Practice 2

Read and Clean

```r
readData = function() {
df = read.csv('titanic.csv', sep=';', header=T)
df = df[complete.cases(df),]
df$sex = droplevels(df$sex) # There is a level "" that shouldn't be there.
df$embarked = droplevels(df$embarked) # There is a level "" that shouldn't be there.
# http://stackoverflow.com/questions/4605206/drop-data-frame-columns-by-name
df = subset(df, select = -c(ticket,cabin) )
# I need to convert all factor features to numbers to produce the correlation matrix.
# First two functions, applied to each row, substitute the strings by numbers.
sexToNum = function(x) ifelse((x %in% "male"), 1, 2)
embToNum = function(x) ifelse(x=="C", 1, ifelse(x=="Q", 2, 3))
#df$pclass = as.numeric(levels(df$pclass))[df$pclass]
df$sex = sexToNum(df$sex)
df$embarked = embToNum(df$embarked)
df$age = as.numeric(df$age)
df$fare = as.numeric(df$fare)
df = df[c("survived","pclass","sex","age","sibsp","parch","fare","embarked")]
df
}
```

Split

```r
splitdf = function(dataframe, seed=NULL, percentage=0.8) {
  if (!is.null(seed)) set.seed(seed)
  index = 1:nrow(dataframe)
  numTrainingSamples = round(length(index) * percentage)
  trainindex = sample(index, numTrainingSamples)
  trainset = dataframe[trainindex, ]
  testset = dataframe[-trainindex, ]
  list(trainset=trainset,testset=testset)
}
```

Feature Selection

```r
easyFeatureSelection = function(split) {
corrs = abs(cor(split$trainset)[1,])
toKeep = corrs[corrs > 0.1 & !is.na(corrs)]
split$trainset = subset(split$trainset, select=names(toKeep))
split$testset = subset(split$testset, select=names(toKeep))
split
}
```

Training & Model Evaluation

```r
modelEvaluation = function(split, formula=survived~.) {
  # Fit the model with the training dataset.
  model = glm(formula, data = split$trainset, family = "binomial")
  # The predicted probabilities given to each sample in the test set.
  probs = predict(model, type="response", newdata = split$testset)
  predictions = data.frame(survived = split$testset$survived, pred=probs)
  myROC = roc(survived ~ probs, predictions)
  optimalThreshold = coords(myROC, "best", ret = "threshold")
  # To compute F1 = 2TP/(2TP+FP+FN)
  T = table(predictions$survived, predictions$pred > optimalThreshold)
  F1 = (2*(T[1,1]))/((2*(T[1,1]))+T[2,1]+T[1,2])
  F1
}
```

Question 1

Single Splits

```r
library(ggplot2)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(caret)
```

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

```r
df = readData()
split = splitdf(df, 10, 0.8)
split = easyFeatureSelection(split)
singleSplitPerformance = modelEvaluation(split)
cat("Performance = ", singleSplitPerformance)
```

