Contents

Task 1: Dataset selection 3
Pre-processing & Attribute information 3
Missing value validation 4
Randomizing the rows 4
Understanding the dataset 4
Feature selection 6
Task 2: Repeated K-fold cross-validation 8
Splitting Training & testing data 8
Repeated k-fold Cross Validation 8
Task 3: Bagging type classifier 10
Confusion matrix 13
Precision & Recall, ROC, AUC 14
Task 4: Stacking Type Classifier 16
Stacking models 19
Precision & Recall 20
ROC & AUC 21
Task 5: Measure Performance 22
5.1 Confusion matrix 22
5.1.1 Confusion matrix for bagging classifier 22
5.1.2 Confusion matrix for stacking classifier 23
5.2 Precision vs. Recall estimation 25
5.2.1 Precision & recall for bagging classifier 25
5.2.2 Precision & recall for stacking classifier 27
5.3 Accuracy Estimation 27
5.4 ROC(receiver operating characteristic curve) & RAUC (receiver under the curve area) 28
5.4.1 ROC & RAUC for bagging classifier 29
5.4.2 ROC & RUAC for bagging classifier 30
5.5 Training time 33
5.5.1 Training time forbagging classifier 33
5.5.2 Training time for stacking classifier 33
5.6 Testing time 34
5.6.1 Testing time for bagging classifier 34

5.6.2 Testing time for stacking classifier........ 35
Conclusion 36
References 37
Appendix – Full code........ 38

Task 1: Dataset selection

Dataset name	Seeds
Dataset url	https://archive.ics.uci.edu/ml/datasets/seeds

The selected dataset was called 'Seeds' with Multivariate characteristic. The dataset comprises kernels belonging to three different varieties of wheat: Kama and Rosa, 70 elements each. To construct the data, seven geometric parameters of wheat kernels were measured. Each class of wheat containing equal number of rows, which enables to train the model with high accuracy.

The dataset states that high quality visualization of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology. The images were recorded on 13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.

Purposefully, The dataset only the Kama and Rosa instances are used in this assignment. Since certain function (such as : ROCR ROC, CaretStack) expects an binary predictions, only first two class variables are taken into training and testing.

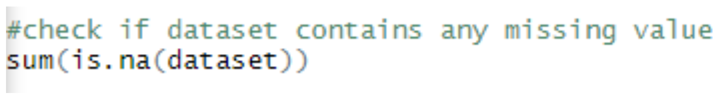
Pre-processing & Attribute information

Initially, the dataset was converted to .csv that can be imported in to RStudio. The dataset contains Numerical continuous variable and class variable. Following are the attribute information of the selected dataset which is represented in dataset.

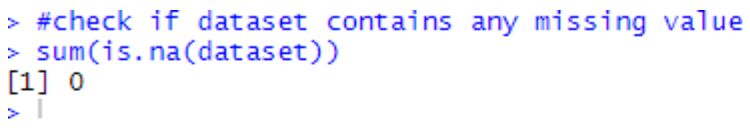
Attribute name in dataset	**Attribute information**	**Type**
V1	area A	Numerical continuous variable
V2	perimeter P,	Numerical continuous variable
V3	compactness C = 4*pi*A/P^2,	Numerical continuous variable
V4	length of kernel	Numerical continuous variable
V5	width of kernel	Numerical continuous variable
V6	asymmetry coefficient	Numerical continuous variable
V7	length of kernel groove	Numerical continuous variable
Wheat	3 Type of wheat (Kama, Rosa)	Categorical ordinal variable

Missing value validation

The missing data in the dataset can compromise the result. Therefore the check was preformed using the following command. The results states there were no missing data.



```
#check if dataset contains any missing value
sum(is.na(dataset))
```

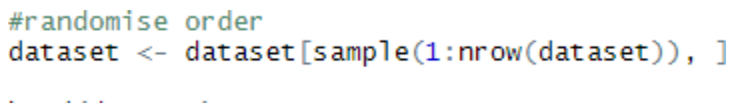


```
> #check if dataset contains any missing value
> sum(is.na(dataset))
[1] 0
> |
```

Randomizing the rows

In selected dataset, rows are ordered by wheat types in order. In other words, first 70 instances are kama instances and last 70 instances are Rosa type of wheat. When splitting the data for training and testing the training data will be having all types of first 2 types. This will lead the model to be more biased for first 2 categories and missing first 2 type of instance in testing data.

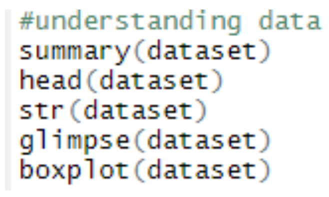
In order to solve this, the instances of the datasets were shuffled / reordered randomly. This enables the training and testing data to have generous amount of all the types of Wheat.



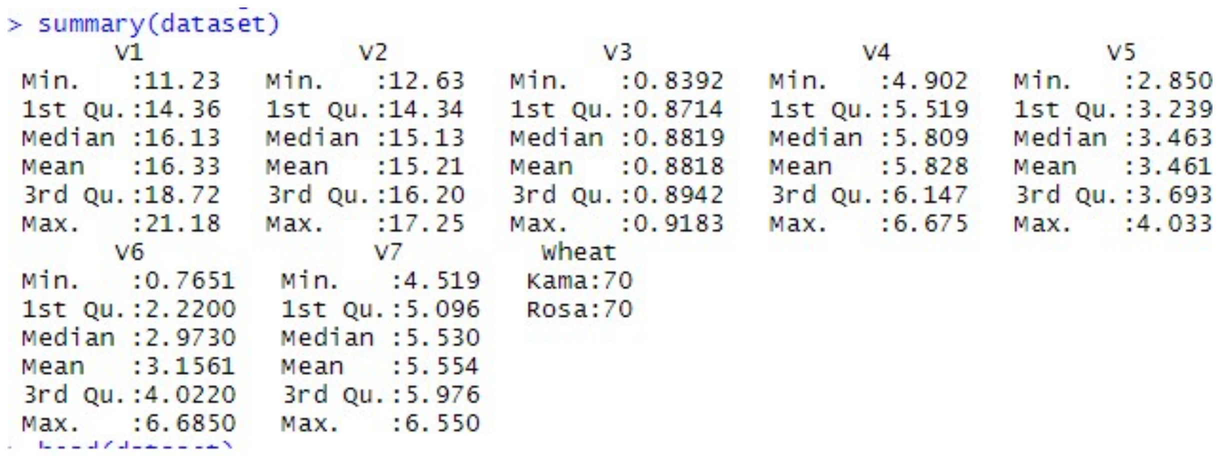
```
#randomise order
dataset <- dataset[sample(1:nrow(dataset)), ]
```

Understanding the dataset

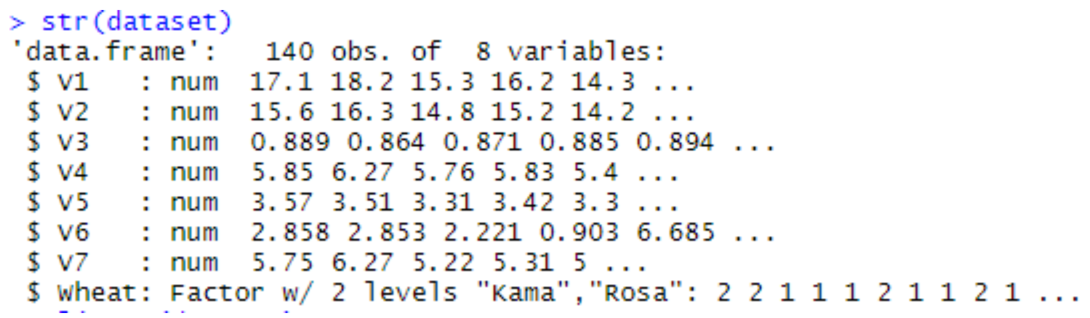
The attributes of the datasets were normalized and values were not distributed. Therefore normalization operation was not required.



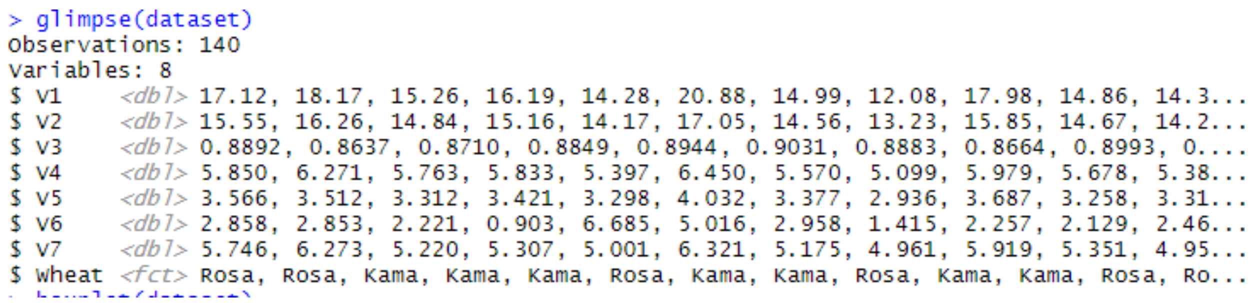
```
#understanding data
summary(dataset)
head(dataset)
str(dataset)
glimpse(dataset)
boxplot(dataset)
```



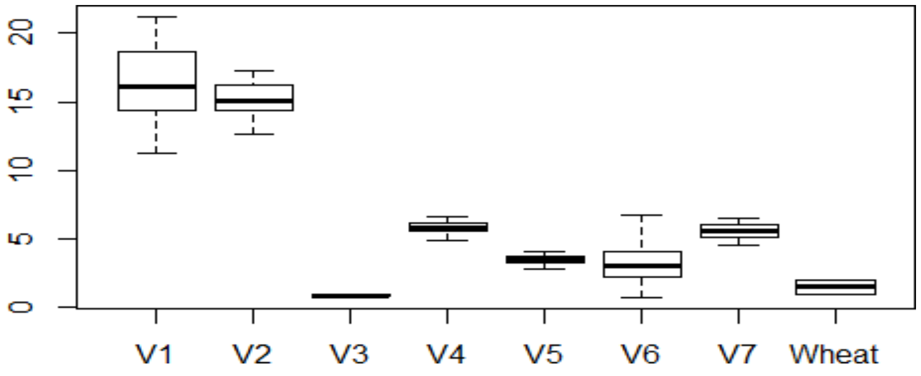
```
> summary(dataset)
       V1              V2              V3               V4              V5       
 Min.   :11.23   Min.   :12.63   Min.   :0.8392   Min.   :4.902   Min.   :2.850  
 1st Qu.:14.36   1st Qu.:14.34   1st Qu.:0.8714   1st Qu.:5.519   1st Qu.:3.239  
 Median :16.13   Median :15.13   Median :0.8819   Median :5.809   Median :3.463  
 Mean   :16.33   Mean   :15.21   Mean   :0.8818   Mean   :5.828   Mean   :3.461  
 3rd Qu.:18.72   3rd Qu.:16.20   3rd Qu.:0.8942   3rd Qu.:6.147   3rd Qu.:3.693  
 Max.   :21.18   Max.   :17.25   Max.   :0.9183   Max.   :6.675   Max.   :4.033  
       V6               V7          Wheat   
 Min.   :0.7651   Min.   :4.519   Kama:70  
 1st Qu.:2.2200   1st Qu.:5.096   Rosa:70  
 Median :2.9730   Median :5.530            
 Mean   :3.1561   Mean   :5.554            
 3rd Qu.:4.0220   3rd Qu.:5.976            
 Max.   :6.6850   Max.   :6.550            
```



```
> str(dataset)
'data.frame':	140 obs. of  8 variables:
 $ V1   : num  17.1 18.2 15.3 16.2 14.3 ...
 $ V2   : num  15.6 16.3 14.8 15.2 14.2 ...
 $ V3   : num  0.889 0.864 0.871 0.885 0.894 ...
 $ V4   : num  5.85 6.27 5.76 5.83 5.4 ...
 $ V5   : num  3.57 3.51 3.31 3.42 3.3 ...
 $ V6   : num  2.858 2.853 2.221 0.903 6.685 ...
 $ V7   : num  5.75 6.27 5.22 5.31 5 ...
 $ Wheat: Factor w/ 2 levels "Kama","Rosa": 2 2 1 1 1 2 1 1 2 1 ...
```



