CS – GY 6923 Machine Learning

Professor: Dr. Raman Kannan

HW3: Ensemble Methods

Avinash Authipudi

Section 1 – Review of the Classifiers

In the previous assignment, we ran 5 classifiers on 2 of our datasets (Dataset1 and Dataset2) and summarized the results. We also found that Decision Tree classifier worked the best on our datasets. We also analyzed what the variance of the different models.

We now proceed to use Ensemble techniques to improve performance of the classification algorithms run on our datasets. In this assignment, we have chosen the following Ensemble Methods:

1. Random Forest
2. Stacking with CV
3. Bagging with CV

We club ensemble methods Stacking and Cross validation in training the Logistic Regression, Decision Trees and KNN classifier models before combining them for prediction.

Since the datasets have a large number of observations, it takes a lot of computing time when algorithms are run on them. Hence, we make use of the parallel computation feature in R which can be seen in the below sections of the assignment. We also create threads that execute parallelly to decrease computation time.

Section 2 – Random Forest Method

The first Ensemble method we run is the Random Forest method which is based on decision tree algorithm. The implementation is as shown below.

Training Phase of Random Forest on Dataset1(ntree=500):

```
> rf <-randomForest(phishing~.,data=traindata, ntree=500)
> print(rf)

Call:
 randomForest(formula = phishing ~ ., data = traindata, ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 8

        OOB estimate of  error rate: 5.02%
Confusion matrix:
      0     1 class.error
0 18751   967  0.04904148
1  1095 20238  0.05132893
> mtry <- tuneRF(traindata[-79],traindata$phishing, ntreeTry=500,
+                stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
mtry = 8  OOB error = 5.03%
Searching left ...
mtry = 6        OOB error = 5.62%
-0.116707 0.01
Searching right ...
mtry = 12       OOB error = 4.57%
0.09249395 0.01
mtry = 18       OOB error = 4.38%
0.03948773 0.01
mtry = 27       OOB error = 4.42%
-0.008333333 0.01
> best.m <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
> print(mtry)
       mtry   OOBError
6.OOB     6 0.05617403
8.OOB     8 0.05030328
12.OOB   12 0.04565053
18.OOB   18 0.04384790
27.OOB   27 0.04421330
```

Once the initial model is run, we tune the model to find the least Out of Bag error rate. The m value related to the least OOB error is chosen in training the model again. This highly reduces correlation. We also have set ntree to 500 which is usually the default value. We have assumed a higher value of ntree since random forest models do not overfit very easily.

```
> rf <-randomForest(phishing~.,data=traindata, mtry=best.m, importance=TRUE,ntree=500)
> print(rf)

Call:
 randomForest(formula = phishing ~ ., data = traindata, mtry = best.m,      importance = TRUE, n
00)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 18

        OOB estimate of  error rate: 4.41%
Confusion matrix:
      0     1 class.error
0 18738   980  0.04970078
1   829 20504  0.03885998
> importance(rf)
                                  0           1 MeanDecreaseAccuracy MeanDecreaseGini
qty_dot_url              33.3152426  33.3804766           40.2667465     2.132182e+02
qty_hyphen_url           31.8725376  27.8400805           37.1360509     1.565280e+02
qty_underline_url        14.8082979  15.7886236           20.3751095     2.435835e+01
qty_slash_url            19.5043023  20.5333610           21.2719463     7.358342e+02
qty_questionmark_url      2.1778207   1.8993682            2.8264539     3.255104e-01
qty_equal_url             6.0739691   9.6794867            7.4504608     1.833447e+01
qty_at_url                3.6305878  11.4262813            7.6511506     4.743491e+00
qty_and_url               4.0071418   6.5495513            7.3389929     5.941803e+00
qty_exclamation_url       0.8547492   3.1191992            3.2628106     1.145096e+00
qty_space_url             0.0000000   0.0000000            0.0000000     4.241518e-02
qty_tilde_url             7.7983256   0.4989491            5.8432038     1.926562e+00
qty_comma_url            21.6276158  -7.7473946           17.1330589     5.084064e+00
qty_plus_url              8.2515373  -0.3160413            7.2214026     3.932840e+00
qty_asterisk_url          0.0000000   2.8647042            2.8645398     2.508463e-01
qty_hashtag_url           0.0000000   0.0000000            0.0000000     9.933333e-03
qty_dollar_url            0.0000000   1.2646715            1.2663735     9.713971e-02
qty_percent_url           9.2551132   8.3008537           11.8460241     1.330945e+01
qty_tld_url              27.2543002  20.5459579           30.7768612     4.263598e+01
length_url               34.6205598  40.3851594           46.6732670     9.753434e+02
qty_plus_file            [illegible]
qty_asterisk_file         4.4827077   3.5503356            3.9860725     2.034126e+02
qty_dollar_file           5.3112830   4.4476387            4.9156931     3.332399e+02
qty_percent_file          5.4432996   4.1576063            4.6848254     3.004976e+02
file_length              20.4481350  12.5092754           17.5304136     5.344134e+02
qty_dot_params            7.0870837  11.5606348            8.7360415     2.798682e+01
qty_hyphen_params         6.4125811   5.9755243            7.0406417     1.822683e+01
qty_underline_params      6.0995220   9.3164366            7.0001027     1.794873e+01
qty_slash_params          6.0477868   6.4758969            6.6673375     1.231688e+01
qty_questionmark_params   6.1680994   6.3481975            6.7282438     1.313156e+01
qty_equal_params          7.8247819   9.5728587            9.0485768     2.631738e+01
qty_at_params             6.7193155   6.8449853            7.2455156     1.161192e+01
qty_and_params            5.9544161   8.8059107            7.2833054     1.761586e+01
qty_exclamation_params    6.6234982   7.9219309            7.2287061     1.317745e+01
qty_comma_params          5.9333463   6.3303325            6.5126032     1.271132e+01
qty_plus_params           6.4782781   5.3758384            7.0382928     1.556803e+01
qty_dollar_params         6.3676676   5.6210769            6.9402822     1.437184e+01
qty_percent_params        6.5701289   6.7089827            7.1193044     1.613515e+01
params_length            10.6339546  15.1939621           17.3175289     5.996611e+01
tld_present_params        6.8998449   9.7394895            7.9263632     2.704724e+01
qty_params                7.6982281  10.1674360            9.4310525     2.555787e+01
email_in_url              3.5592357   2.8335587            4.3132129     7.938024e-01
time_response            63.4905169  58.1424826           75.6626237     6.328106e+02
domain_spf               34.7642979  32.1434754           44.3135374     1.054514e+02
asn_ip                   77.7303170  65.1153421           90.6316897     7.036742e+02
time_domain_activation  155.0861962 162.7532479          197.0569432     2.269844e+03
time_domain_expiration   49.2337006  49.1183259           61.8869838     5.236709e+02
qty_ip_resolved          39.5926415  48.7181604           56.2731221     2.218495e+02
qty_nameservers          61.3534187  57.9192013           73.2460295     3.186312e+02
qty_mx_servers           53.8022940  38.8666414           56.2204704     2.686639e+02
ttl_hostname             75.9070498  71.8784312           88.8362008     6.816860e+02
tls_ssl_certificate      37.5208869  30.1492355           40.0453470     1.380811e+02
qty_redirects            35.6226367  49.0660186           53.7796218     1.859990e+02
url_google_index         -2.5435353   8.0437804            6.6623504     2.319029e+00
domain_google_index       0.4182133   0.7169275            0.7108858     5.723095e-01
url_shortened             9.8878802  13.7780663           14.4553390     2.285991e+01
```