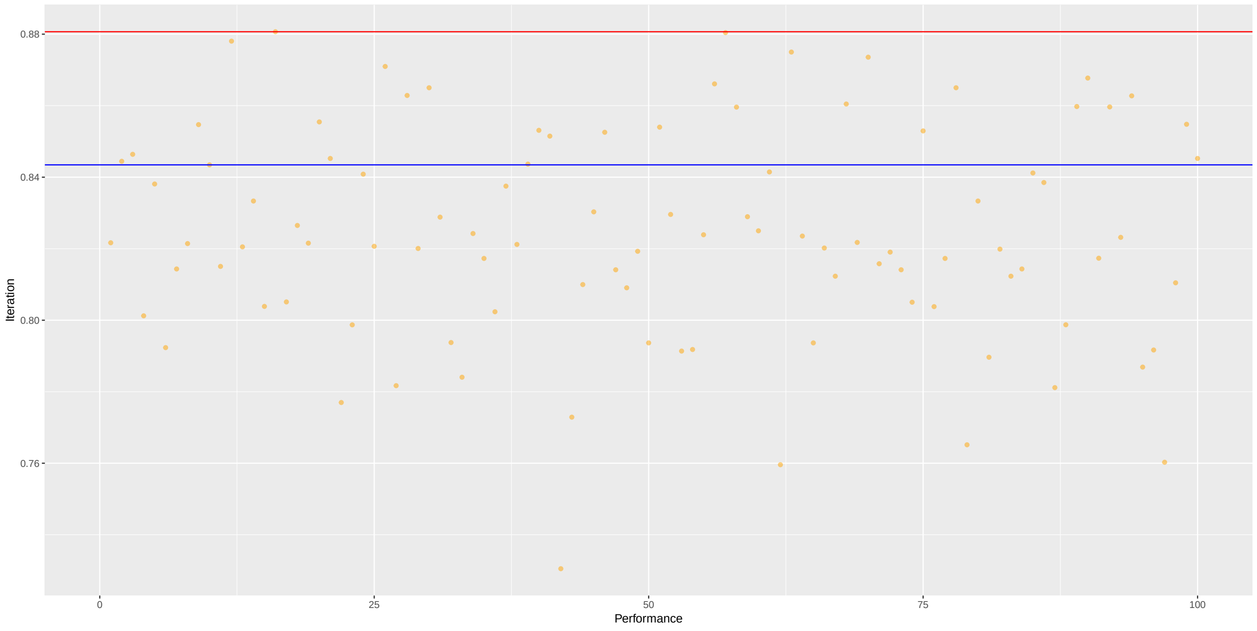
```
## Performance =  0.8434505
```

Iteration

```r
perf = c(0.0)
iterations = data.frame()
for(i in 1:100) {
  split = splitdf(df, i, 0.8)
  split = easyFeatureSelection(split)
  perf[i] = modelEvaluation(split)
  row = data.frame(split =i, perf = perf[i])
  iterations = rbind(iterations, row)
}
indexOfMaxPerformance = which.max(perf)
IT_maxPerf = perf[indexOfMaxPerformance]
cat("Max performance = ", IT_maxPerf)
```

```
## Max performance =  0.8806818
```

Comparison: Plot single split vs iteration splits



Compared with the single split, the iterations had a much better performance.

Question 2

Cross validation

Using corss validation, we will divide the data into 60-20-20: 60 for training, 20 for validation and 20 for testing.

On the 20% validation set:

- create a certain **nbfolds** folds
- On each fold: train the model on part of the data and use the remaining part for validation.
- Compute best performance for each fold depending on the chosen method (roc or acc) and store the results in a dataframe

On the 20% testing set:

- Select the best threshold with the best performance from the data frame
- Predict using the same model on the remaining 20% test set and return performance.

The below method will be used to return best performance based on best threshold among **nbfolds** based on both methods: ROC and Accuracy

```
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```
cross_validation = function(nbfolds, split, method = 'roc'){
  perf = data.frame()
  #create folds
  folds  = createFolds(split$trainset$survived, nbfolds, list = TRUE, returnTrain = TRUE)
  #loop nbfolds times to find optimal threshold
  for(i in 1:nbfolds)
    {
      #train the model on part of the data
      model = glm(survived~., data=split$trainset[folds[[i]],], family = "binomial")
      #validate on the remaining part of the data
      probs = predict(model, type="response", newdata = split$trainset[-folds[[i]],])
      if(method == 'roc')
        {
            #Threshold selection based on roc
            #store predictions in data frame
            predictions = data.frame(survived=split$trainset[-folds[[i]],]$survived, pred=probs)
            myROC = roc(survived ~ probs, predictions)
            optimalThreshold = coords(myROC, "best", ret = "threshold",drop=FALSE)
            T = table(predictions$survived, predictions$pred > optimalThreshold[[1]])
            #Measure performace based on best threshold and add performance + threshold to data frame
            F1 = (2*(T[1,1]))/((2*(T[1,1]))+T[2,1]+T[1,2])
            row = data.frame(threshold = optimalThreshold[[1]], accuracy = F1)
       }
       else
         {
            #Threshold selection based on Accuracy instead of ROC
            #create a prediction object based on the predicted values
            pred = prediction(probs,split$trainset[-folds[[i]],]$survived)
            #measure performance of the prediction
            acc.perf = performance(pred, measure = "acc")
            #Find index of most accurate threshold and add threshold in data frame
            ind = which.max( slot(acc.perf, "y.values")[[1]] )
            acc = slot(acc.perf, "y.values")[[1]][ind]
            optimalThreshold = slot(acc.perf, "x.values")[[1]][ind]
            row = data.frame(threshold = optimalThreshold, accuracy = acc)
         }
        #Sote the best thresholds with their performance in the perf dataframe
        perf = rbind(perf, row)
  }
       #Get the threshold with the max accuracy among the nbfolds and predict based on it on the unseen
       indexOfMaxPerformance = which.max(perf$accuracy)
       optThresh = perf$threshold[indexOfMaxPerformance]
       probs = predict(model, type="response", newdata = split$testset)
       predictions = data.frame(survived=split$testset$survived, pred=probs)
       T = table(predictions$survived, predictions$pred > optThresh)
       F1 = (2*(T[1,1]))/((2*(T[1,1]))+T[2,1]+T[1,2])
       F1
}
```