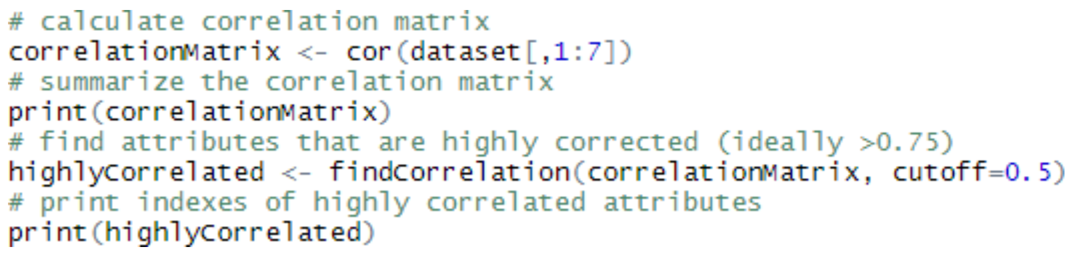
```
> glimpse(dataset)
Observations: 140
Variables: 8
$ V1    <dbl> 17.12, 18.17, 15.26, 16.19, 14.28, 20.88, 14.99, 12.08, 17.98, 14.86, 14.3...
$ V2    <dbl> 15.55, 16.26, 14.84, 15.16, 14.17, 17.05, 14.56, 13.23, 15.85, 14.67, 14.2...
$ V3    <dbl> 0.8892, 0.8637, 0.8710, 0.8849, 0.8944, 0.9031, 0.8883, 0.8664, 0.8993, 0....
$ V4    <dbl> 5.850, 6.271, 5.763, 5.833, 5.397, 6.450, 5.570, 5.099, 5.979, 5.678, 5.38...
$ V5    <dbl> 3.566, 3.512, 3.312, 3.421, 3.298, 4.032, 3.377, 2.936, 3.687, 3.258, 3.31...
$ V6    <dbl> 2.858, 2.853, 2.221, 0.903, 6.685, 5.016, 2.958, 1.415, 2.257, 2.129, 2.46...
$ V7    <dbl> 5.746, 6.273, 5.220, 5.307, 5.001, 6.321, 5.175, 4.961, 5.919, 5.351, 4.95...
$ Wheat <fct> Rosa, Rosa, Kama, Kama, Kama, Rosa, Kama, Kama, Rosa, Kama, Kama, Rosa, Ro...
```



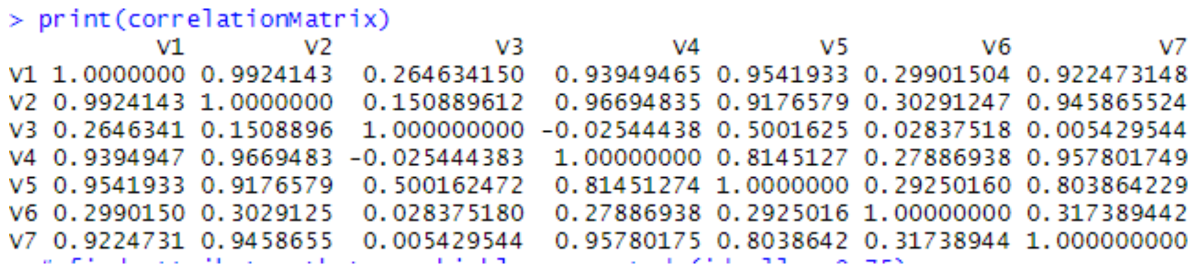
Feature selection

Dataset contains attributes that are highly correlated with each other. It is found that models perform better if highly correlated attributes are removed. Feature selection helps to build models with different subsets of a dataset and identify attributes that are and are not required to build an accurate model.

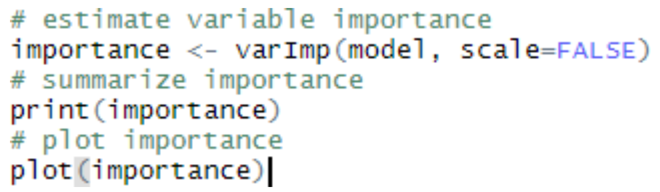
Methods like decision trees have a built-in mechanism to report on variable importance. Caret R library assists to find correlation which will analyze a correlation matrix of attributes that can be removed without harming the accuracy of the model. The importance of features can be estimated from data by building a model.



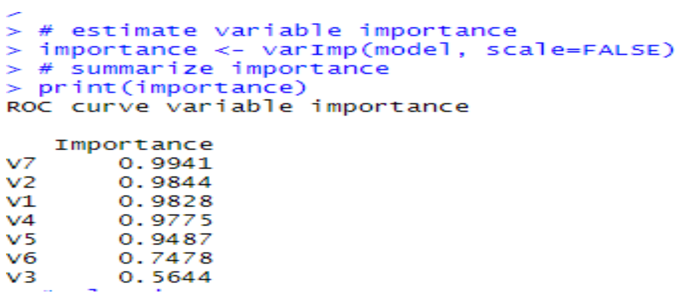
```
# calculate correlation matrix
correlationMatrix <- cor(dataset[,1:7])
# summarize the correlation matrix
print(correlationMatrix)
# find attributes that are highly corrected (ideally >0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.5)
# print indexes of highly correlated attributes
print(highlyCorrelated)
```



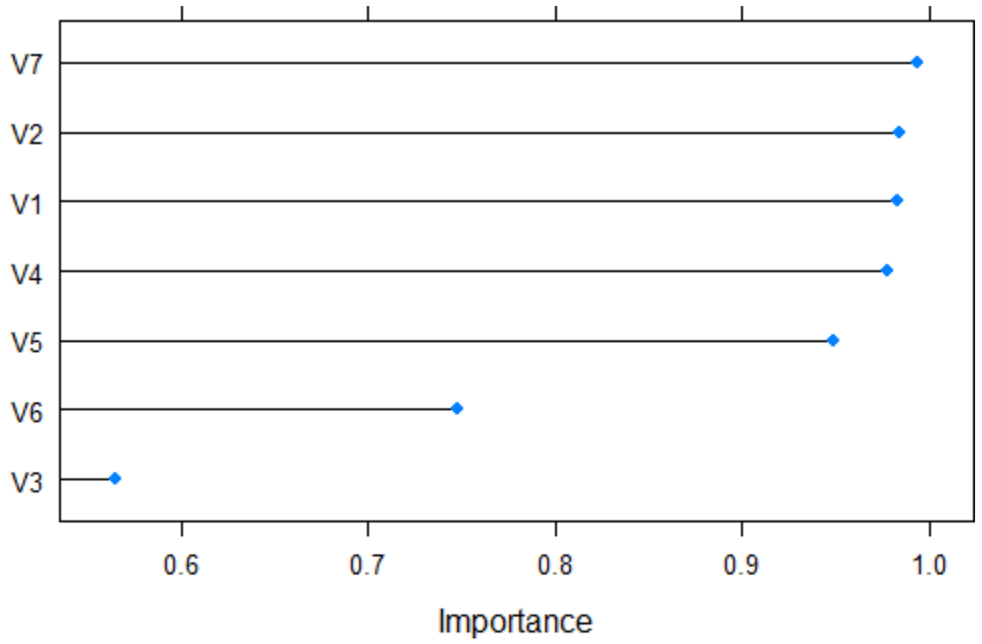
```
> print(correlationMatrix)
          V1        V2          V3          V4        V5         V6          V7
V1 1.0000000 0.9924143  0.264634150  0.93949465 0.9541933 0.29901504 0.922473148
V2 0.9924143 1.0000000  0.150889612  0.96694835 0.9176579 0.30291247 0.945865524
V3 0.2646341 0.1508896  1.000000000 -0.02544438 0.5001625 0.02837518 0.005429544
V4 0.9394947 0.9669483 -0.025444383  1.00000000 0.8145127 0.27886938 0.957801749
V5 0.9541933 0.9176579  0.500162472  0.81451274 1.0000000 0.29250160 0.803864229
V6 0.2990150 0.3029125  0.028375180  0.27886938 0.2925016 1.00000000 0.317389442
V7 0.9224731 0.9458655  0.005429544  0.95780175 0.8038642 0.31738944 1.000000000
```



```
# estimate variable importance
importance <- varImp(model, scale=FALSE)
# summarize importance
print(importance)
# plot importance
plot(importance)
```



```
> # estimate variable importance
> importance <- varImp(model, scale=FALSE)
> # summarize importance
> print(importance)
ROC curve variable importance

   Importance
V7     0.9941
V2     0.9844
V1     0.9828
V4     0.9775
V5     0.9487
V6     0.7478
V3     0.5644
```



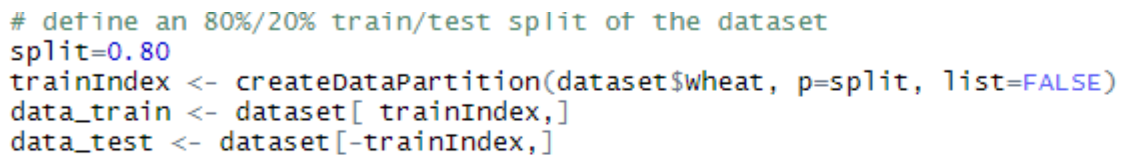
In this dataset, 7 attributes are available, although in the plot showing the accuracy of the different attribute subset sizes. But we can notice that 4 attributes gives almost comparable results. It shows that the V7, V2, V1 and V4 attributes are the highest 4 most important attributes in the dataset and the V3 attribute is the least important.

Task 2: Repeated K-fold cross-validation

Splitting Training & testing data

As next step, the data was split into training and testing data. This enabled one portion of the data to used for training and other part of the unseen data can been used to calculate model efficiency.

Randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model). Make sure to set seed for reproducibility.

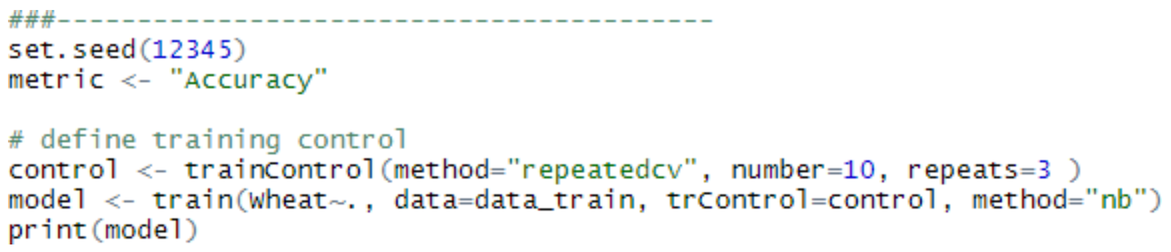


```r
# define an 80%/20% train/test split of the dataset
split=0.80
trainIndex <- createDataPartition(dataset$Wheat, p=split, list=FALSE)
data_train <- dataset[ trainIndex,]
data_test <- dataset[-trainIndex,]
```

Repeated k-fold Cross Validation

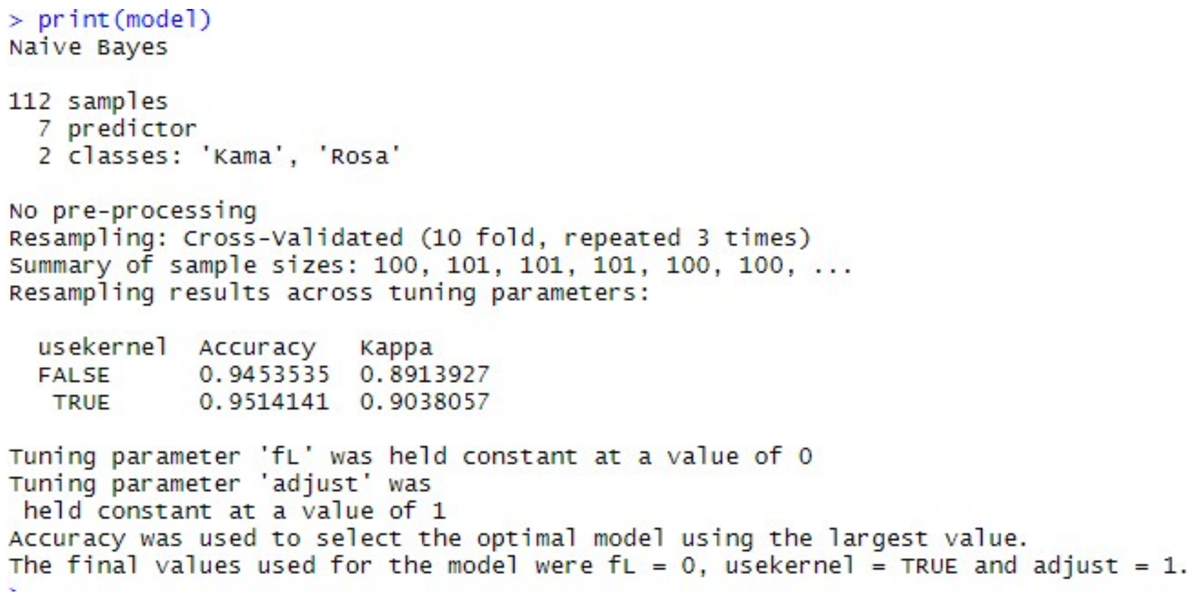
The k-fold cross-validation method evaluates the model performance on different subset of the training data and then calculate the average prediction error rate. The process of splitting the data into k-folds is repeated a number of times, called repeated k-fold cross validation.