Training Phase Confusion Matrix for Dataset1:

```
> rf_pred = predict(rf, traindata[, -79])
> rf_mtab<-table(traindata$phishing,rf_pred)
> rf_cmx<-caret::confusionMatrix(rf_mtab)
> rf_mtab
   rf_pred
        0     1
  0 19714     4
  1    28 21305
> rf_cmx
Confusion Matrix and Statistics

   rf_pred
        0     1
  0 19714     4
  1    28 21305

               Accuracy : 0.9992
                 95% CI : (0.9989, 0.9995)
    No Information Rate : 0.5191
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9984

 Mcnemar's Test P-Value : 4.785e-05

            Sensitivity : 0.9986
            Specificity : 0.9998
         Pos Pred Value : 0.9998
         Neg Pred Value : 0.9987
             Prevalence : 0.4809
         Detection Rate : 0.4802
   Detection Prevalence : 0.4803
      Balanced Accuracy : 0.9992

       'Positive' Class : 0
```

Performance Metrics from the Training Phase of Dataset1:

```
> rf_cmx$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
  9.992205e-01   9.984386e-01   9.988997e-01   9.994668e-01   5.190860e-01   0.000000e+00
 McnemarPValue
  4.785484e-05

> pred1=predict(rf,type = "prob")
> perf = prediction(pred1[,2], traindata$phishing)
> pred3 = performance(perf, "tpr","fpr")
> plot(pred3,main="ROC Curve for Random Forest",col=2,lwd=2)
> abline(a=0,b=1,lwd=2,lty=2,col="gray")
```

The above metrics indicate that the model trained extremely good and we can expect good accuracy from the testing phase.

```
> rf_pred = predict(rf, testdata[, -79])
> rf_mtab<-table(testdata$phishing,rf_pred)
> rf_cmx_test<-caret::confusionMatrix(rf_mtab)
> rf_cmx_test
Confusion Matrix and Statistics

   rf_pred
       0    1
  0 7920  360
  1  382 8932

               Accuracy : 0.9578
                 95% CI : (0.9548, 0.9607)
    No Information Rate : 0.5281
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9154

 Mcnemar's Test P-Value : 0.4407

            Sensitivity : 0.9540
            Specificity : 0.9613
         Pos Pred Value : 0.9565
         Neg Pred Value : 0.9590
             Prevalence : 0.4719
         Detection Rate : 0.4502
   Detection Prevalence : 0.4706
      Balanced Accuracy : 0.9576

       'Positive' Class : 0

> rf_cmx_test$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
     0.9578265      0.9153732      0.9547517      0.9607489      0.5281346      0.0000000
 McnemarPValue
     0.4407459

> pred1=predict(rf,newdata = testdata, type = "prob")
> perf_test = prediction(pred1[,2], testdata$phishing)
> auc_test = performance(perf_test, "auc")
> pred3_test = performance(perf_test, "tpr","fpr")
> plot(pred3_test,main="ROC Curve for Random Forest - Test",col=2,lwd=2)
> abline(a=0,b=1,lwd=2,lty=2,col="gray")
```

The above snippets indicate the accuracy of the model which is a significant improvement over all our previously run classification models.

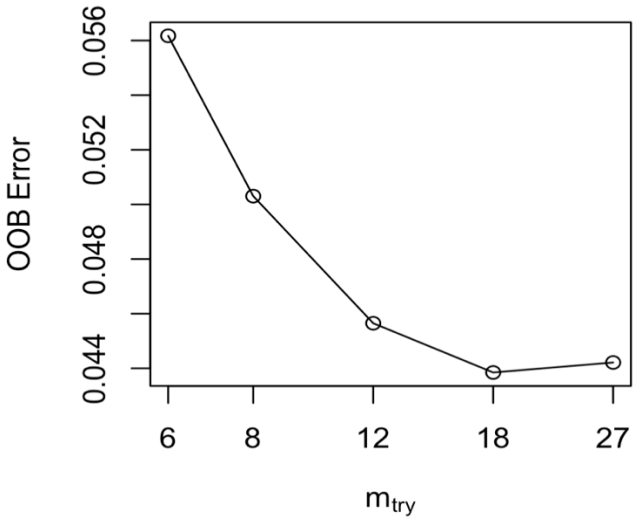

Fig: OOB vs M-try Graph for ntree=500
(Training Phase)

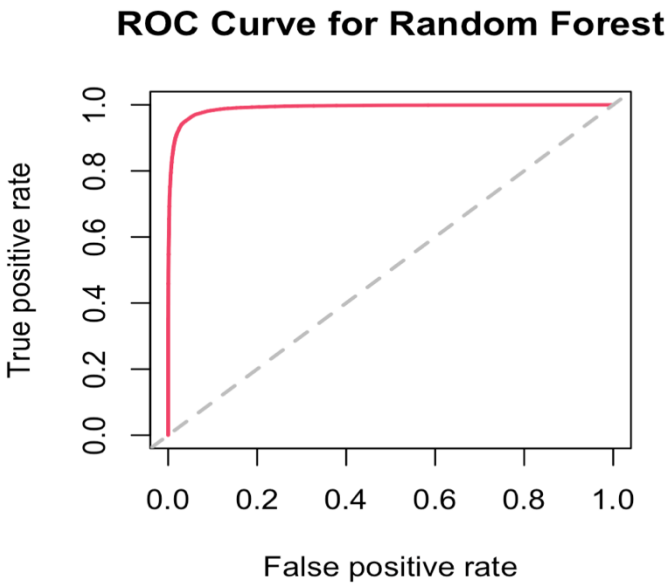

Fig: ROC Curve for Learning Phase
(Dataset1)