Do 100 iterations:

- Split the data randomly in test and train
- run cross validation on 20% of the data and test on the remaining 20%
- Store the results in a dataframe

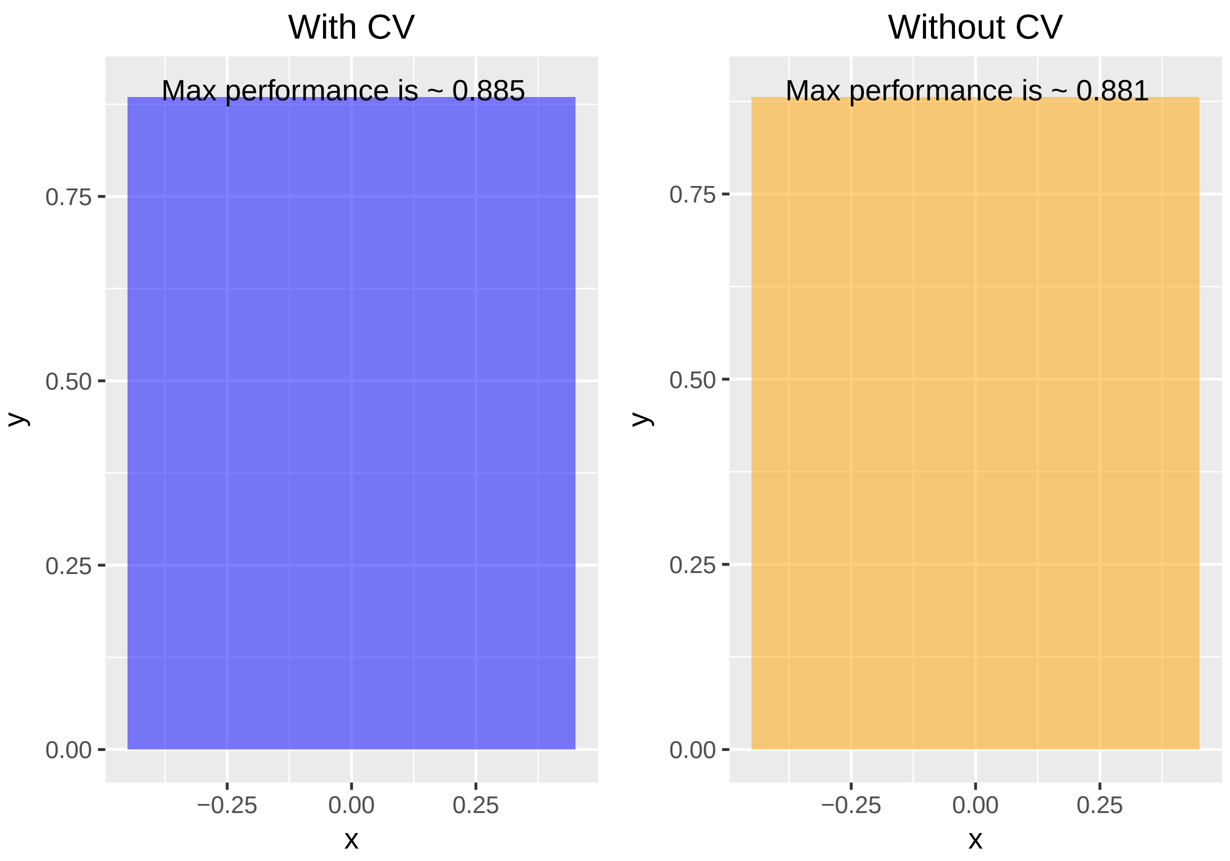
```r
#Split the data into 60-20-20, perform cross validation on validation set and evaluate on test set.
sol = data.frame()
data = data.frame()
for(i in 1:100){
  split = splitdf(df, i, 0.6)
  #Choose variables on the 60% training set
  split = easyFeatureSelection(split)
  #divide the reamining 60% into 50% validation (for cv) and 50% for testing
  testvalset = split$testset
  testsplit = splitdf(testvalset, i, 0.5)
  perf = cross_validation(10,testsplit)
  data = data.frame(split = i, performance = perf)
  sol = rbind(sol, data)
}
indexOfMaxPerformance = which.max(sol$performance)
CV_maxPerf = sol$performance[indexOfMaxPerformance]
```