Repeated K-fold cross-validation randomly split the data set into k-subsets reserve one subset and train the model on all other subsets and test the model on the reserved. This process id repeated until each of the subsets has served as the test set. In practice, Lower value of K is more biased and hence undesirable. On the other hand, higher value of K is less biased, but can suffer from large variability. Based on the finding 10 subsets have been used in this assignment.



```r
###----------------------------------------
set.seed(12345)
metric <- "Accuracy"

# define training control
control <- trainControl(method="repeatedcv", number=10, repeats=3 )
model <- train(Wheat~., data=data_train, trControl=control, method="nb")
print(model)
```

Result



```
> print(model)
Naive Bayes

112 samples
  7 predictor
  2 classes: 'Kama', 'Rosa'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 100, 101, 101, 101, 100, 100, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.9453535  0.8913927
   TRUE      0.9514141  0.9038057

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust' was
 held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.
```

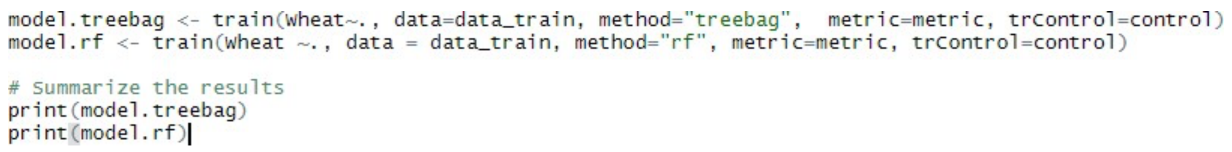
Output shows the model is well optimized where the accuracy is 0.95.

Task 3: Bagging type classifier

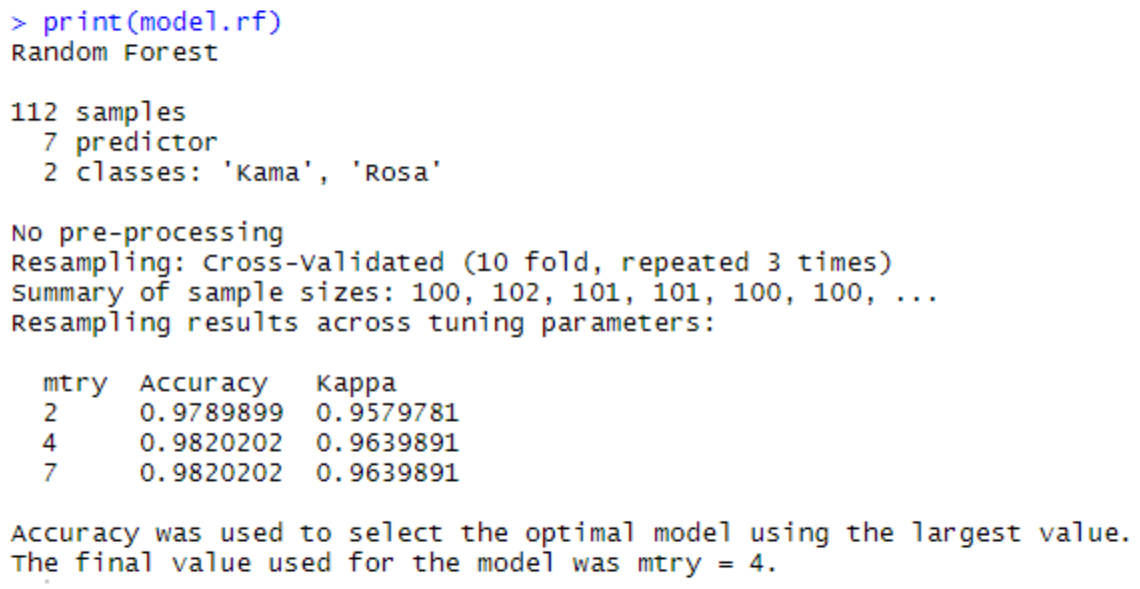
Bagging (Known as bootstrap aggregating) works by constructing various decision tree models from a training data set by repetitively, using multiple bootstrapped subsets averaging the models. It also reduces variance and helps to avoid overfitting.

Random Forest - most commonly used and the most powerful machine learning techniques. It is a special type of bagging applied to decision trees. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, Random decision forests correct for decision trees' habit of overfitting to their training set.

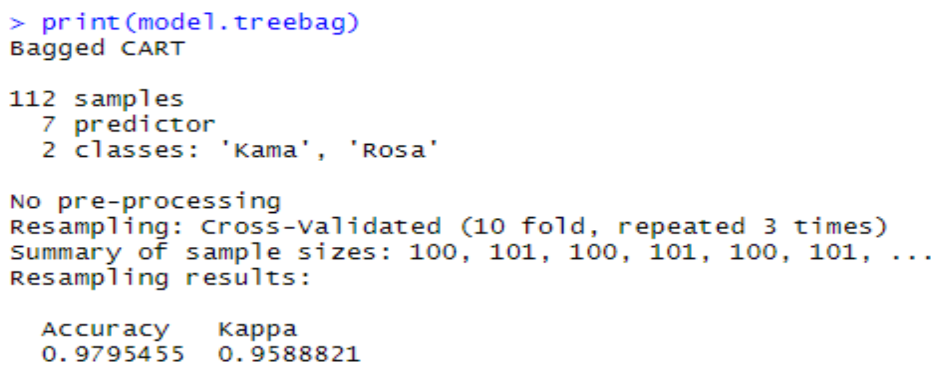
Compared to the standard CART model the random forest provides a strong improvement. Caret train function requires training control parameter.



```
model.treebag <- train(wheat~., data=data_train, method="treebag",  metric=metric, trControl=control)
model.rf <- train(wheat ~., data = data_train, method="rf", metric=metric, trControl=control)

# Summarize the results
print(model.treebag)
print(model.rf)
```

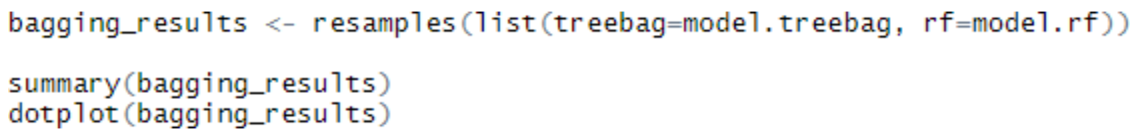


```
> print(model.rf)
Random Forest

112 samples
  7 predictor
  2 classes: 'Kama', 'Rosa'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 100, 102, 101, 101, 100, 100, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  2     0.9789899  0.9579781
  4     0.9820202  0.9639891
  7     0.9820202  0.9639891

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 4.
```

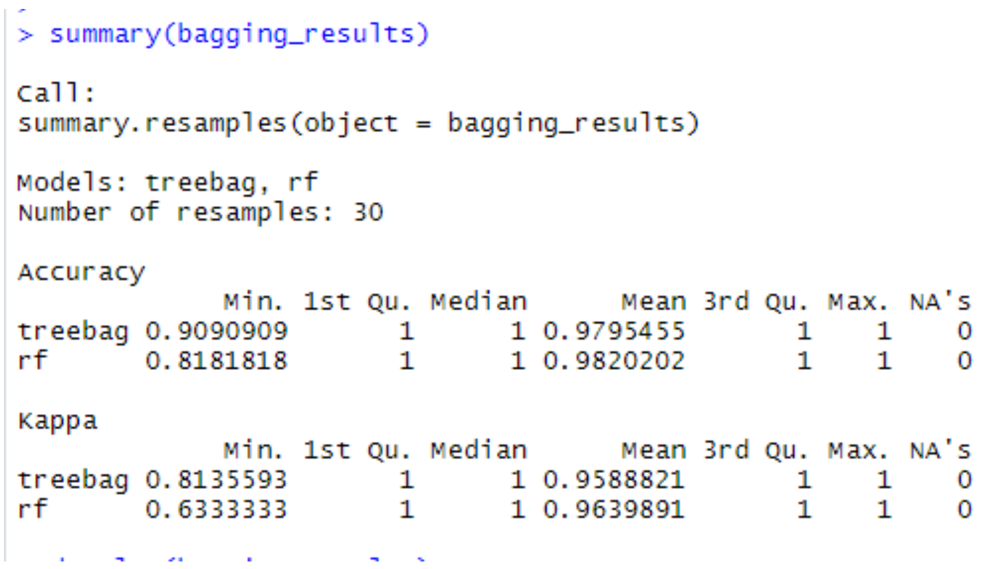


```
> print(model.treebag)
Bagged CART

112 samples
  7 predictor
  2 classes: 'Kama', 'Rosa'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 100, 101, 100, 101, 100, 101, ...
Resampling results:

  Accuracy   Kappa
  0.9795455  0.9588821
```

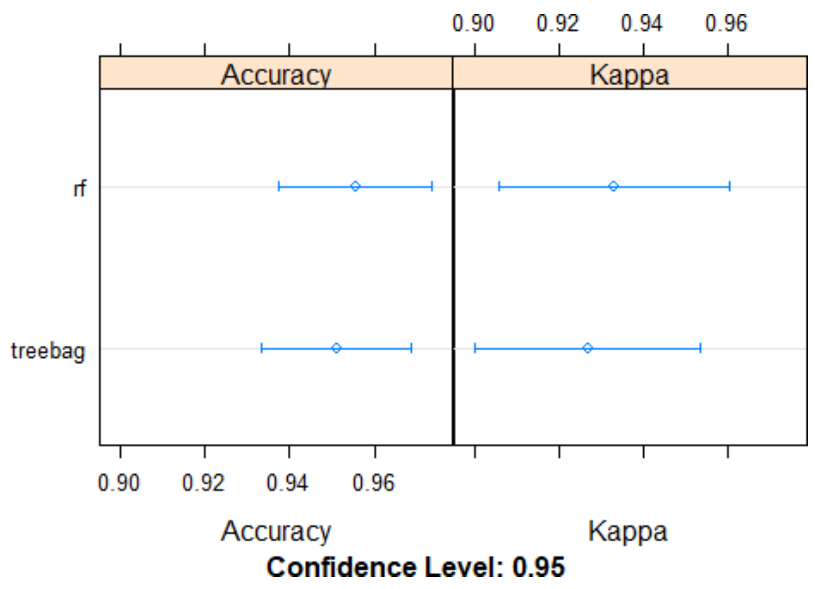
Combining the random forest & tree bag



```
bagging_results <- resamples(list(treebag=model.treebag, rf=model.rf))

summary(bagging_results)
dotplot(bagging_results)
```



```
> summary(bagging_results)

Call:
summary.resamples(object = bagging_results)

Models: treebag, rf
Number of resamples: 30

Accuracy
             Min. 1st Qu. Median      Mean 3rd Qu. Max. NA's
treebag 0.9090909       1      1 0.9795455       1    1    0
rf      0.8181818       1      1 0.9820202       1    1    0

Kappa
             Min. 1st Qu. Median      Mean 3rd Qu. Max. NA's
treebag 0.8135593       1      1 0.9588821       1    1    0
rf      0.6333333       1      1 0.9639891       1    1    0
```



We can see that random forest produces a more accurate model with accuracy of 97%, while . Based on the above created classifiers, the final model accuracy is taken as the mean from the number of repeats.

As next step, the test data will be validated.