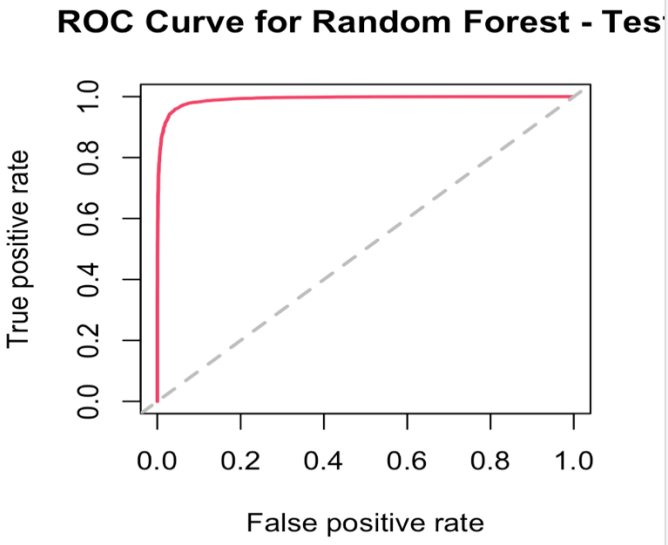

Fig: ROC Curve for the Testing Phase
(Dataset1)

Random Forest Model on Dataset2:

The same procedure is repeated for the smaller variant of the dataset which has lesser number of features. The implementation is as shown below:

Learning Phase of Random Forest Model on Dataset2:

```
> rf_uncor <-randomForest(phishing~.,data=traindata_uncor, ntree=500)
> print(rf_uncor)

Call:
 randomForest(formula = phishing ~ ., data = traindata_uncor,      ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 6

        OOB estimate of  error rate: 5.01%
Confusion matrix:
      0     1 class.error
0 18630  1088  0.05517801
1   967 20366  0.04532883
> mtry_uncor <- tuneRF(traindata_uncor[-37],traindata_uncor$phishing, ntreeTry=500,stepFactor=1.5,improv
e=0.01, trace=TRUE, plot=TRUE)
mtry = 6  OOB error = 5.01%
Searching left ...
mtry = 4        OOB error = 5.72%
-0.1404276 0.01
Searching right ...
mtry = 9        OOB error = 4.78%
0.04616132 0.01
mtry = 13       OOB error = 4.81%
-0.005094244 0.01
> best.m <- mtry_uncor[mtry_uncor[, 2] == min(mtry_uncor[, 2]), 1]
> print(mtry_uncor)
       mtry   OOBError
4.OOB     4 0.05717279
6.OOB     6 0.05013276
9.OOB     9 0.04781857
13.OOB   13 0.04806217
> print(best.m)
[1] 9
```

The best value for mtry is 9 where the OOB error rate is the least. Here as well, we choose ntree as 500.

Confusion Matrix for the Learning Phase:

```
> rf_uncor <-randomForest(phishing~.,data=traindata_uncor, mtry=best.m, importance=TRUE,ntree=500)
> rf_pred_uncor = predict(rf, traindata_uncor[, -37])
Error in eval(predvars, data, env) : object 'qty_equal_url' not found
> rf_pred_uncor = predict(rf_uncor, traindata_uncor[, -37])
> rf_mtab_uncor<-table(traindata_uncor$phishing,rf_pred_uncor)
> rf_cmx_uncor<-caret::confusionMatrix(rf_mtab_uncor)
> rf_cmx_uncor
Confusion Matrix and Statistics

   rf_pred_uncor
       0     1
  0 19696    22
  1    60 21273

               Accuracy : 0.998
                 95% CI : (0.9975, 0.9984)
    No Information Rate : 0.5187
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.996

 Mcnemar's Test P-Value : 4.389e-05

            Sensitivity : 0.9970
            Specificity : 0.9990
         Pos Pred Value : 0.9989
         Neg Pred Value : 0.9972
             Prevalence : 0.4813
         Detection Rate : 0.4798
   Detection Prevalence : 0.4803
      Balanced Accuracy : 0.9980

       'Positive' Class : 0
```

Confusion Matrix for the Testing Phase:

```
> rf_pred_uncor_test = predict(rf_uncor, testdata_uncor[, -37])
> rf_mtab_uncor_test<-table(testdata_uncor$phishing,rf_pred_uncor_test)
> rf_cmx_uncor_test<-caret::confusionMatrix(rf_mtab_uncor_test)
> rf_cmx_uncor_test
Confusion Matrix and Statistics

   rf_pred_uncor_test
       0    1
  0 7863  417
  1  408 8906

               Accuracy : 0.9531
                 95% CI : (0.9499, 0.9562)
    No Information Rate : 0.5299
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9059

 Mcnemar's Test P-Value : 0.7806

            Sensitivity : 0.9507
            Specificity : 0.9553
         Pos Pred Value : 0.9496
         Neg Pred Value : 0.9562
             Prevalence : 0.4701
         Detection Rate : 0.4469
   Detection Prevalence : 0.4706
      Balanced Accuracy : 0.9530

       'Positive' Class : 0
```

Performance Metrics of the Training and Testing phases of the Random Forest Model on Dataset2:

```
> rf_cmx_uncor$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
  9.980025e-01   9.959991e-01   9.975212e-01   9.984110e-01   5.187450e-01   0.000000e+00
 McnemarPValue
  4.389372e-05
> rf_cmx_uncor$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           0.9969629            0.9989669            0.9988843            0.9971875
           Precision               Recall                   F1           Prevalence
           0.9988843            0.9969629            0.9979227            0.4812550
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.4797934            0.4803293            0.9979649
> rf_cmx_uncor_test$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
     0.9531090      0.9058873      0.9498811      0.9561861      0.5298966      0.0000000
 McnemarPValue
     0.7806100
> rf_cmx_uncor_test$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value
           0.9506710            0.9552719            0.9496377            0.9561950
           Precision               Recall                   F1           Prevalence
           0.9496377            0.9506710            0.9501541            0.4701034
      Detection Rate Detection Prevalence    Balanced Accuracy
           0.4469137            0.4706150            0.9529715
>
```

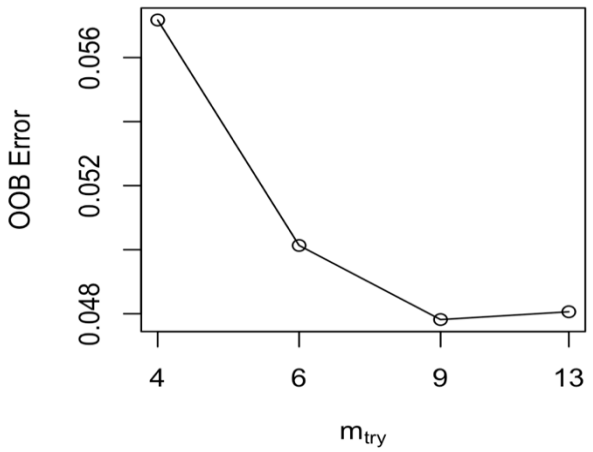

Fig: OOb vs Mtry graph for Dataset2

```
> pred1_uncor=predict(rf_uncor,newdata = traindata_uncor, type = "prob")
> perf_uncor = prediction(pred1_uncor[,2], traindata_uncor$phishing)
> auc_uncor = performance(perf, "auc")
> pred3_uncor = performance(perf_uncor, "tpr","fpr")
> plot(pred3_uncor,main="ROC for Random Forest",col=2,lwd=2)
> abline(a=0,b=1,lwd=2,lty=2,col="gray")
>
>
> pred1_uncor_test=predict(rf_uncor,newdata = testdata_uncor, type = "prob")
> perf_uncor_test = prediction(pred1_uncor_test[,2], testdata_uncor$phishing)
> auc_uncor_test = performance(perf, "auc")
> pred3_uncor_test = performance(perf_uncor_test, "tpr","fpr")
> plot(pred3_uncor_test,main="ROC for Random Forest-Test",col=2,lwd=2)
> abline(a=0,b=1,lwd=2,lty=2,col="gray")
>
```

