Machine Learning with R classifiers

Name: Ng Wei Hong

Table of Contents

(NOTE: not all code workings are shown here, for full codes used in the analysis, please look at the R script

Also, all performance are from testing against our "Defects.csv" dataset. "Defects.csv" is the only dataset we are using in this work.)

1. Data exploration

Working:

```
> numberofdefects = DS[(DS$defects == TRUE),]
> numberofnondefects = DS[(DS$defects == FALSE),]
>
> ratioofdefectstonondefects = nrow(numberofdefects) / nrow(numberofnonde
fects)
> ratioofdefectstonondefects
[1] 0.07364




> summary(DS)
      loc               v.g.              ev.g.             iv.g.
 n
 Min.   :  0.0    Min.   :  1.00    Min.   :  1.00    Min.   :  1.00    Min.
   :   1
 1st Qu.:  7.0    1st Qu.:  1.00    1st Qu.:  1.00    1st Qu.:  1.00    1st Q
u.:  25
 Median : 13.0    Median :  3.00    Median :  1.00    Median :  2.00    Media
n :  58
 Mean   : 23.4    Mean   :  5.51    Mean   :  2.77    Mean   :  3.32    Mean
   : 118
 3rd Qu.: 26.0    3rd Qu.:  6.00    3rd Qu.:  3.00    3rd Qu.:  3.00    3rd Q
u.: 126
 Max.   :602.0    Max.   :136.00    Max.   :123.00    Max.   :123.00    Max.
   :2785
       v                l                 d                 i
e
 Min.   :     0   Min.   :0.0000    Min.   :  0.00    Min.   :  0.0   Min.
 :       0
 1st Qu.:    98   1st Qu.:0.0475    1st Qu.:  5.71    1st Qu.: 14.1   1st Qu
.:     561
 Median :   276   Median :0.0800    Median : 11.64    Median : 23.6   Median
 :    3191
 Mean   :   700   Mean   :0.1286    Mean   : 15.41    Mean   : 32.9   Mean
 :   28849
 3rd Qu.:   676   3rd Qu.:0.1600    3rd Qu.: 20.50    3rd Qu.: 41.1   3rd Qu
.:   12391
 Max.   :25943    Max.   :2.0000    Max.   :270.66    Max.   :598.3   Max.
 :4279633
       b                t                lOCode           lOComment         lOBl
ank
 Min.   :0.000    Min.   :     0    Min.   :  0.0   Min.   :  0.0   Min.
: 0.00
 1st Qu.:0.030    1st Qu.:    31    1st Qu.:  7.0   1st Qu.:  0.0   1st Qu.
: 0.00
 Median :0.090    Median :   177    Median : 13.0   Median :  0.0   Median
: 0.00
 Mean   :0.235    Mean   :  1603    Mean   : 22.5   Mean   :  4.7   Mean
```

```
: 0.94
 3rd Qu.:0.230   3rd Qu.:    688   3rd Qu.: 24.0   3rd Qu.:   5.0   3rd Qu.
: 1.00
 Max.   :8.650   Max.   :237757   Max.   :600.0   Max.   :159.0   Max.
:48.00
 lOCodeAndComment     uniq_Op         uniq_Opnd         total_Op          total
_Opnd
 Min.   :  0.00   Min.   : 1.0   Min.   :  0.0   Min.   :   1.0   Min.
:   0.0
 1st Qu.:  1.00   1st Qu.: 8.0   1st Qu.:  6.0   1st Qu.:  15.0   1st Qu.
:  10.0
 Median :  2.50   Median :12.0   Median : 12.0   Median :  33.0   Median
:  24.0
 Mean   :  6.75   Mean   :13.3   Mean   : 20.9   Mean   :  66.6   Mean
:  50.9
 3rd Qu.:  8.00   3rd Qu.:17.0   3rd Qu.: 25.0   3rd Qu.:  72.0   3rd Qu.
:  56.0
 Max.   :225.00   Max.   :99.0   Max.   :538.0   Max.   :1641.0   Max.
:1144.0
  branchCount       defects
 Min.   :  1.00   Mode :logical
 1st Qu.:  1.00   FALSE:1032
 Median :  5.00   TRUE :76
 Mean   :  9.58
 3rd Qu.: 11.00
 Max.   :236.00




> x<-DS$loc
> y<-DS$lOCode + DS$lOBlank
> t.test(x,y)

        Welch Two Sample t-test

data:  x and y
t = -0.0017, df = 2214, p-value = 1
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.943  2.938
sample estimates:
mean of x mean of y
     23.4      23.4
```

Answer:

The ratio of defect-prone to not-defect-prone is 0.07364.