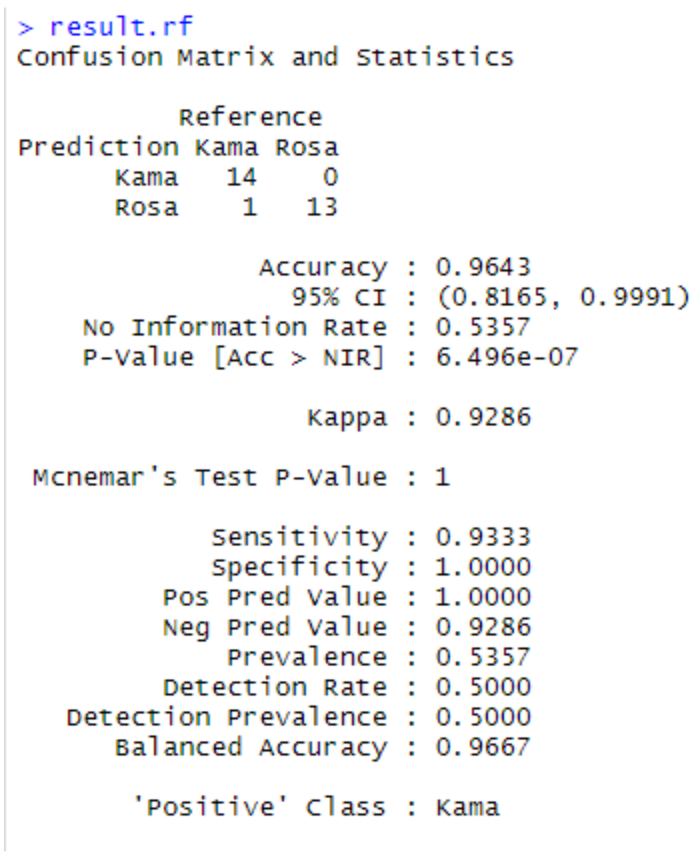
```
#prediction
pred.treebag <- predict(model.treebag, data_test , type='raw')
pred.rf <- predict(model.rf, data_test)

pred.treebag
pred.rf
```

```
> pred.treebag <- predict(model.treebag, data_test , type='raw')
> pred.rf <- predict(model.rf, data_test)
>
> pred.treebag
 [1] Kama Rosa Rosa Kama Rosa Rosa Kama Kama Rosa Rosa Kama Kama Kama Kama Kama Rosa Kama Rosa Kama Kama Rosa
[22] Rosa Kama Kama Rosa Rosa Kama Rosa
Levels: Kama Rosa
> pred.rf
 [1] Kama Rosa Rosa Kama Rosa Rosa Kama Kama Rosa Rosa Kama Kama Kama Kama Kama Rosa Kama Rosa Kama Kama Rosa
[22] Rosa Kama Kama Rosa Rosa Kama Rosa
Levels: Kama Rosa
> |
```

Confusion matrix

```
#confusion matrix
result.treebag <- confusionMatrix(data_test$Wheat,pred.treebag)
result.rf <- confusionMatrix(data_test$Wheat,pred.rf)

result.treebag
result.rf
```

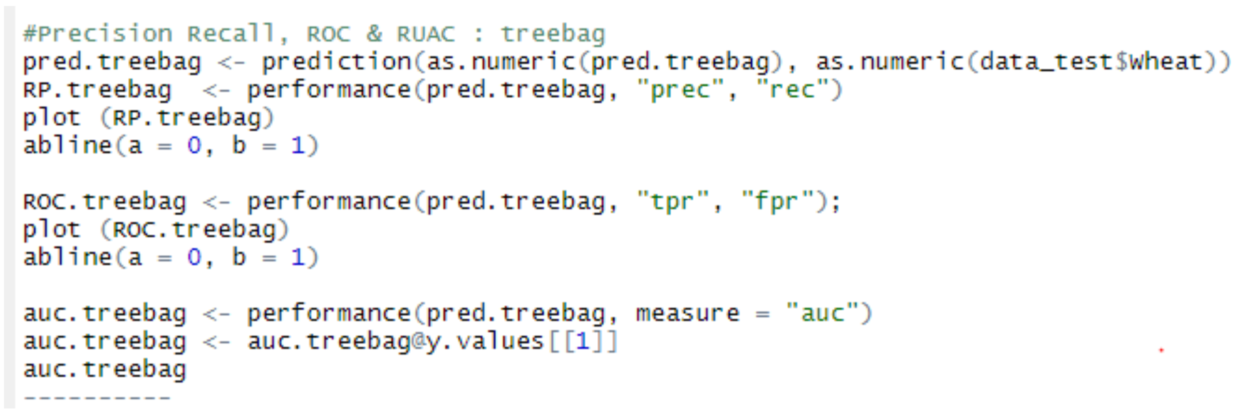
```
> result.treebag
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    1   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5357
    P-Value [Acc > NIR] : 6.496e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9333
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9286
             Prevalence : 0.5357
         Detection Rate : 0.5000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9667

       'Positive' Class : Kama
```



```
> result.rf
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    1   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5357
    P-Value [Acc > NIR] : 6.496e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9333
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9286
             Prevalence : 0.5357
         Detection Rate : 0.5000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9667

       'Positive' Class : Kama

```

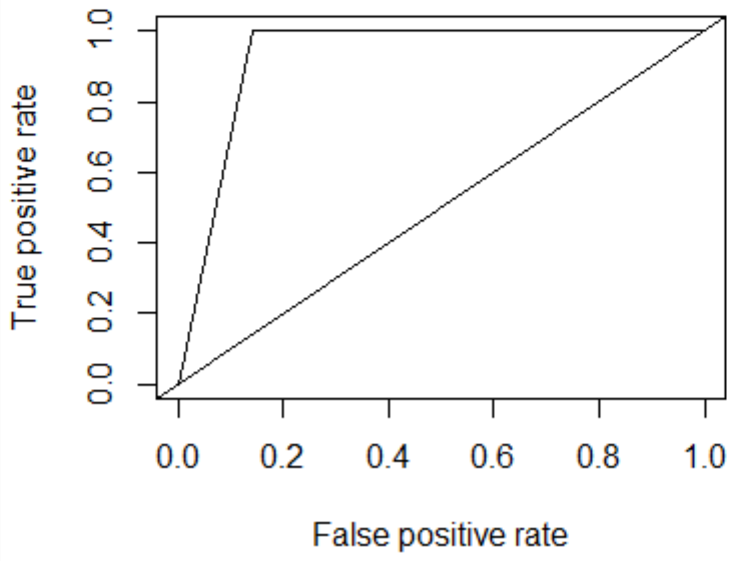
Precision & Recall, ROC, AUC

The precision and recall can be measured in two ways. One using the $byClass and also can be obtained using ROCR class.



```
#Precision Recall, ROC & RUAC : treebag
pred.treebag <- prediction(as.numeric(pred.treebag), as.numeric(data_test$Wheat))
RP.treebag  <- performance(pred.treebag, "prec", "rec")
plot (RP.treebag)
abline(a = 0, b = 1)

ROC.treebag <- performance(pred.treebag, "tpr", "fpr");
plot (ROC.treebag)
abline(a = 0, b = 1)

auc.treebag <- performance(pred.treebag, measure = "auc")
auc.treebag <- auc.treebag@y.values[[1]]
auc.treebag
----------
```

```
pred.rf <- prediction(as.numeric(pred.rf), as.numeric(data_test$wheat))
RP.rf <- performance(pred.rf, "prec", "rec")
plot (RP.rf)
abline(a = 0, b = 1)

ROC.rf <- performance(pred.rf, "tpr", "fpr");
plot (ROC.rf)
abline(a = 0, b = 1)

auc.rf <- performance(pred.rf, measure = "auc")
auc.rf<- auc.rf@y.values[[1]]
auc.rf
```

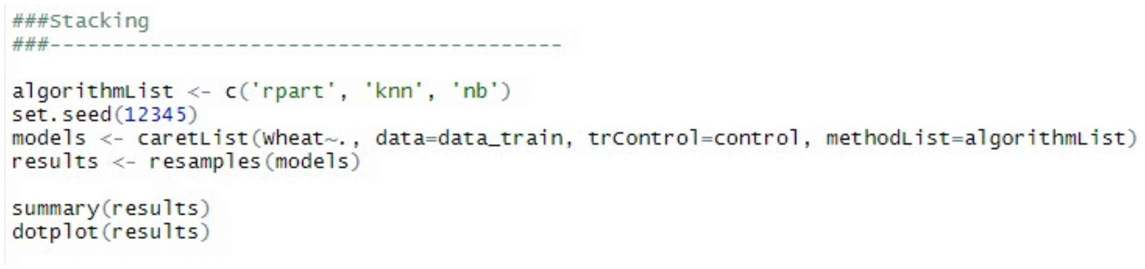


```
> #Precision Recall, ROC & RUAC : treebag
> pred.treebag <- prediction(as.numeric(pred.treebag), as.numeric(data_test$Whea
> RP.treebag  <- performance(pred.treebag, "prec", "rec")
> plot (RP.treebag)
> abline(a = 0, b = 1)
>
> ROC.treebag <- performance(pred.treebag, "tpr", "fpr");
> plot (ROC.treebag)
> abline(a = 0, b = 1)
>
> auc.treebag <- performance(pred.treebag, measure = "auc")
> auc.treebag <- auc.treebag@y.values[[1]]
> auc.treebag
[1] 0.9285714
```

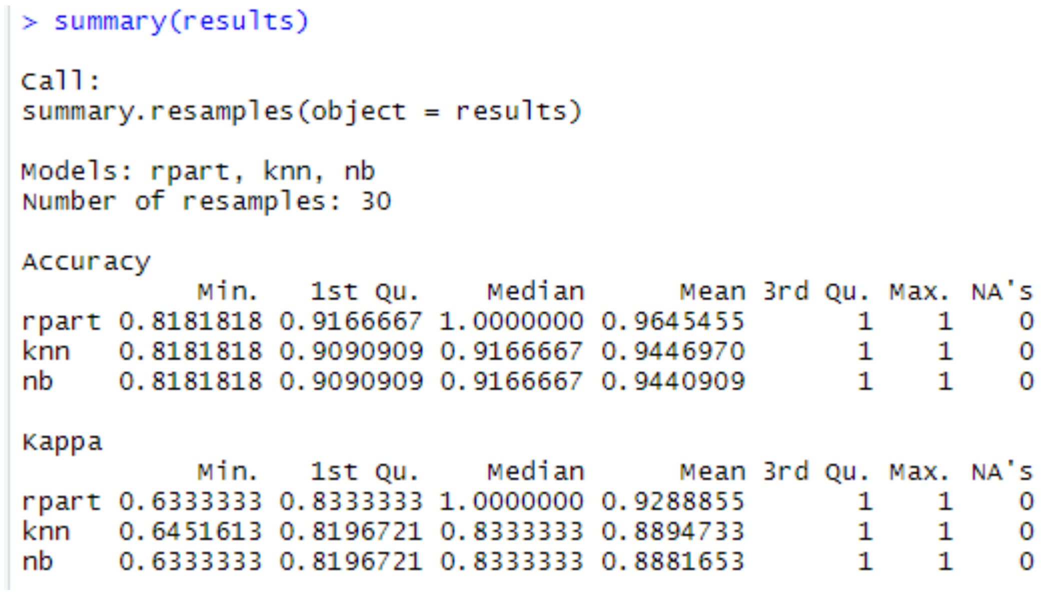
Task 4: Stacking Type Classifier

Stacking is an ensemble machine learning algorithm which combine multiple classification models via a meta-classifier. Compared to single model, stacking is that it can harness the capabilities of a range of well-performing models on a classification or regression task and make predictions that have better performance.

Below is an example that creates these 3 sub-models.



```
###Stacking
###-------------------------------------------

algorithmList <- c('rpart', 'knn', 'nb')
set.seed(12345)
models <- caretList(Wheat~., data=data_train, trControl=control, methodList=algorithmList)
results <- resamples(models)

summary(results)
dotplot(results)
```

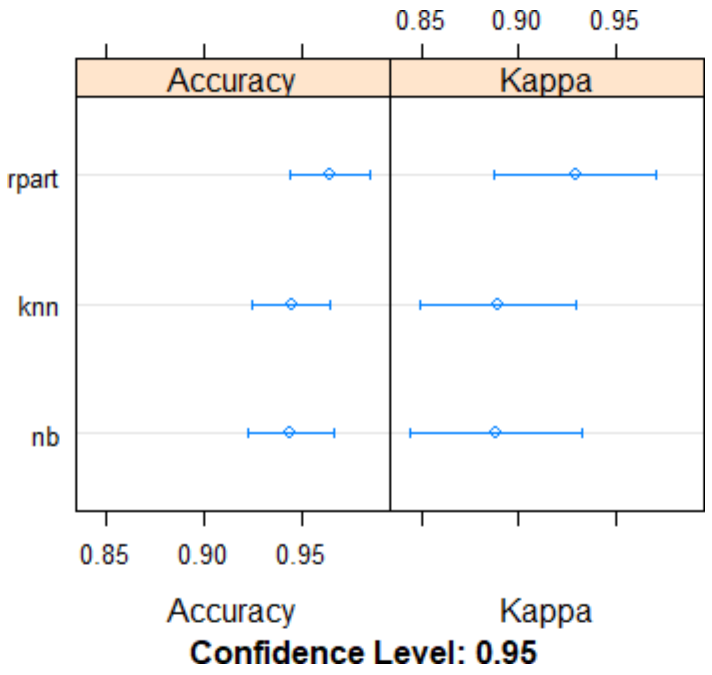


```
> summary(results)

Call:
summary.resamples(object = results)

Models: rpart, knn, nb
Number of resamples: 30

Accuracy
           Min.   1st Qu.    Median      Mean 3rd Qu. Max. NA's
rpart 0.8181818 0.9166667 1.0000000 0.9645455       1    1    0
knn   0.8181818 0.9090909 0.9166667 0.9446970       1    1    0
nb    0.8181818 0.9090909 0.9166667 0.9440909       1    1    0

Kappa
           Min.   1st Qu.    Median      Mean 3rd Qu. Max. NA's
rpart 0.6333333 0.8333333 1.0000000 0.9288855       1    1    0
knn   0.6451613 0.8196721 0.8333333 0.8894733       1    1    0
nb    0.6333333 0.8196721 0.8333333 0.8881653       1    1    0
```

After constructing model, to determine the accuracy of the model, predicting the outcome for new unseen observations not used to build the model.