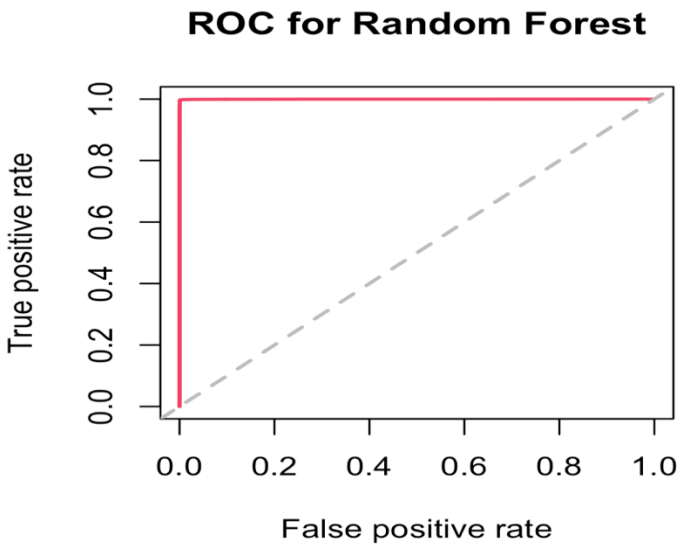

Fig: ROC Curve for the Training Phase (Dataset2)

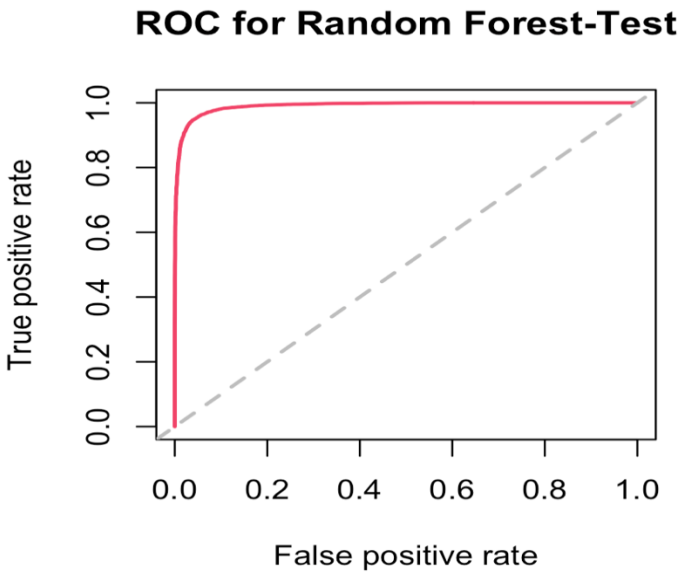

Fig: ROC Curve for the Testing Phase (Dataset2)

We repeat the same process now by setting ntree=100. The implementation is given below:

Training Phase of Random Forest on Dataset1(ntree=100):

```
>
> rf <- randomForest(phishing~., data=traindata, ntree=100)
> print(rf)

Call:
 randomForest(formula = phishing ~ ., data = traindata, ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 8

        OOB estimate of  error rate: 5.13%
Confusion matrix:
      0     1 class.error
0 18717  1001  0.05076580
1  1104 20229  0.05175081
>
> mtry <- tuneRF(traindata[, -79], traindata$phishing, ntreeTry = 100, stepFactor = 1.5, improve = 0.01,
 trace = TRUE, plot = TRUE)
mtry = 8  OOB error = 5.18%
Searching left ...
mtry = 6        OOB error = 5.86%
-0.1327059 0.01
Searching right ...
mtry = 12       OOB error = 4.68%
0.09552941 0.01
mtry = 18       OOB error = 4.5%
0.03850156 0.01
mtry = 27       OOB error = 4.48%
0.00487013 0.01
```

We calculate the best Mtry which has the least OOB error rate as shown below.

```
> best.m <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
> print(mtry)
       mtry   OOBError
6.OOB     6 0.05863438
8.OOB     8 0.05176488
12.OOB   12 0.04681981
18.OOB   18 0.04501717
27.OOB   27 0.04479793
> rf <- randomForest(phishing~., data=traindata, mtry = best.m, ntree=100)
```

Learning Phase Confusion Matrix for Dataset1

```
>
> rf_pred = predict(rf, traindata[, -79])
> rf_mtab <- table(traindata$phishing, rf_pred)
> rf_cmx <- caret::confusionMatrix(rf_mtab)
> rf_cmx
Confusion Matrix and Statistics

   rf_pred
       0     1
  0 19715     3
  1    11 21322

               Accuracy : 0.9997
                 95% CI : (0.9994, 0.9998)
    No Information Rate : 0.5195
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.9993

 Mcnemar's Test P-Value : 0.06137

            Sensitivity : 0.9994
            Specificity : 0.9999
         Pos Pred Value : 0.9998
         Neg Pred Value : 0.9995
             Prevalence : 0.4805
         Detection Rate : 0.4803
   Detection Prevalence : 0.4803
      Balanced Accuracy : 0.9997

       'Positive' Class : 0

> rf_cmx$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
    0.99965896     0.99931687     0.99942786     0.99981354     0.51947577     0.00000000
 McnemarPValue
    0.06136883
```

The performance metrics are as indicated above. The model accuracy indicates a very good fit.

Testing Phase Confusion Matrix for Dataset1:

```
> rf_pred = predict(rf, testdata[, -79])
> rf_mtab <- table(testdata$phishing, rf_pred)
> rf_cmx_test <- caret::confusionMatrix(rf_mtab)
> rf_cmx_test
Confusion Matrix and Statistics

   rf_pred
      0    1
  0 7919  361
  1  385 8929

               Accuracy : 0.9576
                 95% CI : (0.9545, 0.9605)
    No Information Rate : 0.528
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9149

 Mcnemar's Test P-Value : 0.3997

            Sensitivity : 0.9536
            Specificity : 0.9611
         Pos Pred Value : 0.9564
         Neg Pred Value : 0.9587
             Prevalence : 0.4720
         Detection Rate : 0.4501
   Detection Prevalence : 0.4706
      Balanced Accuracy : 0.9574

       'Positive' Class : 0

> rf_cmx_test$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
     0.9575992      0.9149181      0.9545167      0.9605292      0.5280209      0.0000000
 McnemarPValue
     0.3997377
```

We repeat the process over the smaller version of the dataset.