We noticed that:

[DS$loc == DS$lOCode + DS$IOBlank]

, as proven by working code shown above, where the p-value is = 1, we accept the null hypothesis that they are equal in terms of mean.

Since they are the same, we have decided that [DS$loc] is redundant and will be removed in pre-processing in next step.

Because all other data the column can take in the Dataset has more than 1 distinct value and does not seem redundant, we are not excluding them, since we do not understand the data enough yet to make any statements about their importance and whether they are useless in our analysis.

All columns having distinct dataset is found by using the code “apply(DS,2,unique)”.

2. Pre-processing

Working:

```
> #we are making a pre-processed dataset that does not include [DS$loc] f
or use in our analysis
> DSpreprocessed = DS[,2:22]
>

> #encode defects column as factor for analysis
> DSpreprocessed$defects = as.factor(DSpreprocessed$defects)
```

Answer:

We created a dataframe that excludes [DS$loc] in pre-processing, because the column is redundant in our analysis.

Also, [DSpreprocessed$defects] column is encoded as factors in pre-processing for analysis.

3. Separating datasets

Working:

```
> #separate into training and test dataset
> set.seed(28055322) #random seed
> train.row = sample(1:nrow(DSpreprocessed), 0.7*nrow(DSpreprocessed))
> iris.train = DSpreprocessed[train.row,]
> iris.test = DSpreprocessed[-train.row,]
```

Answer:

We separated our pre-processed dataset into training and test dataset for our analysis later.

4. and 5. Creating Classifiers and testing their accuracy.

Here, we will create the following classifiers and test their accuracy:

- Decision Tree
- Naïve Bayes
- Bagging
- Boosting
- Random Forest

Working:

Tree model:

```
> #build tree with traning data, with defect as target, and others as pre
dictors
> #and calculate its accuracy
> dtreefit <- tree(defects ~. ,data = iris.train)
> treepredict <- predict(dtreefit, iris.test, type = "class")
>
> v <- table(observed = iris.test$defects , predicted = treepredict)
> v
        predicted
observed FALSE TRUE
   FALSE   304   15
   TRUE     10    4
> accuracy <- (v[1] + v[4])/(sum(v))
> accuracy
[1] 0.9249
```

Naive Bayes model:

```
> #build Naive Bayes classifier, and calculate accuracy
> dbayesfit <- naiveBayes(defects ~. ,data = iris.train)
> bayespredict <- predict(dbayesfit, iris.test)
>
> v <- table(observed = iris.test$defects , predicted = bayespredict)
> v
        predicted
observed FALSE TRUE
```

```
   FALSE    295    24
   TRUE       7     7
> accuracy <- (v[1] + v[4])/(sum(v))
> accuracy
[1] 0.9069
```

Bagging model:

```
> #build classifier via bagging, and calculate accuracy
> dbagfit <- bagging(defects ~. ,data = iris.train, mfinal = 10)
> bagpredict <- predict.bagging(dbagfit, newdata = iris.test)
>
> v <- table(observed = iris.test$defects , predicted = bagpredict$class)
> v
        predicted
observed FALSE TRUE
   FALSE    315    4
   TRUE      11    3
> accuracy <- (v[1] + v[4])/(sum(v))
> accuracy
[1] 0.955
```

Boosting model:

```
> #build classifier via boosting, and calculate accuracy
> dboostfit <- boosting(defects ~. ,data = iris.train, mfinal = 100)
> boostpredict <- predict.boosting(dboostfit, newdata = iris.test)
>
> v <- table(observed = iris.test$defects , predicted = boostpredict$clas
s)
> v
        predicted
observed FALSE TRUE
   FALSE    313    6
   TRUE       9    5
> accuracy <- (v[1] + v[4])/(sum(v))
> accuracy
[1] 0.955
```

Random Forest model:

```
> #build random forest classifier, and calculate accuracy
> drandomforestfit <- randomForest(defects ~. ,data = iris.train)
> randomforestpredict <- predict(drandomforestfit, newdata = iris.test)
>
```

```
> v <- table(observed = iris.test$defects , predicted = randomforestpredi
ct)
> v
        predicted
observed FALSE TRUE
   FALSE   314    5
   TRUE      9    5
> accuracy <- (v[1] + v[4])/(sum(v))
> accuracy
[1] 0.958
```