It involves

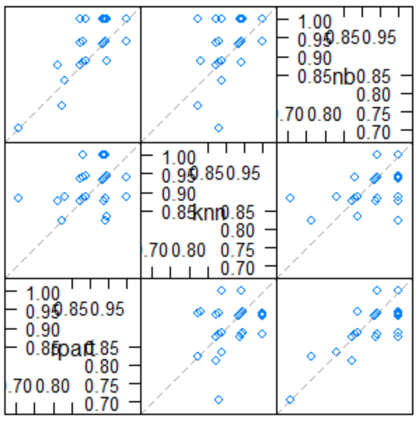
- Build the model on a training data set
- Apply the model on a new test data set to make predictions
- Compute the prediction errors



```
# correlation between results
modelCor(results)
splom(results)
```

```
> dotplot(results)
> # correlation between results
> modelCor(results)
            rpart         knn        nb
rpart 1.000000000 0.008099119 0.6961790
knn   0.008099119 1.000000000 0.4777048
nb    0.696179026 0.477704751 1.0000000
```

Accuracy

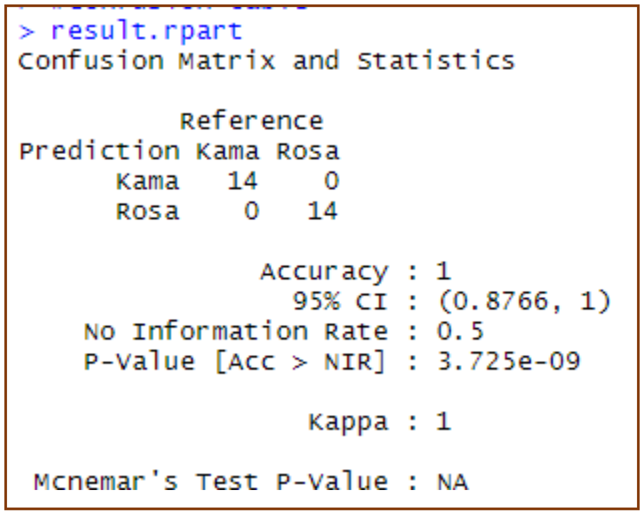


Scatter Plot Matrix

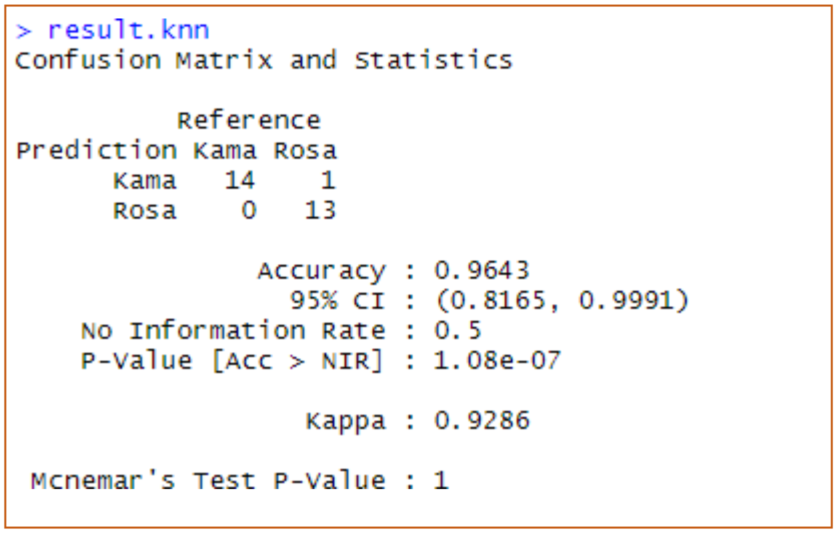
If the predictions for the sub-models were highly corrected (>0.75) then they would be making the same or very similar predictions most of the time reducing the benefit of combining the predictions. We can see that all pairs of predictions have generally low correlation. But one have high correlation. The two methods with the highest correlation between their predictions are rPart and NB at 0.69 correlation which is a high correlation. As next high value. correlation between is 0.47 found between NB & KNN. But this is not more than 0.75, still this is considered as low correlation.

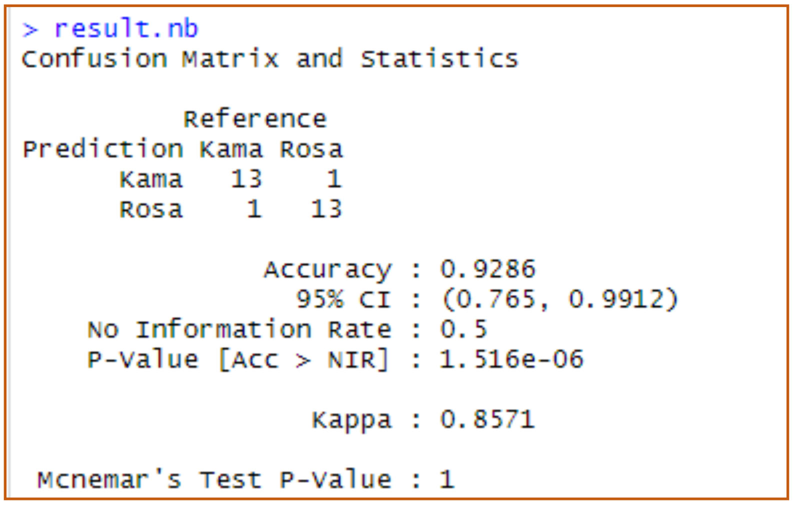
Next step will be testing / prediction.

```
#prediction: rpart
stackpredicted.rpart <- predict(models$rpart, data_test)
result.rpart <- confusionMatrix(stackpredicted.rpart, data_test$Wheat)

#prediction: knn
stackpredicted.knn <- predict(models$knn, data_test)
result.knn <- confusionMatrix(stackpredicted.knn, data_test$Wheat)

#prediction: nb
stackpredicted.nb <- predict(models$nb, data_test)
result.nb <- confusionMatrix(stackpredicted.nb, data_test$Wheat)

#confusion table
result.rpart
result.knn
result.nb
```



```
> result.rpart
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    0   14

               Accuracy : 1
                 95% CI : (0.8766, 1)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 3.725e-09

                  Kappa : 1

 Mcnemar's Test P-Value : NA
```


```
> result.knn
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    1
      Rosa    0   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1.08e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1
```


```
> result.nb
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   13    1
      Rosa    1   13

               Accuracy : 0.9286
                 95% CI : (0.765, 0.9912)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1.516e-06

                  Kappa : 0.8571

 Mcnemar's Test P-Value : 1
```

The separate knn model on the testing data has an accuracy 96%, and the separate rpart model on the testing data has an accuracy of 1.0 . Both of these accuracies are very high, and these two models, knn and rpart, should be used rather than the stacked model. However, rpart on the testing data have lower accuracy than the ones on the training data.

Stacking models

Stacking enables combining the prediction of different models and predictions by the sub-models have low correlation. This means that models are capable to work with various type of data. Following code combine the predictions of the classifiers using knn model.

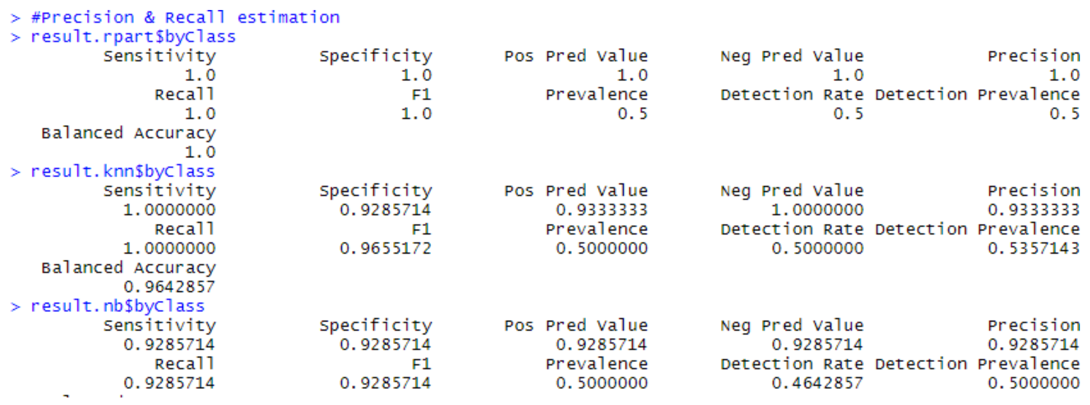
```
# stack using knn
stack <- caretStack(models, method="knn", metric="Accuracy", trControl=control)
stack
```

```
> stack
A knn ensemble of 3 base models: rpart, knn, nb

Ensemble results:
k-Nearest Neighbors

336 samples
  3 predictor
  2 classes: 'Kama', 'Rosa'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 304, 302, 303, 302, 302, 302, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.9614063  0.9227940
  7  0.9753417  0.9506851
  9  0.9723708  0.9447303

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.
> |
```

Precision & Recall

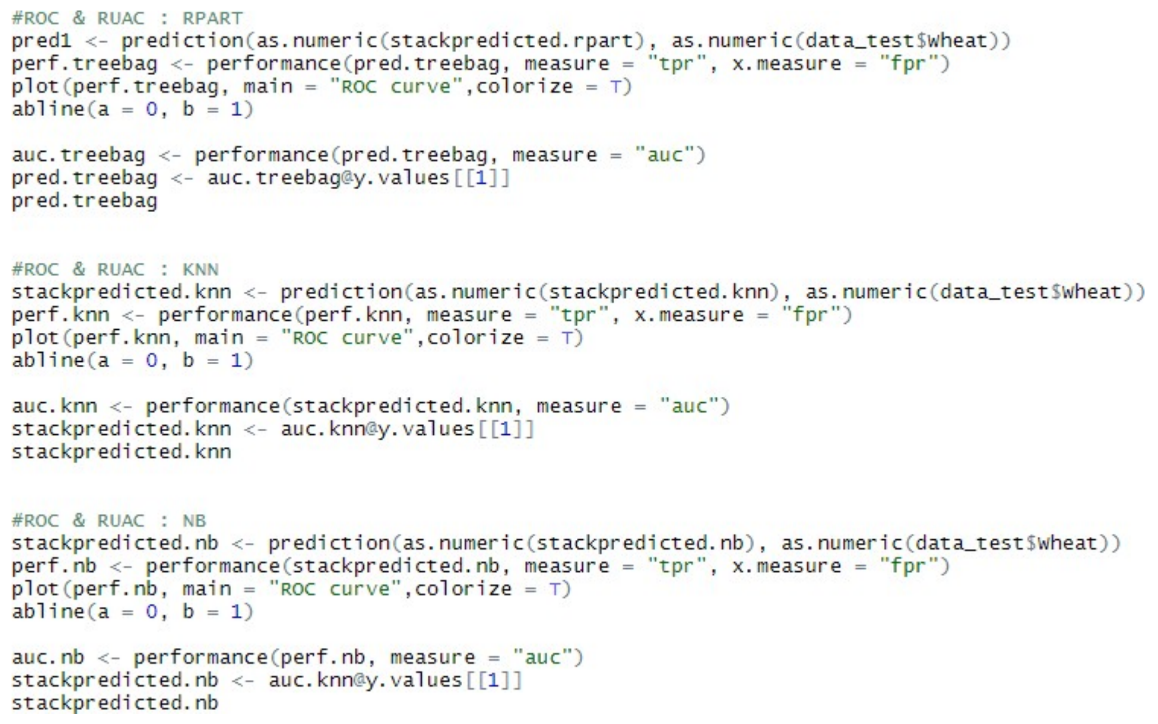
The precision and recall can be measured in two ways. One using the $byClass and also can be obtained using ROCR class.

```
#confusion table
result.rpart
result.knn
result.nb

#Precision & Recall estimation
result.rpart$byClass
result.knn$byClass
result.nb$byClass
```



```
> #Precision & Recall estimation
> result.rpart$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
                 1.0                  1.0                  1.0                  1.0                  1.0 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
                 1.0                  1.0                  0.5                  0.5                  0.5 
   Balanced Accuracy 
                 1.0 
> result.knn$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
           1.0000000            0.9285714            0.9333333            1.0000000            0.9333333 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
           1.0000000            0.9655172            0.5000000            0.5000000            0.5357143 
   Balanced Accuracy 
           0.9642857 
> result.nb$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
           0.9285714            0.9285714            0.9285714            0.9285714            0.9285714 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
           0.9285714            0.9285714            0.5000000            0.4642857            0.5000000 
```

ROC & AUC



```
#ROC & RUAC : RPART
pred1 <- prediction(as.numeric(stackpredicted.rpart), as.numeric(data_test$Wheat))
perf.treebag <- performance(pred.treebag, measure = "tpr", x.measure = "fpr")
plot(perf.treebag, main = "ROC curve",colorize = T)
abline(a = 0, b = 1)

auc.treebag <- performance(pred.treebag, measure = "auc")
pred.treebag <- auc.treebag@y.values[[1]]
pred.treebag


#ROC & RUAC : KNN
stackpredicted.knn <- prediction(as.numeric(stackpredicted.knn), as.numeric(data_test$Wheat))
perf.knn <- performance(perf.knn, measure = "tpr", x.measure = "fpr")
plot(perf.knn, main = "ROC curve",colorize = T)
abline(a = 0, b = 1)

auc.knn <- performance(stackpredicted.knn, measure = "auc")
stackpredicted.knn <- auc.knn@y.values[[1]]
stackpredicted.knn


#ROC & RUAC : NB
stackpredicted.nb <- prediction(as.numeric(stackpredicted.nb), as.numeric(data_test$Wheat))
perf.nb <- performance(stackpredicted.nb, measure = "tpr", x.measure = "fpr")
plot(perf.nb, main = "ROC curve",colorize = T)
abline(a = 0, b = 1)

auc.nb <- performance(perf.nb, measure = "auc")
stackpredicted.nb <- auc.knn@y.values[[1]]
stackpredicted.nb
```

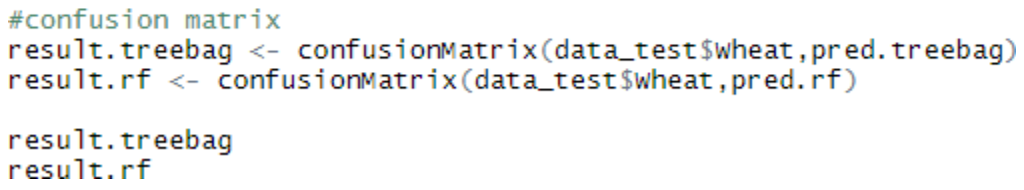
.