Learning Phase for Dataset2:

```
> rf_uncor <- randomForest(phishing~., data=traindata_uncor, ntree=100)
> mtry_uncor <- tuneRF(traindata_uncor[, -37], traindata_uncor$phishing, ntreeTry = 100, stepFactor = 1.
5, improve = 0.01, trace = TRUE, plot = TRUE)
mtry = 6  OOB error = 5.19%
Searching left ...
mtry = 4        OOB error = 5.77%
-0.112729 0.01
Searching right ...
mtry = 9        OOB error = 4.97%
0.04133396 0.01
mtry = 13       OOB error = 4.95%
0.003919647 0.01
```

Learning Phase Confusion Matrix for Dataset2:

```
> rf_uncor <- randomForest(phishing~., data=traindata_uncor, mtry = best.m, ntree=100)
> rf_pred_uncor = predict(rf_uncor, traindata_uncor[,-37])
> rf_mtab_uncor <- table(traindata_uncor$phishing, rf_pred_uncor)
> rf_cmx_uncor <- caret::confusionMatrix(rf_mtab_uncor)
> rf_cmx_uncor
Confusion Matrix and Statistics

   rf_pred_uncor
       0     1
  0 19703    15
  1    39 21294

               Accuracy : 0.9987
                 95% CI : (0.9983, 0.999)
    No Information Rate : 0.5191
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9974

 Mcnemar's Test P-Value : 0.001749

            Sensitivity : 0.9980
            Specificity : 0.9993
         Pos Pred Value : 0.9992
         Neg Pred Value : 0.9982
             Prevalence : 0.4809
         Detection Rate : 0.4800
   Detection Prevalence : 0.4803
      Balanced Accuracy : 0.9987

       'Positive' Class : 0

> rf_cmx_uncor$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
   0.998684563    0.997365169    0.998283984    0.999011654    0.519086015    0.000000000
 McnemarPValue
   0.001748637
```

The performance indicated above implies a very good fit of the model.

Testing Phase Confusion Matrix for Dataset2:

```
> rf_pred_uncor_test = predict(rf_uncor, testdata_uncor[,-37])
> rf_mtab_uncor_test <- table(testdata_uncor$phishing, rf_pred_uncor_test)
> rf_cmx_uncor_test <- caret::confusionMatrix(rf_mtab_uncor_test)
> rf_cmx_uncor_test
Confusion Matrix and Statistics

   rf_pred_uncor_test
      0    1
  0 7856  424
  1  411 8903

               Accuracy : 0.9525
                 95% CI : (0.9493, 0.9556)
    No Information Rate : 0.5301
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9047

 Mcnemar's Test P-Value : 0.6779

            Sensitivity : 0.9503
            Specificity : 0.9545
         Pos Pred Value : 0.9488
         Neg Pred Value : 0.9559
             Prevalence : 0.4699
         Detection Rate : 0.4465
   Detection Prevalence : 0.4706
      Balanced Accuracy : 0.9524

       'Positive' Class : 0

> rf_cmx_uncor_test$overall
      Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull AccuracyPValue
     0.9525406      0.9047440      0.9492949      0.9556358      0.5301239      0.0000000
 McnemarPValue
     0.6779390
```

The performance metrics are indicated above. When we contrast the accuracy of the Random Forest Model run using ntree=500 and ntree=100, we can conclude that there is no significant increase in terms of accuracy or other performance metrics. Hence we conclude the Random Forest Model and note down the accuracy.

Section 3 – Stacking with 5-fold Cross Validation

We club the ensemble methods Stacking and Cross Validation to boost the performance of the classifiers. Here we first split the dataset into training and testing partitions. We then use the training partition to run 3 different classifiers which is Logistic Regression, KNN and Decision Trees to train the partition. The method trainControl uses cross validation in training the best performing model. After this, we club the results of the 3 classifiers into a dataframe and again train the entire result using Decision Trees.

```
library(parallel)
library(doParallel)
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
fitControl <- trainControl(method = "cv",number = 5, allowParallel = TRUE)
```

```
> inTrain = createDataPartition(ncpdata.new$phishing, p = 3 / 4)[[1]]
> training = ncpdata.new[inTrain,]
> testing = ncpdata.new[-inTrain,]
```

```
system.time(modelFitLR <- train(phishing ~ ., method="glm",data=training, trControl = fitControl))
system.time(modelFitKNN <- train(phishing ~ ., method="knn",data=training, trControl = fitControl))
system.time(modelFitRpart <- train(phishing ~ ., method="rpart",data=training, trControl = fitControl))
```

```
>
> predLRtrain <- predict(modelFitLR,newdata=training)
> predKNNtrain <- predict(modelFitKNN,newdata=training)
> predRparttrain <- predict(modelFitRpart,newdata=training)
>
> predDF <- data.frame(predLRtrain, predKNNtrain, predRparttrain, phishing = training$phishing, stringsA
sFactors = F)
>
> modelStack <- train(phishing ~ ., data = predDF, method = "rpart")
>

> modelStack
CART

43985 samples
    3 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 43985, 43985, 43985, 43985, 43985, 43985, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.01104815  0.9132445  0.8262215
  0.01209581  0.9080840  0.8157388
  0.79094243  0.7634119  0.5105321

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.01104815.
```

The testing partition is also predicted with the 3 classifiers and clubbed as a single data frame. Finally, the result of the final classifier on the union of the 3 classifier outputs is used to predict the testing data frame obtained.

```
> require(tidyverse)
> names(predDF)[names(predDF) == "predLRtrain"] <- "predLR"
> names(predDF)[names(predDF) == "predKNNtrain"] <- "predKNN"
> names(predDF)[names(predDF) == "predRparttrain"] <- "predRpart"
```

```
> predLR <- predict(modelFitLR,newdata=testing)
> predKNN <- predict(modelFitKNN, newdata = testing)
> predRpart <- predict(modelFitRpart, newdata = testing)
```

```
> testPredLevelOne <- data.frame(predLR, predKNN, predRpart, phishing = testing$phishing, stringsAsFacto
rs = F)
> confusionMatrix(predLR, testing$phishing)$overall[1]
 Accuracy
0.9027967
>
> confusionMatrix(predKNN, testing$phishing)$overall[1]
 Accuracy
0.8531378
> confusionMatrix(predRpart, testing$phishing)$overall[1]
Accuracy
0.861869
```

```
> modelStack <- train(phishing ~ ., data = predDF, method = "rpart")
> combPred <- predict(modelStack, testPredLevelOne)
>
> confusionMatrix(combPred, testing$phishing)$overall[1]
 Accuracy
0.9124829
```

As seen above, the accuracy is very good compared to the individual classifier accuracies.