Answer:

Accuracy for each model:

- Tree: 0.9249
- Naïve Bayes: 0.9069
- Bagging: 0.955
- Boosting: 0.955
- Random Forest: 0.958

6. Assessing classifier performance

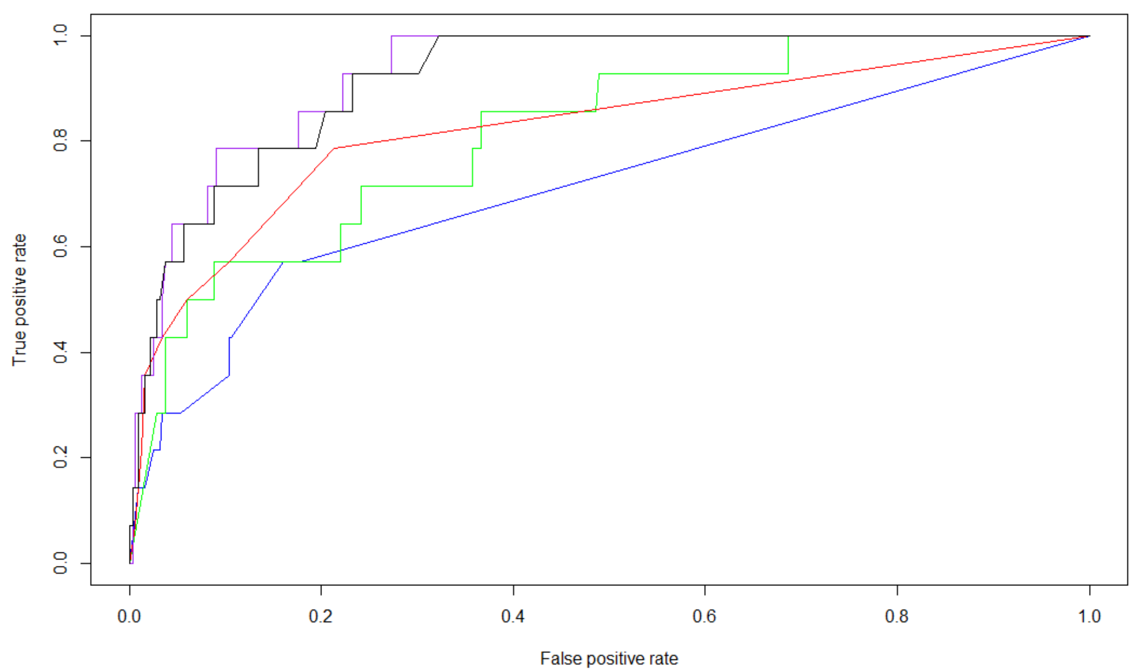
Here, we plot False positive rate against true positive rate for classifiers, and also calculating their AUC score(for assessing classifier performance)

Working:

```
> ################       Question 6      ######################
>
> #get confidence level for all classfiers
> conf.tree.predict <- predict(dtreefit, iris.test)
> conf.bayes.predict <- predict(dbayesfit, iris.test, type = "raw")
> conf.bag.predict<- bagpredict$prob
> conf.boosting.predict<- boostpredict$prob
> conf.randomforest.predict <- predict(drandomforestfit, iris.test, type
= "prob")
>
>
>
> #plot ROCR for all classifier
> prediction.tree <- prediction(conf.tree.predict[,2], iris.test$defects)
> perf.tree <- performance(prediction.tree, "tpr", "fpr")
> plot(perf.tree,col = "blue")
>
>
> prediction.bayes <- prediction(conf.bayes.predict[,2], iris.test$defect
s)
> perf.bayes <- performance(prediction.bayes, "tpr", "fpr")
> plot(perf.bayes,add = TRUE, col = "green")
>
>
>
> prediction.bag <- prediction(conf.bag.predict[,2], iris.test$defects)
> perf.bag <- performance(prediction.bag, "tpr", "fpr")
> plot(perf.bag,add = TRUE, col = "red")
>
>
>
> prediction.boosting <- prediction(conf.boosting.predict[,2], iris.test$
defects)
> perf.boosting <- performance(prediction.boosting, "tpr", "fpr")
> plot(perf.boosting,add = TRUE, col = "purple")
>
>
>
> prediction.forest <- prediction(conf.randomforest.predict[,2], iris.tes
t$defects)
> perf.forest <- performance(prediction.forest, "tpr", "fpr")
> plot(perf.forest,add = TRUE, col = "black")
>
>
> #calculate AUC and print
```

```
> auctree <- performance(prediction.forest,"auc")
> print(as.numeric(auctree@y.values))
[1] 0.9183
>
> aucbayes <- performance(prediction.bayes,"auc")
> print(as.numeric(aucbayes@y.values))
[1] 0.8115764
>
> aucbag <- performance(prediction.bag,"auc")
> print(as.numeric(aucbag@y.values))
[1] 0.8215
>
> aucboosting <- performance(prediction.boosting,"auc")
> print(as.numeric(aucboosting@y.values))
[1] 0.9276
>
> aucforest <- performance(prediction.forest,"auc")
> print(as.numeric(aucforest@y.values))
[1] 0.9183
```

