the maximal possible accuracy of random forest

```
optimal_cutoff <- 0.252
```

i

Compute accuracy of the random forest classifier based on default cutoff and the cutoff from task (h). Compare them and throughly explain the reasons for the difference that you observe.

```
tpr  <- sapply(cutoffs_test_data, function(cut) sum((probs_test >= cut)  == TRUE &
test_data$y == 1)/(sum(test_data$y == 1)))
fpr  <- sapply(cutoffs_test_data, function(cut) sum((probs_test >= cut)  == TRUE &
test_data$y == 0)/(sum(test_data$y == 0)))
stats <- data.frame(cutoffs_test_data, tpr, fpr)

cutoffs <- sort(unique(probs_test))

#computing the cut used inrandom forest
rf_dt <- data.frame(class = predict(rf_fit, newdata = test_data, type = 'raw'),
           probs = probs_test)
rf_df <- rf_dt[with(rf_dt, order(probs)),]

rf_df <- rf_df[rf_df$class == "X1",]
rf_df <- rf_df[with(rf_df, order(probs)),]
cutoff <- rf_df$probs[1]

accuracy_default <- (sum((probs_test >= cutoff) == FALSE & test_data$y == 0) +
            sum((probs_test >= cutoff) == TRUE & test_data$y == 1))/length(probs_test)

accuracy2 <- (sum((probs_test >= optimal_cutoff) == FALSE & test_data$y == 0) +
            sum((probs_test >= optimal_cutoff) == TRUE & test_data$y ==
1))/length(probs_test)

print(paste("Accuracy for default cutoff is", accuracy_default))
```

```
## [1] "Accuracy for default cutoff is 0.65"
```

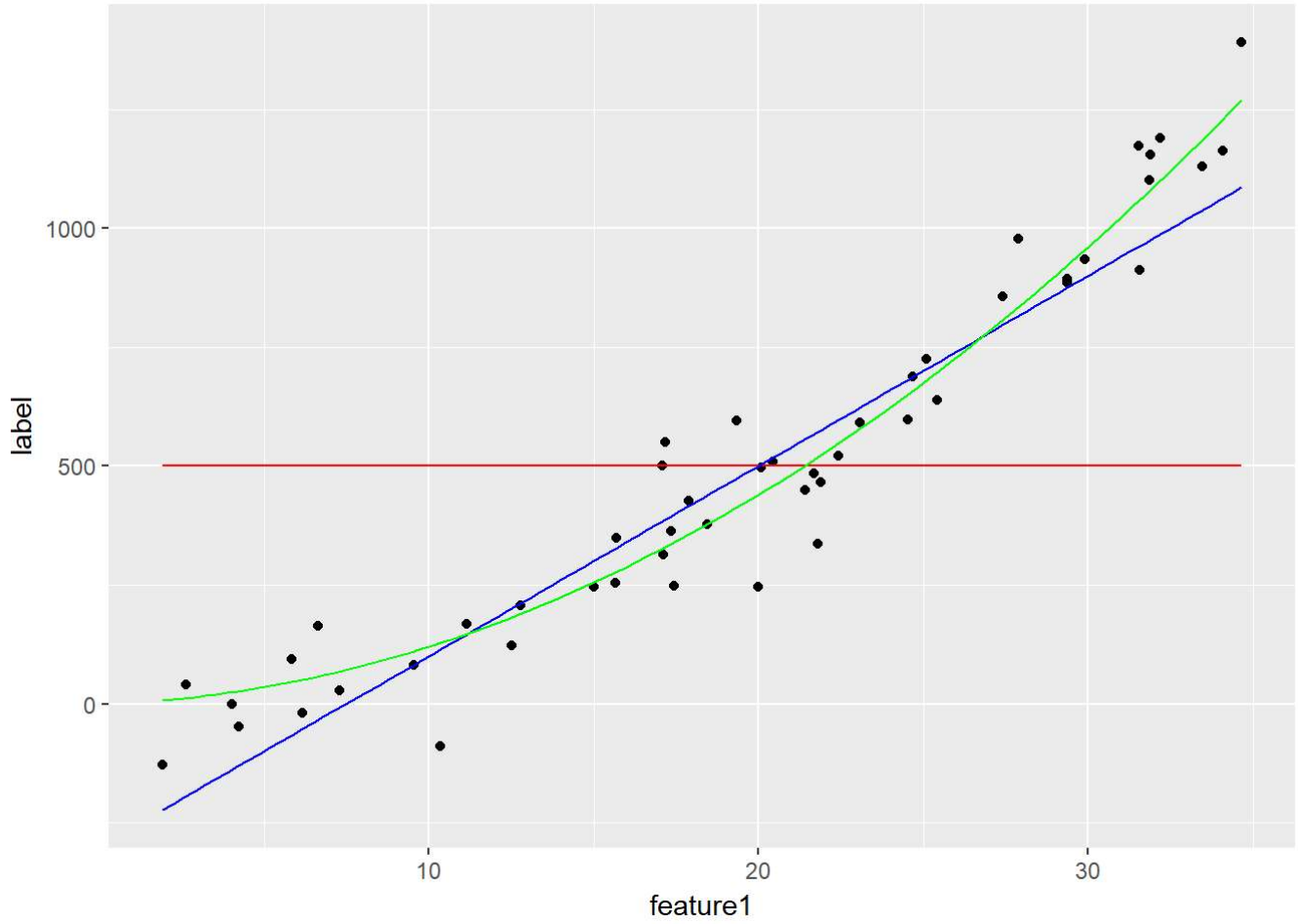
```
print(paste("Accuracy for cutoff in taskh is", accuracy2))
```

```
## [1] "Accuracy for cutoff in taskh is 0.63"
```