The same procedure is done with the smaller dataset and the accuracy is measured.

```
> library(parallel)
> library(doParallel)
> cluster <- makeCluster(detectCores() - 1)
> registerDoParallel(cluster)
> fitControl <- trainControl(method = "cv",number = 5, allowParallel = TRUE)
>
> inTrain_uncor = createDataPartition(ncpdata_uncor$phishing, p = 3 / 4)[[1]]
> training_uncor = ncpdata_uncor[inTrain_uncor, ]
> testing_uncor = ncpdata_uncor[-inTrain_uncor, ]
>
> system.time(modelFitLR_uncor <- train(phishing ~ ., method="glm",data=training_uncor, trControl = fitC
ontrol))
   user  system elapsed
  2.659   0.349  12.123
```

```
> system.time(modelFitKNN_uncor <- train(phishing ~ ., method="knn",data=training_uncor, trControl = fit
Control))
   user  system elapsed
  1.171   0.287  46.927
> system.time(modelFitRpart_uncor <- train(phishing ~ ., method="rpart",data=training_uncor, trControl =
 fitControl))
   user  system elapsed
  7.499   0.217   8.619
>
> predLR_uncor <- predict(modelFitLR_uncor, newdata = training_uncor)
> predKNN_uncor <- predict(modelFitKNN_uncor, newdata = training_uncor)
> predRpart_uncor <- predict(modelFitRpart_uncor, newdata = training_uncor)
>
> predDF_uncor <- data.frame(predLR_uncor, predKNN_uncor, predRpart_uncor, phishing=training_uncor$phish
ing, stringsAsFactors = F)
>
> modelStack_uncor <- train(phishing ~ ., data=predDF_uncor, method="rpart")
> predLR_uncor <- predict(modelFitLR_uncor, newdata = testing_uncor)
> predKNN_uncor <- predict(modelFitKNN_uncor, newdata = testing_uncor)
> predRpart_uncor <- predict(modelFitRpart_uncor, newdata = testing_uncor)
>
> testPredLevelOne_uncor <- data.frame(predLR_uncor, predKNN_uncor, predRpart_uncor, phishing=testing_un
cor$phishing, stringsAsFactors = F)
> confusionMatrix(predLR_uncor, testing_uncor$phishing)$overall[[1]]
[1] 0.8824011
> confusionMatrix(predKNN_uncor, testing_uncor$phishing)$overall[[1]]
[1] 0.8400409
> confusionMatrix(predRpart_uncor, testing_uncor$phishing)$overall[[1]]
[1] 0.85
>
> combPred_uncor <- predict(modelStack_uncor, testPredLevelOne_uncor)
>
> confusionMatrix(combPred_uncor, testing_uncor$phishing)$overall[[1]]
[1] 0.9005457
```

The result of the stacking ensemble method run on the second dataset is also very good compared to the individual classifier accuracies.

Section 4 – Bagging with 5-fold Cross Validation

The next ensemble method we see is bagging. We use “parallel” and “doParallel” packages in R to use parallel computation. The below implementation shows bagging applied on our 2 datasets using “treebag” method.

Learning Phase Bagging on Dataset1:

```
> require(parallel)
> require(caret)
> require(doParallel)
>
>
>
> cl<-makePSOCKcluster(5)
> registerDoParallel(cl)
> start.time<-proc.time()
> fitControl <- trainControl(method = "cv",number = 5, allowParallel = TRUE)
> train.bagg <- train(as.factor(phishing) ~ .,
+                     data=traindata,
+                     method="treebag",
+                     trControl=fitControl,
+                     importance=TRUE)
> train.bagg
Bagged CART

41051 samples
   78 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 32841, 32842, 32840, 32841, 32840
Resampling results:

  Accuracy   Kappa
  0.9499402  0.89969
```

Learning Phase Confusion Matrix on Dataset1:

```
> bagg.classTrain <-  predict(train.bagg,
+                             type="raw")
> confusionMatrix(traindata$phishing,bagg.classTrain)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 19700    18
         1    12 21321

               Accuracy : 0.9993
                 95% CI : (0.999, 0.9995)
    No Information Rate : 0.5198
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9985

 Mcnemar's Test P-Value : 0.3613

            Sensitivity : 0.9994
            Specificity : 0.9992
         Pos Pred Value : 0.9991
         Neg Pred Value : 0.9994
             Prevalence : 0.4802
         Detection Rate : 0.4799
   Detection Prevalence : 0.4803
      Balanced Accuracy : 0.9993

       'Positive' Class : 0
```

Testing Phase Confusion Matrix on Dataset1:

```
> bagg.classTest <-  predict(train.bagg,
+                            newdata = testdata,
+                            type="raw")
> confusionMatrix(testdata$phishing,bagg.classTest)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7883  397
         1  420 8894

               Accuracy : 0.9536
                 95% CI : (0.9504, 0.9566)
    No Information Rate : 0.5281
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9068

 Mcnemar's Test P-Value : 0.4415

            Sensitivity : 0.9494
            Specificity : 0.9573
         Pos Pred Value : 0.9521
         Neg Pred Value : 0.9549
             Prevalence : 0.4719
         Detection Rate : 0.4481
   Detection Prevalence : 0.4706
      Balanced Accuracy : 0.9533

       'Positive' Class : 0
```

Learning Phase Bagging on Dataset2:

```
> train.bagg_uncor <- train(as.factor(phishing) ~ .,
+                     data=traindata_uncor,
+                     method="treebag",
+                     trControl=fitControl,
+                     importance=TRUE)
> train.bagg_uncor
Bagged CART

41051 samples
   36 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 32841, 32840, 32842, 32840, 32841
Resampling results:

  Accuracy   Kappa
  0.9433145  0.88642
```

Learning Phase Confusion Matrix on Dataset2:

```
> bagg.classTrain_uncor <-  predict(train.bagg_uncor,
+                             type="raw")
> confusionMatrix(traindata_uncor$phishing,bagg.classTrain_uncor)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 19683    35
         1    32 21301

               Accuracy : 0.9984
                 95% CI : (0.9979, 0.9987)
    No Information Rate : 0.5197
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9967

 Mcnemar's Test P-Value : 0.807

            Sensitivity : 0.9984
            Specificity : 0.9984
         Pos Pred Value : 0.9982
         Neg Pred Value : 0.9985
             Prevalence : 0.4803
         Detection Rate : 0.4795
   Detection Prevalence : 0.4803
      Balanced Accuracy : 0.9984

       'Positive' Class : 0
```

Testing Phase confusion matrix on Dataset2:

```
> bagg.classTest_uncor <-  predict(train.bagg_uncor,
+                             newdata = testdata_uncor,
+                             type="raw")
> confusionMatrix(testdata_uncor$phishing,bagg.classTest_uncor)
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 7831  449
         1  461 8853

               Accuracy : 0.9483
                 95% CI : (0.9449, 0.9515)
    No Information Rate : 0.5287
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8962

 Mcnemar's Test P-Value : 0.7154

            Sensitivity : 0.9444
            Specificity : 0.9517
         Pos Pred Value : 0.9458
         Neg Pred Value : 0.9505
             Prevalence : 0.4713
         Detection Rate : 0.4451
   Detection Prevalence : 0.4706
      Balanced Accuracy : 0.9481

       'Positive' Class : 0
```

The bagging model achieved excellent accuracy in the testing phases on both the datasets. It also executes fast on large datasets.