Task 5: MeasurePerformance

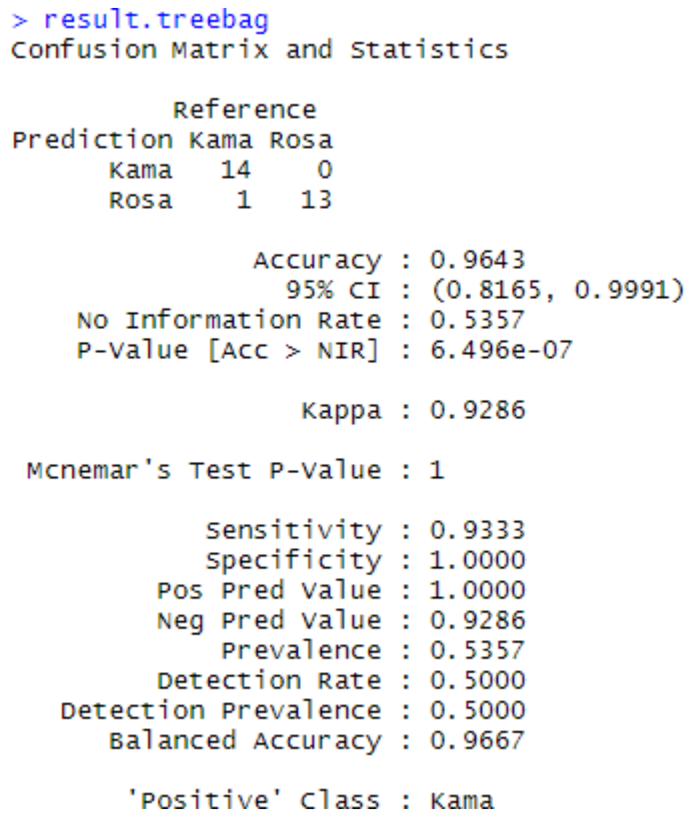
5.1 Confusion matrix

Confusion matrix is used for performance measurement for machine learning classification. It shows predicted classes with associated statistics. The numbers in the confusion matrix are the percent of samples in each category.

5.1.1 Confusion matrix for bagging classifier



```
#confusion matrix
result.treebag <- confusionMatrix(data_test$Wheat,pred.treebag)
result.rf <- confusionMatrix(data_test$Wheat,pred.rf)

result.treebag
result.rf
```



```
> result.treebag
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    1   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5357
    P-Value [Acc > NIR] : 6.496e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9333
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9286
             Prevalence : 0.5357
         Detection Rate : 0.5000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9667

       'Positive' Class : Kama
```

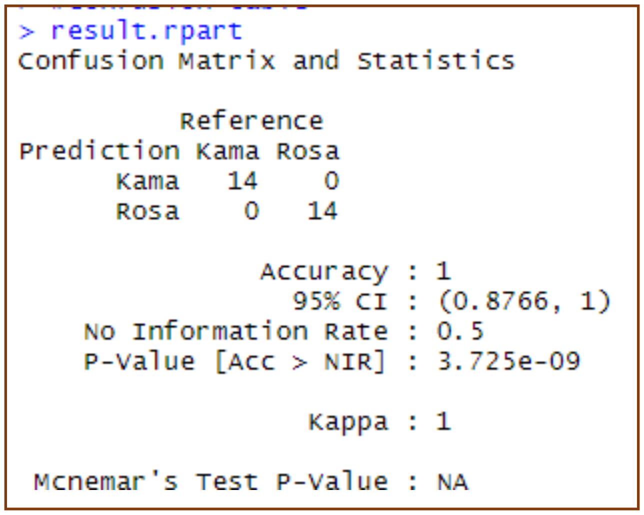
```
> result.rf
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    1   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5357
    P-Value [Acc > NIR] : 6.496e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9333
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9286
             Prevalence : 0.5357
         Detection Rate : 0.5000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9667

       'Positive' Class : Kama
```

Result

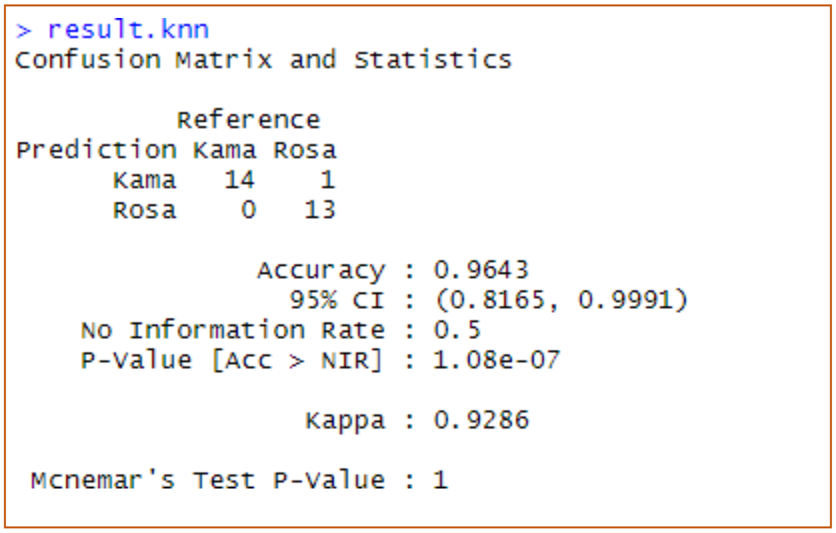
Based on the confusion matrix, both bagged CART & random forest have similar accuracy of 96%. When looking at the confusion matrix, in both model have only one mis-classification where one Rosa instance has been classified as Kama. As conclusion, both model are optimized well.

5.1.2 Confusion matrix for stacking classifier

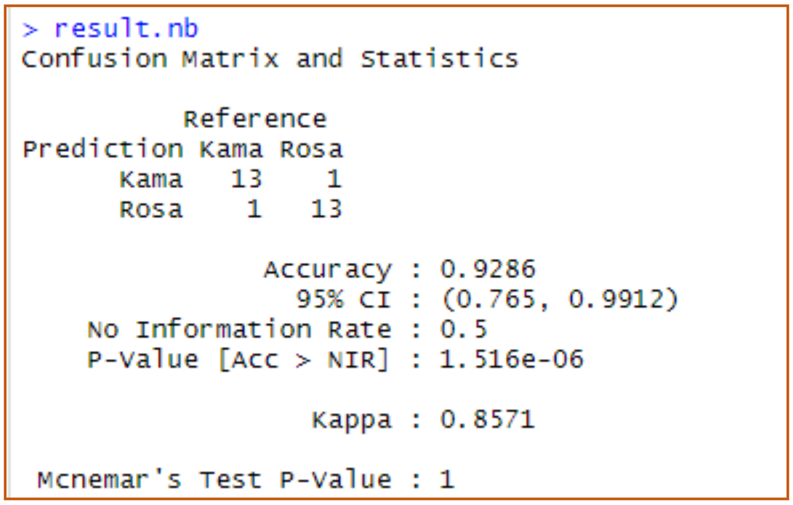
```
#prediction: rpart
stackpredicted.rpart <- predict(models$rpart, data_test)
result.rpart <- confusionMatrix(stackpredicted.rpart, data_test$Wheat)

#prediction: knn
stackpredicted.knn <- predict(models$knn, data_test)
result.knn <- confusionMatrix(stackpredicted.knn, data_test$Wheat)

#prediction: nb
stackpredicted.nb <- predict(models$nb, data_test)
result.nb <- confusionMatrix(stackpredicted.nb, data_test$Wheat)

#confusion table
result.rpart
result.knn
result.nb
```



```
> result.rpart
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    0   14

               Accuracy : 1
                 95% CI : (0.8766, 1)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 3.725e-09

                  Kappa : 1

 Mcnemar's Test P-Value : NA
```



```
> result.knn
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    1
      Rosa    0   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1.08e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1
```



```
> result.nb
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   13    1
      Rosa    1   13

               Accuracy : 0.9286
                 95% CI : (0.765, 0.9912)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1.516e-06

                  Kappa : 0.8571

 Mcnemar's Test P-Value : 1
```

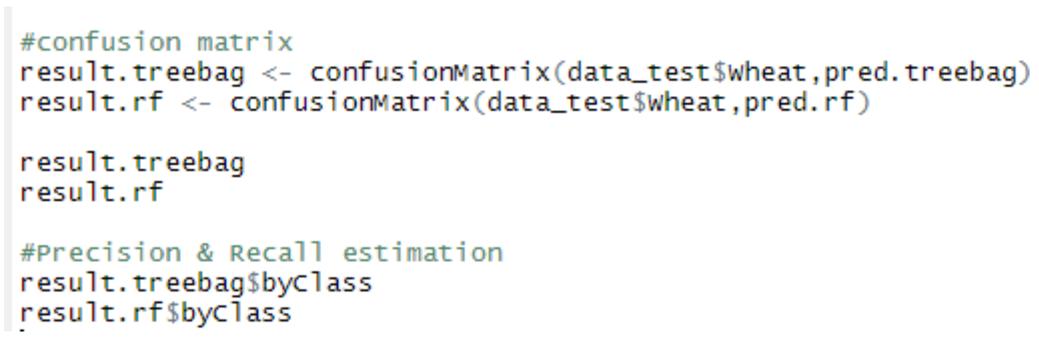
Result

Based on the confusion matrix result, we can see rpart have high accuracy, as second KNN has the accuracy of 96.43% and NB has the accuracy of 92.86%. RPART model has not mis classification. And there have 1 & 2 mis-classification. Therefore, we can see that RPART is the best model in this scenario.

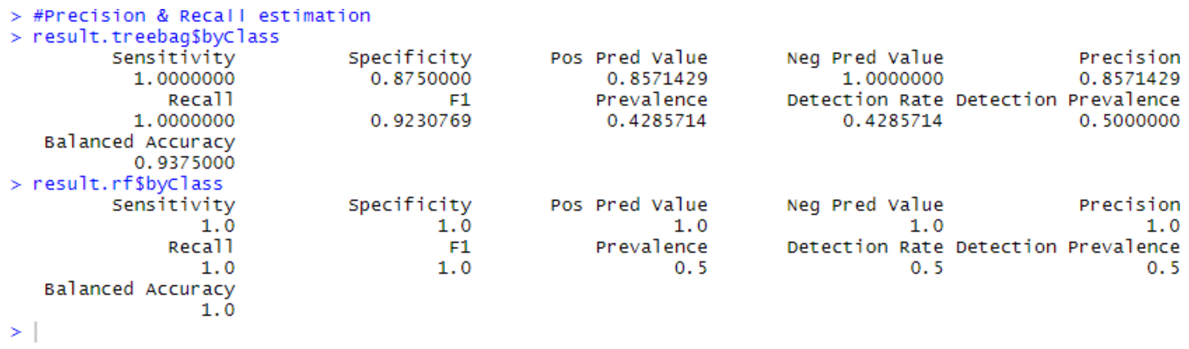
5.2 Precision vs. Recall estimation

Another way to evaluate classifier accuracy, is precision and recall. Recall is the same as sensitivity. The precision and recall can be measured in two ways. One using the $byClass and also can be obtained using ROCR class.