Graph:



Answer:

For, AUC values, please refer to question 7’s table.

7. Classifier performance assessment

Table:

Classifier	Accuracy	AUC score
Tree	0.9249	0.9183
Naïve Bayes	0.9069	0.8115764
Bagging	0.955	0.8215
Boosting	0.955	0.9276
Random forest	0.958	0.9183

Answer:

In terms of accuracy for predicting the test data used for our analysis, Random forest classifier is the best here. In terms of AUC score in relation to the test data used for our analysis, Boosting classifier is best in this regard.

For "best" classifier, there seems to be no clear winner between our Boosting and Random forest classifier.

If the test data used for our analysis changes, such as when larger testing data is used, the values for Accuracy and AUC score would change then, due to a different test data, and with that, it may be possible to determine the best classifier here.

8. Determining most important variables in our data

We do this by finding the most important variables in our classifiers.

Working:

```
> ################        Question 8       ######################
>
> dtreefit
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

  1) root 775 400 FALSE ( 0.920 0.080 )
    2) loCodeAndComment < 6.5 527 100 FALSE ( 0.972 0.028 )
      4) l < 0.46 491  80 FALSE ( 0.984 0.016 )
        8) loComment < 2.5 411  30 FALSE ( 0.995 0.005 )
         16) loCode < 3.5 44  20 FALSE ( 0.955 0.045 )
           32) loCodeAndComment < 0.5 37   0 FALSE ( 1.000 0.000 ) *
           33) loCodeAndComment > 0.5 7   8 FALSE ( 0.714 0.286 ) *
         17) loCode > 3.5 367   0 FALSE ( 1.000 0.000 ) *
        9) loComment > 2.5 80  40 FALSE ( 0.925 0.075 )
         18) uniq_Op < 10.5 36   0 FALSE ( 1.000 0.000 ) *
         19) uniq_Op > 10.5 44  40 FALSE ( 0.864 0.136 )
           38) i < 31.815 28   9 FALSE ( 0.964 0.036 ) *
           39) i > 31.815 16  20 FALSE ( 0.688 0.312 )
             78) n < 149 10  10 FALSE ( 0.500 0.500 ) *
             79) n > 149 6   0 FALSE ( 1.000 0.000 ) *
      5) l > 0.46 36  40 FALSE ( 0.806 0.194 )
       10) v < 33.935 27   0 FALSE ( 1.000 0.000 ) *
       11) v > 33.935 9  10 TRUE ( 0.222 0.778 ) *
    3) loCodeAndComment > 6.5 248 200 FALSE ( 0.810 0.190 )
      6) loComment < 37.5 232 200 FALSE ( 0.845 0.155 )
       12) loComment < 1.5 51  60 FALSE ( 0.706 0.294 )
         24) n < 202 39  50 FALSE ( 0.615 0.385 )
           48) loCode < 26.5 33  40 FALSE ( 0.697 0.303 ) *
           49) loCode > 26.5 6   5 TRUE ( 0.167 0.833 ) *
         25) n > 202 12   0 FALSE ( 1.000 0.000 ) *
       13) loComment > 1.5 181 100 FALSE ( 0.884 0.116 )
         26) loBlank < 10.5 174 100 FALSE ( 0.902 0.098 )
           52) uniq_Opnd < 15.5 22   0 FALSE ( 1.000 0.000 ) *
           53) uniq_Opnd > 15.5 152 100 FALSE ( 0.888 0.112 )
            106) loCode < 19.5 22  30 FALSE ( 0.682 0.318 )
              212) loBlank < 0.5 9   0 FALSE ( 1.000 0.000 ) *
              213) loBlank > 0.5 13  20 TRUE ( 0.462 0.538 ) *
            107) loCode > 19.5 130  70 FALSE ( 0.923 0.077 )
              214) uniq_Opnd < 35.5 54  10 FALSE ( 0.981 0.019 )
                428) i < 25.76 5   5 FALSE ( 0.800 0.200 ) *
                429) i > 25.76 49   0 FALSE ( 1.000 0.000 ) *
              215) uniq_Opnd > 35.5 76  60 FALSE ( 0.882 0.118 )
                430) branchCount < 24.5 57  50 FALSE ( 0.842 0.158 ) *
                431) branchCount > 24.5 19   0 FALSE ( 1.000 0.000 ) *
         27) loBlank > 10.5 7  10 TRUE ( 0.429 0.571 ) *