Section 5 – Estimating Variance of Classification Models

We can estimate the variance of the classifiers by running the models multiple times and calculating the variance of the mean of accuracies. For this, we split the dataset into train and test datasets in 90:10 ratio where 90 percent is reserved for training and 10 percent is reserved for testing.

We calculate approximate variances for logistic regression and KNN classifiers by the below implementation.

Variation Estimation for Logistic Regression on Dataset1:

```
> fpr <- NULL
> fnr <- NULL
> pbar <- create_progress_bar('text')
> k <- 10

> for(i in 1:k){
+     smp_size <- floor(0.90 * nrow(ncpdata.new))
+     index <- sample(seq_len(nrow(ncpdata.new)),size=smp_size)
+     train_cv <- ncpdata.new[index, ]
+     test_cv <- ncpdata.new[-index, ]
+     model <- glm(phishing~.,family=binomial,data=train_cv)
+     results_prob <- predict(model,test_cv,family=binomial)
+     results <- as.factor(ifelse(results_prob < 0.5,0,1))
+     answers <- as.factor(test_cv$phishing)
+     misClasificError <- mean(answers != results)
+     cm <- confusionMatrix(answers, results)
+     acc[i] <- cm$overall[[1]]
+     fpr[i] <- cm$table[2]/(nrow(ncpdata.new)-smp_size)
+     fnr[i] <- cm$table[3]/(nrow(ncpdata.new)-smp_size)
+     pbar$step()
+ }

> mean(acc)
[1] 0.8924638
```

Variation Estimation for Logistic Regression on Dataset1:

```
> fpr_uncor <- NULL
> fnr_uncor <- NULL
> pbar <- create_progress_bar('text')
> k <- 10
> pbar$init(k)
  |                                                                              |   0%> acc_u
ncor <- NULL
> set.seed(43)
```

Variation Estimation for Logistic Regression on Dataset2:

```
> for(i in 1:k){
+     smp_size <- floor(0.90 * nrow(ncpdata_uncor))
+     index <- sample(seq_len(nrow(ncpdata_uncor)),size=smp_size)
+     train_cv_uncor <- ncpdata_uncor[index, ]
+     test_cv_uncor <- ncpdata_uncor[-index, ]
+     model <- glm(phishing~.,family=binomial,data=train_cv_uncor)
+     results_prob <- predict(model,test_cv_uncor,family=binomial)
+     results <- as.factor(ifelse(results_prob < 0.5,0,1))
+     answers <- as.factor(test_cv_uncor$phishing)
+     cm_uncor <- confusionMatrix(answers, results)
+     acc_uncor[i] <- cm_uncor$overall[[1]]
+     fpr_uncor[i] <- cm_uncor$table[2]/(nrow(ncpdata_uncor)-smp_size)
+     fnr_uncor[i] <- cm_uncor$table[3]/(nrow(ncpdata_uncor)-smp_size)
+     pbar$step()
+ }

> mean(acc_uncor)
[1] 0.868474

> var(acc)
[1] 8.188084e-06
>
> var(acc_uncor)
[1] 1.584836e-05
```

Plotting accuracies for the 2 datasets.

```
> par(mfcol=c(1,1))
> hist(acc,xlab='Accuracy',ylab='Freq',
+      col='cyan',border='blue',density=30)
```

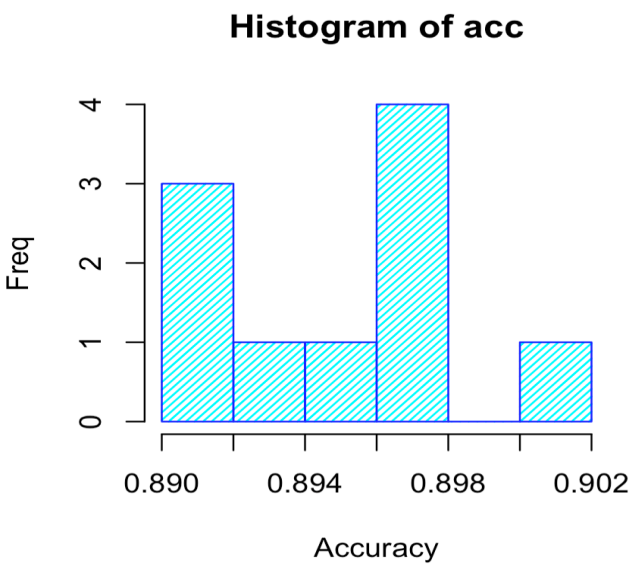

Fig: Accuracy graph for Dataset1(LR)

```
> par(mfcol=c(1,1))
> hist(acc_uncor,xlab='Accuracy',ylab='Freq',
+      col='cyan',border='blue',density=30)
```

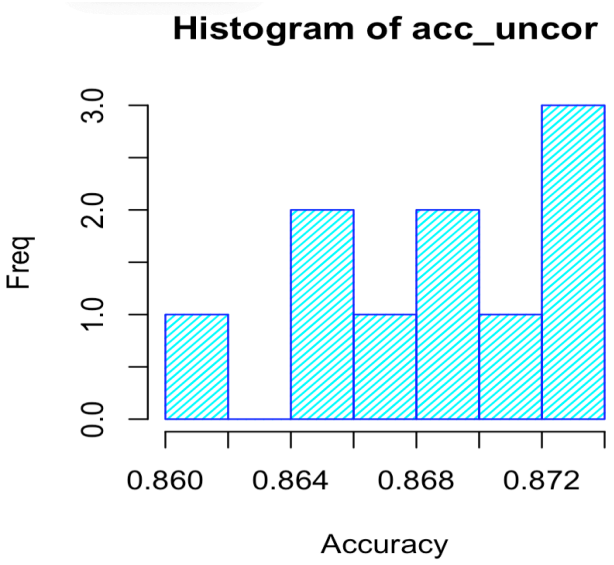

Fig: Accuracy graph for Dataset2(LR)

We repeat the same procedure but for the KNN classifier.