Comparison: with and without CV

We can see that with CV the performance has slightly increased compared to no CV.

```r
library(gridExtra)
indexOfMaxPerformance = which.max(sol$performance)
maxPerf = sol$performance[indexOfMaxPerformance]
d1= data.frame(x=c(0), y=c(CV_maxPerf))
d2= data.frame(x=c(0), y=c(IT_maxPerf))
p1= ggplot(data=d1, aes(x, y)) +
  geom_bar(stat="identity", fill = 'blue', alpha = 0.5)+
  annotate("text", label= paste(c("Max performance is ~"), round(CV_maxPerf,3), " "), x= 0, y = CV_maxPe
  ggtitle("With CV") +
  theme(plot.title = element_text(hjust = 0.5))
p2 = ggplot(data=d2, aes(x, y)) +
  geom_bar(stat="identity", fill = 'orange', alpha = 0.5) +
  annotate("text", label= paste(c("Max performance is ~"), round(IT_maxPerf,3), " "), x= 0, y = IT_maxPe
  ggtitle("Without CV") +
  theme(plot.title = element_text(hjust = 0.5))

grid.arrange(p1, p2, ncol=2)
```



Question 3

Use Accuracy as a measure instead of ROC to choose optimal threshold. Same as above: Do 100 iterations and continuously split the data into train and test sets.

```r
sol1 = data.frame()
for(i in 1:100){
  split = splitdf(df, i, 0.6)
  split = easyFeatureSelection(split)
  testvalset = split$testset
  testsplit = splitdf(testvalset, i, 0.5)
  perf1 = cross_validation(10,testsplit, 'acc')
  data2 = data.frame(split = i, performance = perf1)
  sol1 = rbind(sol1, data2)
}
```