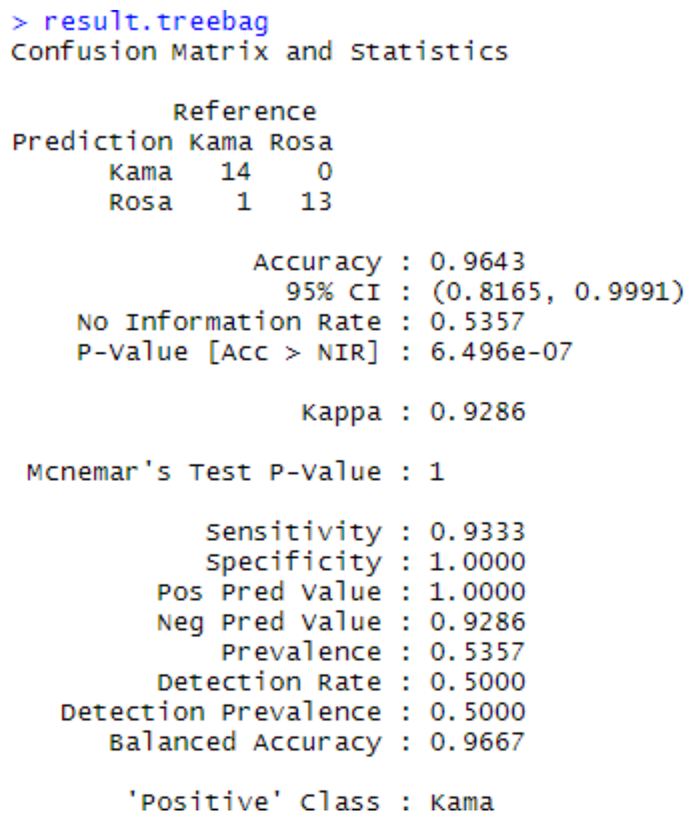
5.2.1 Precision & recall for bagging classifier



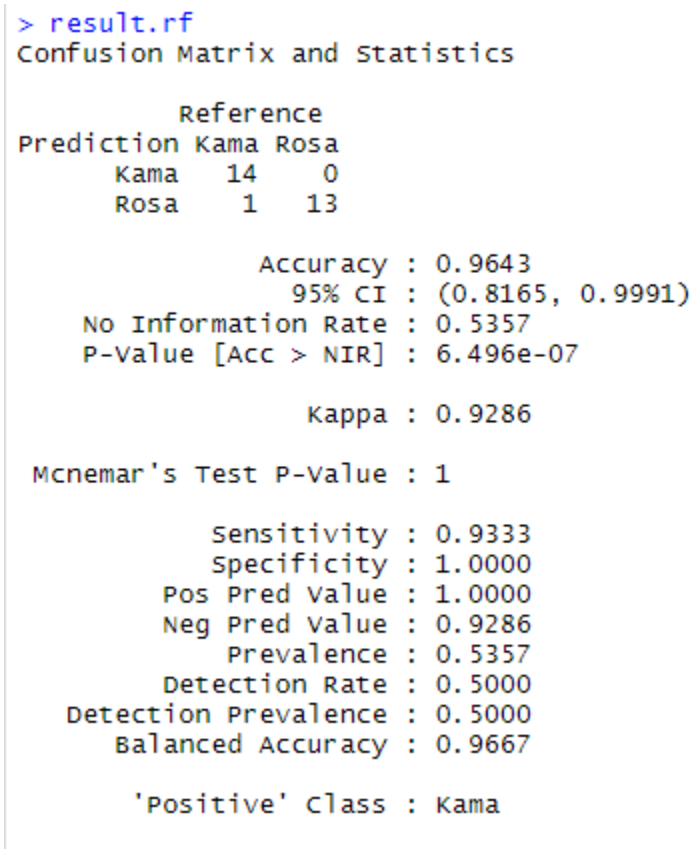
```
#confusion matrix
result.treebag <- confusionMatrix(data_test$Wheat,pred.treebag)
result.rf <- confusionMatrix(data_test$Wheat,pred.rf)

result.treebag
result.rf

#Precision & Recall estimation
result.treebag$byClass
result.rf$byClass
```



```
> #Precision & Recall estimation
> result.treebag$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
           1.0000000            0.8750000            0.8571429            1.0000000            0.8571429 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
           1.0000000            0.9230769            0.4285714            0.4285714            0.5000000 
   Balanced Accuracy 
           0.9375000 
> result.rf$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
                 1.0                  1.0                  1.0                  1.0                  1.0 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
                 1.0                  1.0                  0.5                  0.5                  0.5 
   Balanced Accuracy 
                 1.0 
> |
```



```
> result.treebag
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    1   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5357
    P-Value [Acc > NIR] : 6.496e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9333
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9286
             Prevalence : 0.5357
         Detection Rate : 0.5000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9667

       'Positive' Class : Kama
```

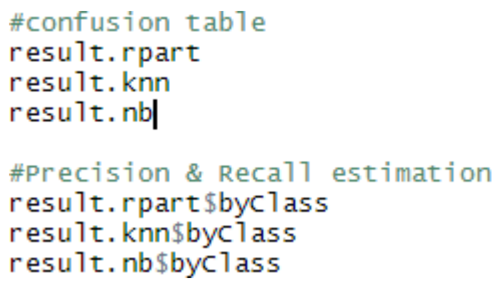


```
> result.rf
Confusion Matrix and Statistics

          Reference
Prediction Kama Rosa
      Kama   14    0
      Rosa    1   13

               Accuracy : 0.9643
                 95% CI : (0.8165, 0.9991)
    No Information Rate : 0.5357
    P-Value [Acc > NIR] : 6.496e-07

                  Kappa : 0.9286

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9333
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9286
             Prevalence : 0.5357
         Detection Rate : 0.5000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9667

       'Positive' Class : Kama
```

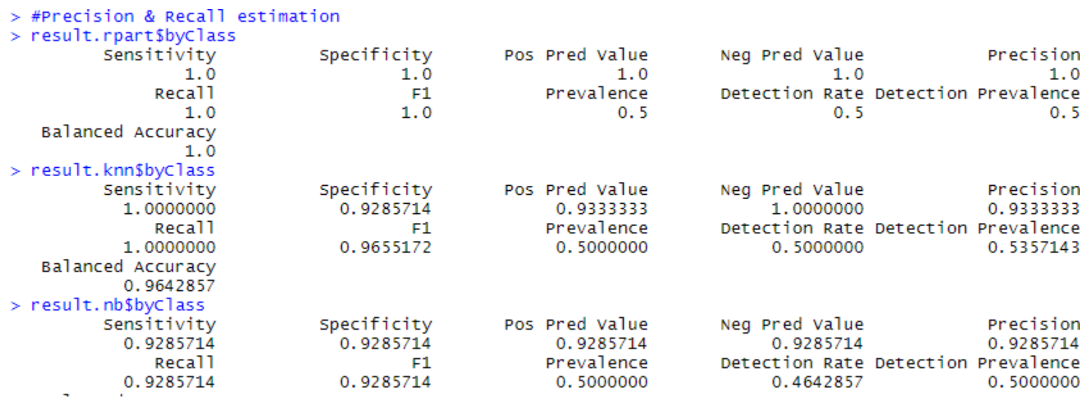
Result

Based on the confusion matrix, we can see that both models have similar accuracy level of 96.4%. So both are well optimized model.

5.2.2 Precision & recall for stacking classifier



```
#confusion table
result.rpart
result.knn
result.nb

#Precision & Recall estimation
result.rpart$byClass
result.knn$byClass
result.nb$byClass
```



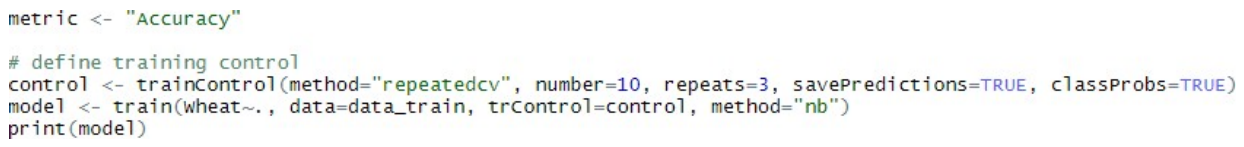
```
> #Precision & Recall estimation
> result.rpart$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
                 1.0                  1.0                  1.0                  1.0                  1.0 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
                 1.0                  1.0                  0.5                  0.5                  0.5 
   Balanced Accuracy 
                 1.0 
> result.knn$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
           1.0000000            0.9285714            0.9333333            1.0000000            0.9333333 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
           1.0000000            0.9655172            0.5000000            0.5000000            0.5357143 
   Balanced Accuracy 
           0.9642857 
> result.nb$byClass
         Sensitivity          Specificity       Pos Pred Value       Neg Pred Value            Precision 
           0.9285714            0.9285714            0.9285714            0.9285714            0.9285714 
              Recall                   F1           Prevalence       Detection Rate Detection Prevalence 
           0.9285714            0.9285714            0.5000000            0.4642857            0.5000000 
```

Result

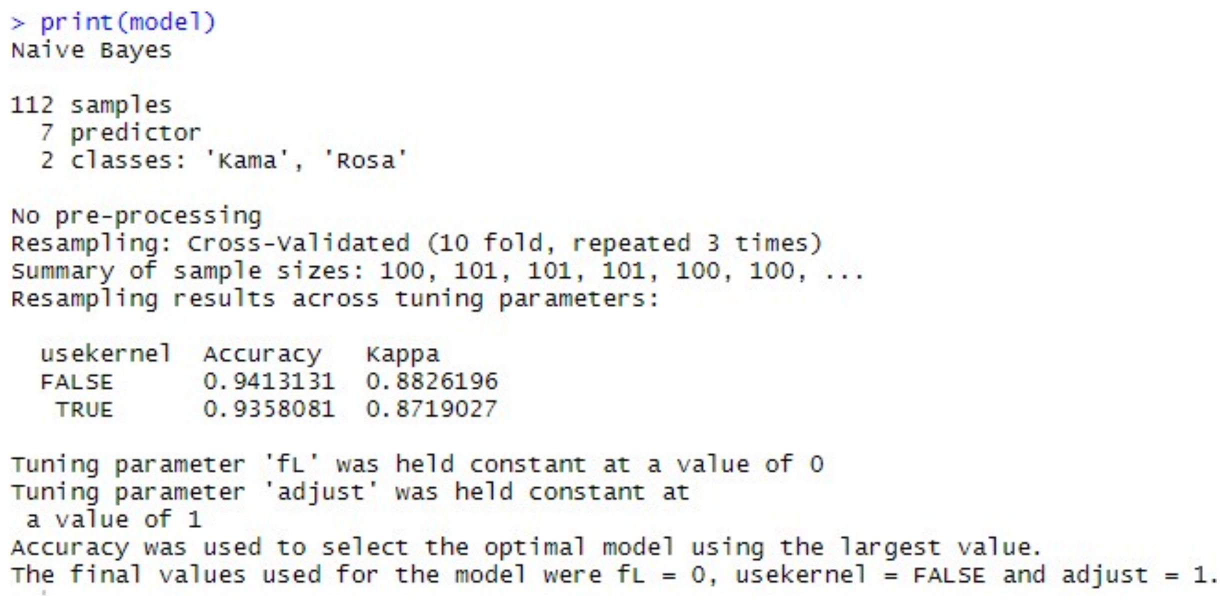
The RPART has precision value of 1, while KNN has the value of 0.9333 which is still considered high. For NB, we had precision of 92%, so if the prediction is positive, then there is an empirically 0.92 chance that the seed belongs to that type of Wheat.

5.3 Accuracy Estimation

In this assignment, Repeated k-fold Cross Validation has been used for accuracy estimation, considering the amount of data and complexity. From the Repeated k-fold, the model accuracy is taken as the mean from the number of repeats. As pre the result, the model is optimized to have accuracy of 94% which is considered high.



```
metric <- "Accuracy"

# define training control
control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE, classProbs=TRUE)
model <- train(Wheat~., data=data_train, trControl=control, method="nb")
print(model)
```



```
> print(model)
Naive Bayes

112 samples
  7 predictor
  2 classes: 'Kama', 'Rosa'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 100, 101, 101, 101, 100, 100, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.9413131  0.8826196
   TRUE      0.9358081  0.8719027

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust' was held constant at
 a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = FALSE and adjust = 1.
```

5.4 ROC(receiver operating characteristic curve) & RAUC (receiver under the curve area)

ROC curves are used in binary classification to output of a classifier. In order to extend ROC curve, the output was converted to binary. ROC curve is a performance measurement for classification at different settings. ROC is a probability curve.

RAUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at finding between Kama and Rosa seeds.