```

```
      7) loComment > 37.5 16  20 TRUE ( 0.312 0.688 )
       14) uniq_Opnd < 69.5 7   8 FALSE ( 0.714 0.286 ) *
       15) uniq_Opnd > 69.5 9   0 TRUE ( 0.000 1.000 ) *

> #print summary and importance data, to determine most
important variables for each classifier
> summary(dtreefit)

Classification tree:
tree(formula = defects ~ ., data = iris.train)
Variables actually used in tree construction:
 [1] "loCodeAndComment" "l"
 [3] "loComment"        "loCode"
 [5] "uniq_Op"          "i"
 [7] "n"                "v"
 [9] "loBlank"          "uniq_Opnd"
[11] "branchCount"
Number of terminal nodes:  22
Residual mean deviance:  0.235 = 177 / 753
Misclassification error rate: 0.0542 = 42 / 775
> dbagfit$importance
               b      branchCount                d
          0.0000           0.7568           2.4709
               e            ev.g.                i
          3.7522           0.8004           9.5081
           iv.g.                l          loBlank
          0.9000           3.5950          12.6414
          loCode loCodeAndComment        loComment
          8.0869          11.0901          16.0507
               n                t         total_Op
         11.4449           0.0000           2.1883
      total_Opnd          uniq_Op        uniq_Opnd
          0.9326           1.3570           3.3574
               v             v.g.
          9.8988           1.1688
> dboostfit$importance
               b      branchCount                d
           0.000            1.786            6.748
               e            ev.g.                i
           3.345            1.715           12.953
           iv.g.                l          loBlank
           2.163            1.806            7.447
          loCode loCodeAndComment        loComment
           6.483           12.832           10.756
               n                t         total_Op
           6.974            0.000            2.546
      total_Opnd          uniq_Op        uniq_Opnd
           2.694            4.846            7.894
               v             v.g.
           3.072            3.938
> drandomforestfit$importance
                 MeanDecreaseGini
v.g.                        2.875
ev.g.                       1.306
iv.g.                       2.290
n                           6.100
```

```
v                            7.556
l                            3.189
d                            4.908
i                            8.080
e                            6.044
b                            4.236
t                            6.126
lOCode                       6.742
lOComment                    9.734
lOBlank                      7.192
lOCodeAndComment             7.896
uniq_Op                      4.326
uniq_Opnd                    7.910
total_Op                     5.940
total_Opnd                   6.083
branchCount                  2.907
```

Answer:

Tree

Important variables for decision tree are:

- [1] "lOCodeAndComment"
- [2]"l"
- [3] "lOComment"
- [4]"lOCode"
- [5] "uniq_Op"
- [6]"i"
- [7] "n"
- [8]"v"
- [9] "lOBlank"
- [10]"uniq_Opnd"
- [11] "branchCount"

The most important variable for tree is “lOCodeAndComment", because it is the root node, which means it gives the most information gain and is therefore the most important.

The variables that can be omitted from tree are, the ones that are not important, meaning the ones that are not shown in the list of the 11 important variables for the decision tree above.