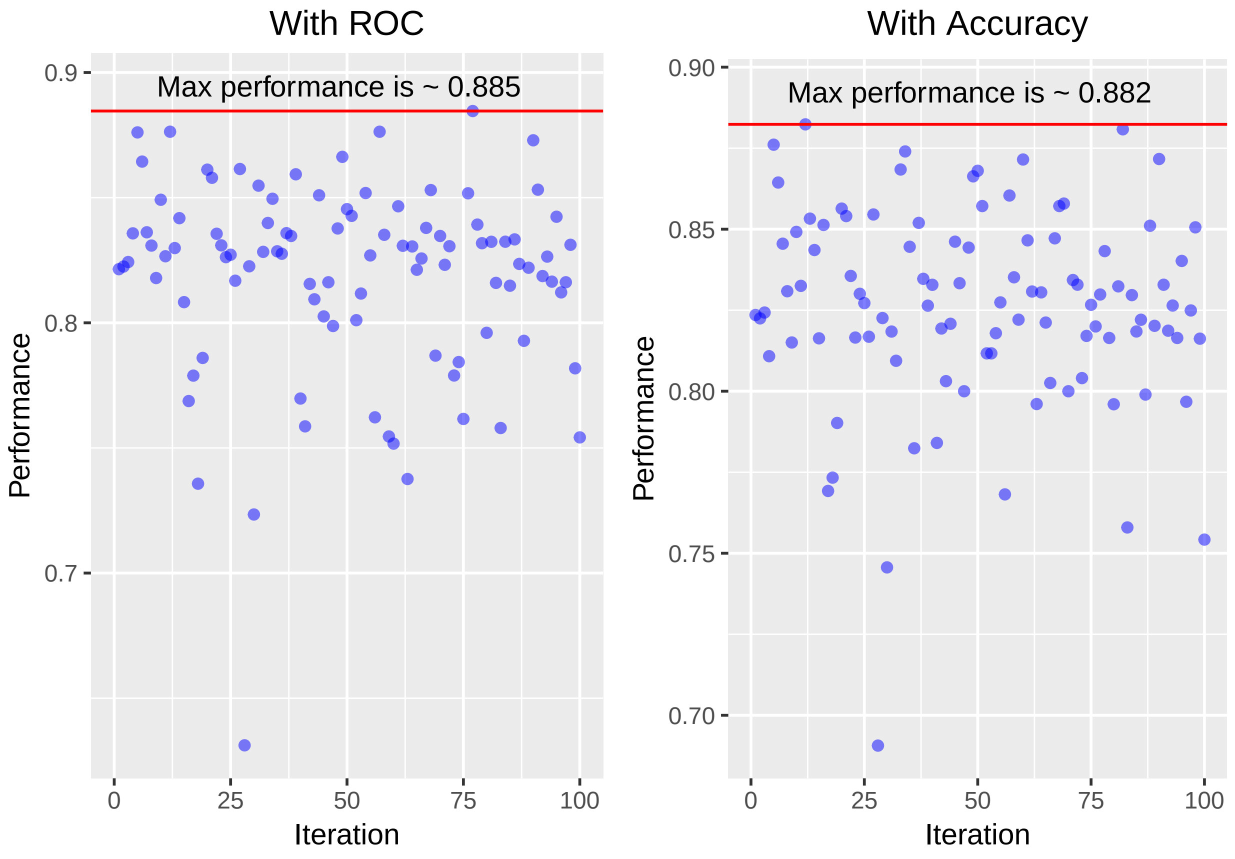
Comparison: CV with ROC vs Accuracy as measures

```r
indexOfMaxPerformance1 = which.max(sol1$performance)
Acc_maxPerf = sol1$performance[indexOfMaxPerformance1]

p1 = ggplot(data = sol, aes(y=performance, x=split)) +
  xlab("Iteration") +
  ylab("Performance") +
  xlim(0,100) +
  geom_point(alpha = 0.5, color = "blue") +
  geom_hline(yintercept = maxPerf,  color = "red") +
  annotate("text", label= paste(c("Max performance is ~"), round(CV_maxPerf,3), " "), x= 50, y = CV_maxI
  ggtitle("With ROC") +
  theme(plot.title = element_text(hjust = 0.5))

p2 = ggplot(data = sol1, aes(y=performance, x=split)) +
  xlab("Iteration") +
  ylab("Performance") +
  xlim(0,100) +
  geom_point(alpha = 0.5, color = "blue") +
  geom_hline(yintercept = Acc_maxPerf,  color = "red") +
  annotate("text", label= paste(c("Max performance is ~"), round(Acc_maxPerf,3), " "), x= 50, y = Acc_ma
  ggtitle("With Accuracy") +
  theme(plot.title = element_text(hjust = 0.5))


grid.arrange(p1, p2, ncol=2)
```



The performance of the ROC measure is sightly better than the Accuracy measure.