Variation Estimation for KNN on Dataset1:

```
> fpr_dt <- NULL
> fnr_dt <- NULL
> pbar <- create_progress_bar('text')
> k <- 10
> pbar$init(k)
  |                                                                                                                      |   0%> acc_d
t <- NULL
> set.seed(43)
> penalty.matrix <- matrix(c(0,1,10,0), byrow=TRUE, nrow=2)
>
> for(i in 1:k){
+     smp_size <- floor(0.90 * nrow(ncpdata.new))
+     index <- sample(seq_len(nrow(ncpdata.new)),size=smp_size)
+     train_cv <- ncpdata.new[index, ]
+     test_cv <- ncpdata.new[-index, ]
+     tree <- rpart(phishing~., data=train_cv, parms = list(loss = penalty.matrix),method = "class")
+     results_prob <- predict(tree, newdata = test_cv[-79], type ='class')
+     cm_dt <- confusionMatrix(table(test_cv$phishing, results_prob))
+     acc_dt[i] <- cm_dt $overall[[1]]
+     fpr_dt[i] <- cm_dt $table[2]/(nrow(ncpdata.new)-smp_size)
+     fnr_dt[i] <- cm_dt $table[3]/(nrow(ncpdata.new)-smp_size)
+     pbar$step()
+ }
  |=================================================================================================| 100%
>
> mean(acc_dt)
[1] 0.8558738
>
> var(acc_dt)
[1] 2.74113e-05
```

Variation Estimation for KNN on Dataset2:

```
> fpr_dt_uncor <- NULL
> fnr_dt_uncor <- NULL
> pbar <- create_progress_bar('text')
> k <- 10
> pbar$init(k)
  |                                                                                  |   0%> acc_d
t_uncor <- NULL
> set.seed(43)
> penalty.matrix <- matrix(c(0,1,10,0), byrow=TRUE, nrow=2)
> for(i in 1:k){
+     smp_size <- floor(0.90 * nrow(ncpdata_uncor))
+     index <- sample(seq_len(nrow(ncpdata_uncor)),size=smp_size)
+     train_cv_uncor <- ncpdata_uncor[index, ]
+     test_cv_uncor <- ncpdata_uncor[-index, ]
+     tree <- rpart(phishing~., data=train_cv_uncor, parms = list(loss = penalty.matrix),method = "clas
s")
+     results_prob <- predict(tree, newdata = test_cv_uncor[-37], type ='class')
+     cm_dt_uncor <- confusionMatrix(table(test_cv_uncor$phishing, results_prob))
+     acc_dt_uncor[i] <- cm_dt_uncor$overall[[1]]
+     fpr_dt_uncor[i] <- cm_dt_uncor$table[2]/(nrow(ncpdata_uncor)-smp_size)
+     fnr_dt_uncor[i] <- cm_dt_uncor$table[3]/(nrow(ncpdata_uncor)-smp_size)
+     pbar$step()
+ }
  |==================================================================================| 100%
>
> mean(acc_dt_uncor)
[1] 0.8023359
>
> var(acc_dt_uncor)
[1] 2.360619e-05
```

Plotting accuracies for the 2 datasets.

```
> par(mfcol=c(1,1))
> hist(acc_dt,xlab='Accuracy',ylab='Freq',
+ col='cyan',border='blue',density=30)
>
> par(mfcol=c(1,1))
> hist(acc_dt_uncor,xlab='Accuracy',ylab='Freq',
+ col='cyan',border='blue',density=30)
> 
```

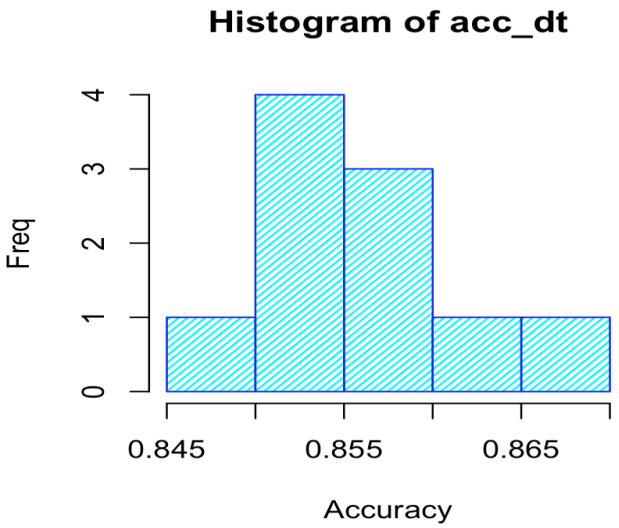

Fig: Accuracy graph for Dataset1(KNN)

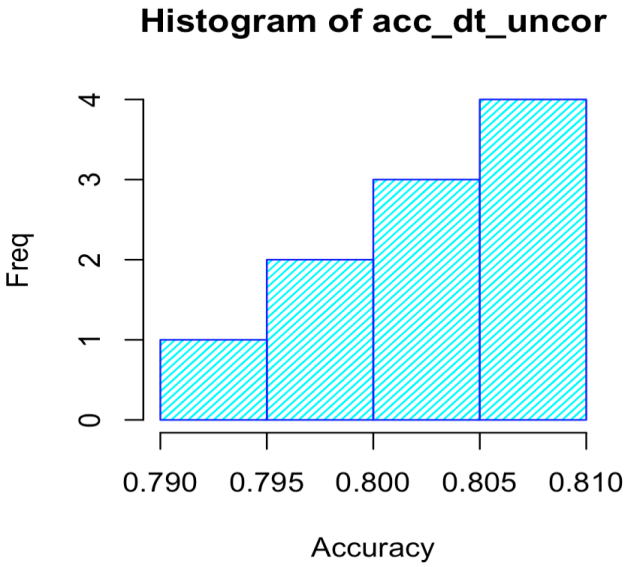

Fig: Accuracy graph for Dataset2(KNN)

We observe from running the above code that the Logistic Regression and KNN Classification models on our dataset produce very little variance which indicates a good fit on our models.

Section 6 - Summary of Ensemble Models

We have run Random Forest and Stacking with Cross Validation on our 2 variants of the Dataset. The summary of testing accuracies is tabulated below:

Model	Dataset1	Dataset2
	Testing Accuracy	Testing Accuracy
Random Forest – ntree=500	95.78	95.31
Random Forest – ntree=100	95.76	95.25
Stacking with CV	91.24	90.05
Bagging with CV	95.36	94.83

A similar conclusion can be draw from these observations as well. The larger dataset has higher accuracy when compared to the dataset with lesser observations.

Random Forest performed the best when compared to Stacking and Bagging. The individual classifier models have very low variance as indicated by the variance of accuracies above. This implies that the models are not overfitting and models were able to ideally train on the datasets.

PS: Thank You Note

I sincerely thank Professor Dr. Raman for giving me an opportunity to be a part of this course which has been an amazing journey and a great learning experience inclined towards the industry practices.