Bagging

Important variables for our Bagging classifier are:

- [1] “lOComment”
- [2] “lOBlank”
- [3] "lOCodeAndComment"
- [4] “n”
- [5] “v”
- [6] “i”

- [7] “lOCode”

The most important variable for Bagging Classifier is “lOComment”, with an importance value of 16.0507.

The variables that can be omitted from the classifier are, the ones that are not important, as in those that are not shown in the list of its 7 important variables above.

Boosting

Important variables for the Boosting classifier are:

- [1] “i”
- [2] "lOCodeAndComment"
- [3] “lOComment”
- [4] “uniq_Opnd”
- [5] “lOBlank”
- [6] “n”
- [7] “d”
- [8] “lOcode”
- [9] “uniq_Op”

The most important variable for our Boosting classifier is “i”, with an importance value of 12.953.

Variables that can be omitted are the ones that are not in the list of the 9 important variables directly above this.

Random Forest

Important variables for Random Forest classifier are:

- [1] “lOComment”
- [2] “i”
- [3] “uniq_Opnd”
- [4] "lOCodeAndComment"
- [5] “lOBlank”
- [6] “lOCode”
- [7] “t”
- [8] “n”
- [9] “total_Opnd”
- [10] “total_Op”
- [11] “d”

The most important variable for the Random Forest classifier is “lOComment”, with an importance value of 9.734.

Variables that can be omitted are the ones that are not in the list of the 9 important variables directly above this.

Reason for omitting variables: Because they are not important.

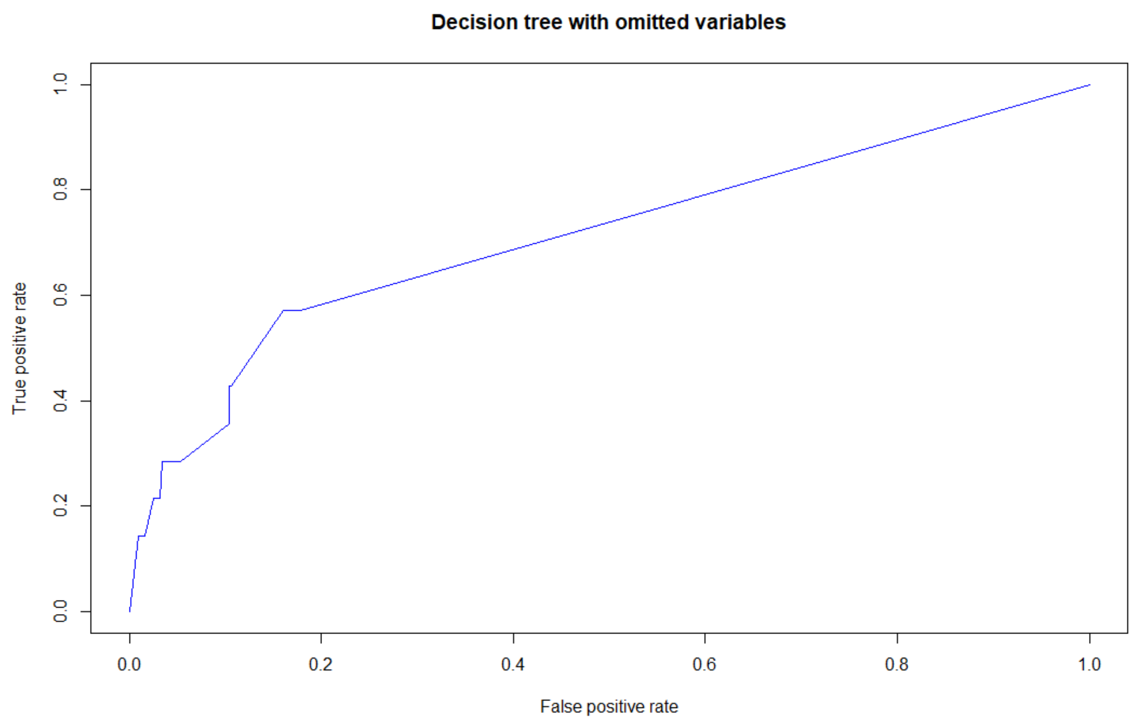
9. Modified tree with only important variables, and its performance