There was no significant difference .

Exe2

```
p
```



a Which function seems to be the best fit for the data visually?

function f2 seems to be the best fit because its balanced ie points are many points are lieing on both sides of the line.

b

```
mse_f1 <- sum((df$label - f1(df$feature1))^2)/nrow(df)
mse_f2 <- sum((df$label - f2(df$feature1))^2)/nrow(df)
mse_f3 <- sum((df$label - f3(df$feature1))^2)/nrow(df)
```

c

```
library(knitr)
table1 <- data.frame("FUNC"=c(paste("f1"), paste("f2")),
                     "MSE" = c(mse_f1, mse_f2),
                     "RMSE"= c(sqrt(mse_f1), sqrt(mse_f2))
                     )
kable(table1)
```

FUNC	MSE	RMSE
f1	153537.7	391.8388

FUNC	MSE	RMSE
f2	15006.6	122.5014

##d ch is the best according to MSE and RMSE? Is it the same as what you found to be the best
Whi function visually?

Regression line f3 has the least RMSE and therefore its the best regression line No, its not the same as I had visualy predicated

e Why do you think RMSE is often preferred to MSE?

The RMSE is prefered because it decribes the variability of the data from the actual mean and its in the same unit as the data. whereas the variance is expressed in squared units which could be hard to compare from the original data

Exe3

a Read in the data, remove the name column and separate the data into training data d_train and test data d_test with equal sizes;

```
df <- read.csv("auto_mpg_cleaned.csv", sep = ";")
df$name = NULL

d_train <- df[1:196,]
d_test <- df[197:392,]
```

b

define the following function to be used in evaluation:

```
MSE = function(model,data) mean((data$mpg-predict(model,newdata=data))^2)
```

c

learn the empty linear model (without any features) and the full linear model (using all features)

```
empty_model = lm(formula = mpg ~ 1,data = d_train)
full_model  = lm(formula = mpg ~ .,data = d_train)
```

d

create a function to report the performance of a model:

```
print(report)
```

```
##                                         formula n_features train_mse
## 1                                             1          1  33.72584
## 2 cyl + disp + hpow + weight + accel + year + orig          7  10.11731
##    test_mse
## 1 154.32421
## 2  49.35137
```

e

start eliminating features from the full model until AIC stops improving.