Working:

```
>
> #save random seed
> x<- .Random.seed
>
>
>
>
> ################       Question 9      ######################
>
>
>
>
> set.seed(x)
>
>
>
> #make a decision tree, with omitted variables
> new.treefit <- tree(defects ~ lOCodeAndComment + l + lOComment + lOCode
 + uniq_Op + i + n + v + lOBlank + uniq_Opnd + branchCount, data = iris.t
rain)
> new.treepredict <- predict(new.treefit, iris.test,type = "class")
>
>
> #calculate accuracy and print
>
> v <- table(observed = iris.test$defects , predicted = new.treepredict)
> v
        predicted
observed FALSE TRUE
   FALSE   307   12
   TRUE     10    4
> accuracy <- (v[1] + v[4])/(sum(v))
> accuracy
[1] 0.9339
>
> #plot ROCR
> new.conf.tree.predict <- predict(new.treefit, iris.test)
> new.prediction.tree <- prediction(new.conf.tree.predict[,2], iris.test$
defects)
> new.perf.tree <- performance(new.prediction.tree, "tpr", "fpr")
> plot(new.perf.tree,col = "blue", main = "Decision tree with omitted var
iables")
>
> #calculate AUC and print
> new.auctree <- performance(new.prediction.tree,"auc")
> print(as.numeric(new.auctree@y.values))
[1] 0.7109
>
```

```
>
>
>
> y <- .Random.seed
```

Graph:



Answer:

As shown in the Code workings, the decision tree created by omitting non-important variables has improved accuracy, but poorer AUC score, compared to the original decision with nothing omitted.

New tree(made with only important variables shown in the section for "Determining most important variables in our data"):

Accuracy | AUC score

0.9339	0.7109

Original tree(with all attributes included):

Accuracy | AUC score

0.9249	0.9183

10.Neural network classifier and its performance

Working:

```
> ################        Question 10       ######################
> set.seed(y)
>
> #make indicators for neutralnet
> iris.test$thedefects <- iris.test$defects == TRUE
> iris.test$thenondefects <- iris.test$defects == FALSE
> iris.train$thedefects <- iris.train$defects == TRUE
> iris.train$thenondefects <- iris.train$defects == FALSE
>
> #make neural network classifier, and check accuracy, using 3 most impor
tant variables from each classifier in question 8
> nnfit = neuralnet(thedefects + thenondefects~ lOComment + l +lOCodeAndC
omment + i + lOBlank + uniq_Opnd, iris.train , hidden=2, threshold = 0.01
)
>
> #does not need the target attribute for predicting
> nnpredict <- compute(nnfit, iris.test[,1:20])
> round.nnpredict <- round(nnpredict$net.result,0)
>
> df.round.nnpredict <- as.data.frame(as.table(round.nnpredict))
>
> s.df.round.nnpredict <-df.round.nnpredict[!df.round.nnpredict$Freq==0,]
> s.df.round.nnpredict$FREQ = NULL
> colnames(s.df.round.nnpredict) = c("Obs", "defects")
> s.df.round.nnpredict = s.df.round.nnpredict[order(s.df.round.nnpredict$
Obs),]
> #calculate accuracy and print
>
> v <- table(observed = iris.test$thedefects , predicted = s.df.round.nnp
redict$defects)
> v
        predicted
observed   A   B
   FALSE   3 316
   TRUE    3  11
> accuracy <- (v[2] + v[3])/(sum(v))
> accuracy
[1] 0.958
```

Answer:

For our data, the classifier's accuracy is as good as Random Forest's classifier's accuracy.

This is because it used the 3 most important attributes of all other classifier for classifying, and it is Neural network.

Summary:

Question//Step 1**(Data exploration)**:

- We explored the data and found that:
  - *[DS$loc == DS$IOCode + DS$IOBlank]*
  - *All other attributes seems normal, and we don't know enough about them to exclude*

Question//Step 2**(Pre-processing):**

- Excluded [DS$loc] in our pre-processing because it is redundant

In Question//Step 3 to Question//Step 7, we made classifiers for tree, naïve baiyes, etc, and summary of its performance is shown in the table:

Table:

Classifier	Accuracy	AUC score
Tree	0.9249	0.9183
Naïve Bayes	0.9069	0.8115764
Bagging	0.955	0.8215
Boosting	0.955	0.9276
Random forest	0.958	0.9183

Question 8**(Determining most important variables in our data)**:

- Found the most important variables for the classifiers

Question `10**(Neural network classifier)**:

- Made neural network classifier with 3 most important attributes from each classifier
- The classifier's accuracy is as good as Random Forest's classifier's accuracy, which is 0.958