f

start adding features to the empty model until AIC stops improving, again using function step

g

show the final report table and discuss the results

```
print(report)
```

```
##                                         formula n_features train_mse
## 1                                             1          1  33.72584
## 2 cyl + disp + hpow + weight + accel + year + orig          7  10.11731
## 3         cyl + disp + hpow + weight + year + orig          6  10.12170
## 4                                           year          1  21.75317
## 5                                 year + weight          2  11.14750
## 6                          year + weight + hpow          3  10.58642
## 7                    year + weight + hpow + orig          4  10.37179
##    test_mse
## 1 154.32421
## 2  49.35137
## 3  48.60073
## 4 111.72128
## 5  68.92867
## 6  68.21662
## 7  60.56472
```

- while steping backwards ie decreasing the the feature , after decreasing by one step, AIC stopped improving.
- while for **steping dowwn** AIC stoped improving on the third step.

Which method was better, eliminating or adding, (e) or (f)?

eliminating method was better ie it had the least MSE for both train and test data.

As you saw, sometimes a model with less features can work better than the full model. Why, how can it be?

A model with less can work better than a full a model especially if the few features it has are more correlated to the the test feature being investigated.

h

As a comparison, pick the features that have p-values (Pr(>|t|)) below 0.001

```
##p_value = 0.001
report3 = data.frame()
fit = lm(formula = mpg ~ year, data = d_train)
report3 = add_to_report(report3, fit, d_train, d_test)

#summary(fit)

##p_value = 0.01
fit = lm(formula = mpg ~ year + weight + hpow, data = d_train)
report3 = add_to_report(report3, fit, d_train, d_test)

#p_value = 0.1
fit = lm(formula = mpg ~ year + weight + hpow +disp + cyl, data = d_train)
report3 = add_to_report(report3, fit, d_train, d_test)
print(report3)
```

```
##                            formula n_features train_mse  test_mse
## 1                             year          1  21.75317 111.72128
## 2             year + weight + hpow          3  10.58642  68.21662
## 3 year + weight + hpow + disp + cyl          5  10.25539  59.47237
```

When compairing feature with p less than 0.001, only one feature ***year*** had p value less than 0.001 but still its train and test MSE was still high than the minimum in stepwise

When compairing features with p<0.01, ***year, weight and hpow*** had p value less than 0.01 , though the MSE value droped to 10.58 for train data, it was still slightly higher than the minimum in stepwise.

When compairing features with p<0.1, *** year, weight, hpow, didp and cyl*** had p value less than 0.1 , though the MSE droped , it was slightly higher the minimum in stepwise

i

discussing the results

- while steping backwards ie decreasing the the feature , AIC stoped improving on the 5th step with less MSE for the train data compared to one with large train instance.
- while for **stepind forward** AIC stoped improving on the third step. with less MSE for the train data compared to one with large train instance

What is the winning method now?

stepwise method is the winning method because it generated the least MSE

which model is the best

The step forward model with 1 step was the best because it had the least test MSE in the test instance.

Are the results different from the case with a larger training set, and why?

Yes the results are different, the ones with larger training set have a high train_mse compared to the ones with 20 training instances.

I think the MSE is lower with lower training instance especially when most of the features in the training instance are correlated to the test feature under investigation.

j

```
model  = step(empty_model_20, direction='forward',scope=formula(full_model_20),steps=1)
```

```
## Start:  AIC=66.33
## mpg ~ 1
##
##          Df Sum of Sq    RSS    AIC
## + cyl     1   125.408 373.34 62.535
## + disp    1   115.950 382.80 63.035
## + weight  1    98.040 400.71 63.950
## + year    1    73.264 425.48 65.150
## + orig    1    51.047 447.70 66.168
## <none>                498.75 66.327
## + hpow    1    32.399 466.35 66.984
## + accel   1    28.447 470.30 67.153
##
## Step:  AIC=62.54
## mpg ~ cyl
```

```
coef(model)
```

```
## (Intercept)         cyl
##   56.468421   -5.744737
```

Thinking of the meaning of this feature I think the negative intercept does not make sense , In my opionion I think the lower the number of cylinders the lower the lower the miles a car can cover.