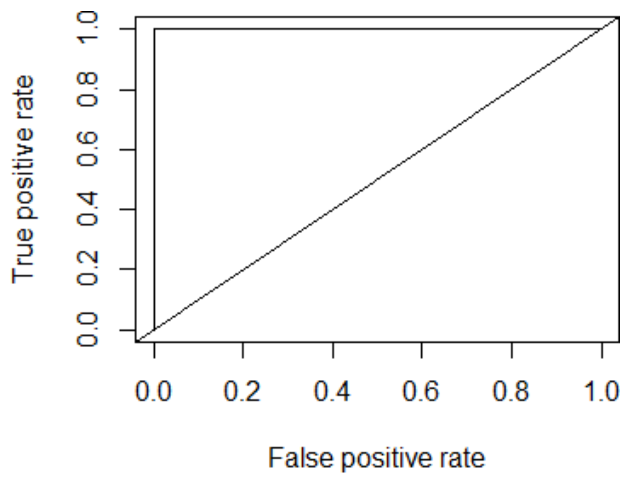
An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability.

5.4.1 ROC & RAUC for bagging classifier

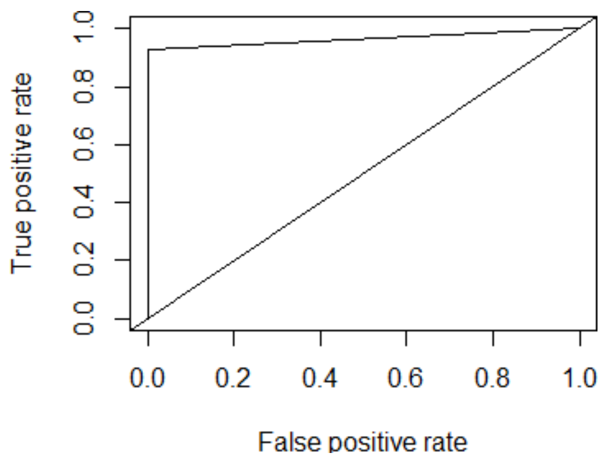
```
#Precision Recall, ROC & RUAC : treebag
pred.treebag <- prediction(as.numeric(pred.treebag), as.numeric(data_test$Wheat))
RP.treebag  <- performance(pred.treebag, "prec", "rec")
plot (RP.treebag)
abline(a = 0, b = 1)

ROC.treebag <- performance(pred.treebag, "tpr", "fpr");
plot (ROC.treebag)
abline(a = 0, b = 1)

auc.treebag <- performance(pred.treebag, measure = "auc")
auc.treebag <- auc.treebag@y.values[[1]]
auc.treebag
----------
```

```
pred.rf <- prediction(as.numeric(pred.rf), as.numeric(data_test$Wheat))
RP.rf <- performance(pred.rf, "prec", "rec")
plot (RP.rf)
abline(a = 0, b = 1)

ROC.rf <- performance(pred.rf, "tpr", "fpr");
plot (ROC.rf)
abline(a = 0, b = 1)

auc.rf <- performance(pred.rf, measure = "auc")
auc.rf<- auc.rf@y.values[[1]]
auc.rf
```

Tree bag



Random Forest

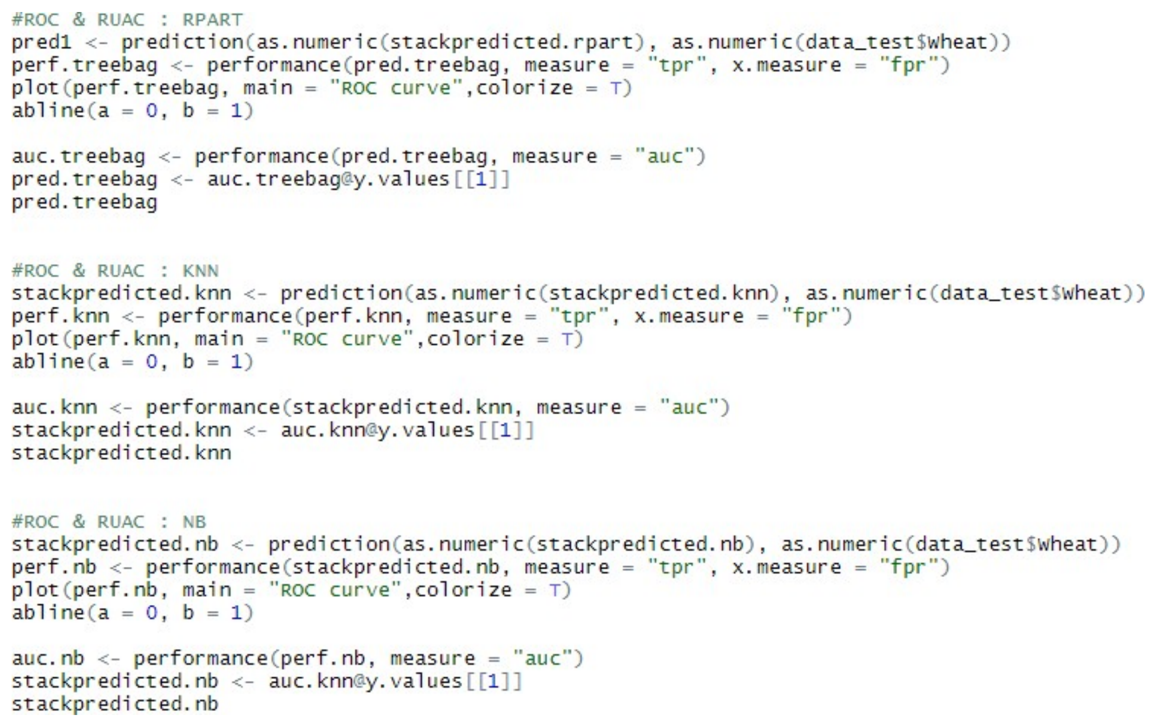


Model	AUC Value
Tree bag	1.0
Random Forest	0.92

Result

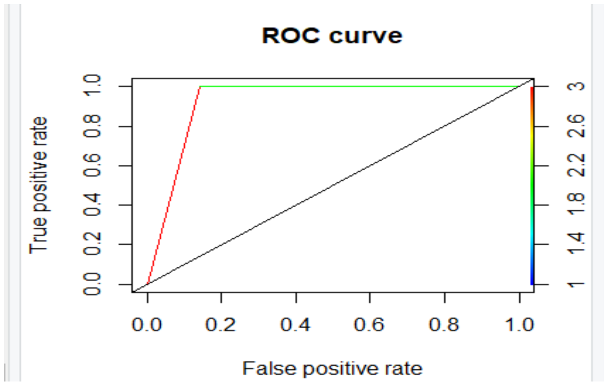
Tree Bag shows high ROC curve & AUC value , while Random forest shows lower ROC & AUC value. Therefore in this case, Tree bag is considered as optimized model.

5.4.2 ROC & RUAC for bagging classifier

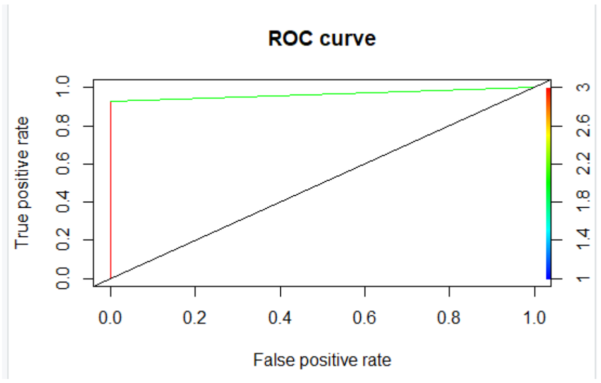


```
#ROC & RUAC : RPART
pred1 <- prediction(as.numeric(stackpredicted.rpart), as.numeric(data_test$wheat))
perf.treebag <- performance(pred.treebag, measure = "tpr", x.measure = "fpr")
plot(perf.treebag, main = "ROC curve",colorize = T)
abline(a = 0, b = 1)

auc.treebag <- performance(pred.treebag, measure = "auc")
pred.treebag <- auc.treebag@y.values[[1]]
pred.treebag


#ROC & RUAC : KNN
stackpredicted.knn <- prediction(as.numeric(stackpredicted.knn), as.numeric(data_test$wheat))
perf.knn <- performance(perf.knn, measure = "tpr", x.measure = "fpr")
plot(perf.knn, main = "ROC curve",colorize = T)
abline(a = 0, b = 1)

auc.knn <- performance(stackpredicted.knn, measure = "auc")
stackpredicted.knn <- auc.knn@y.values[[1]]
stackpredicted.knn


#ROC & RUAC : NB
stackpredicted.nb <- prediction(as.numeric(stackpredicted.nb), as.numeric(data_test$wheat))
perf.nb <- performance(stackpredicted.nb, measure = "tpr", x.measure = "fpr")
plot(perf.nb, main = "ROC curve",colorize = T)
abline(a = 0, b = 1)

auc.nb <- performance(perf.nb, measure = "auc")
stackpredicted.nb <- auc.knn@y.values[[1]]
stackpredicted.nb
```

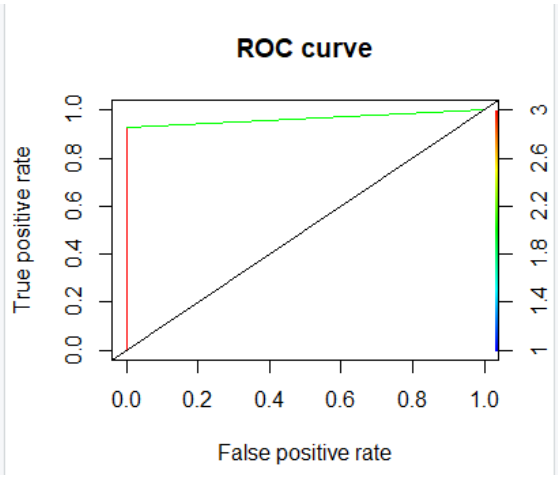
ROC for RPART

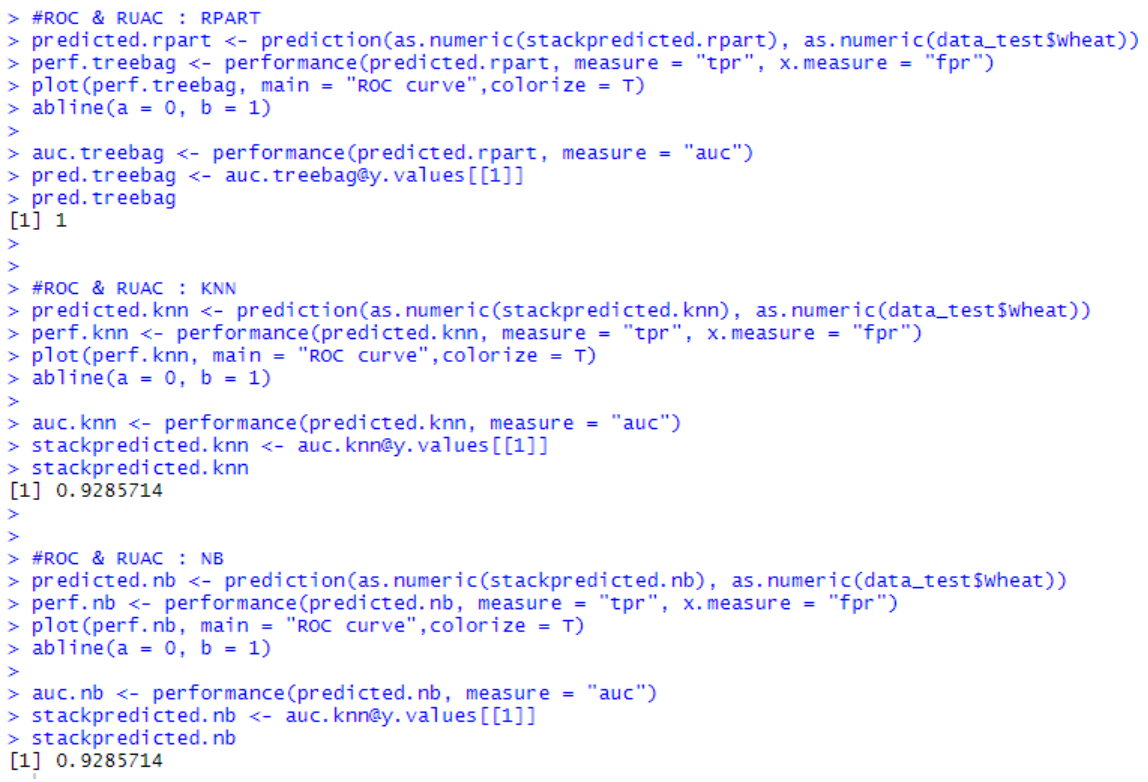


ROC for KNN



ROC for NB





```
> #ROC & RUAC : RPART
> predicted.rpart <- prediction(as.numeric(stackpredicted.rpart), as.numeric(data_test$wheat))
> perf.treebag <- performance(predicted.rpart, measure = "tpr", x.measure = "fpr")
> plot(perf.treebag, main = "ROC curve",colorize = T)
> abline(a = 0, b = 1)
>
> auc.treebag <- performance(predicted.rpart, measure = "auc")
> pred.treebag <- auc.treebag@y.values[[1]]
> pred.treebag
[1] 1
>
>
> #ROC & RUAC : KNN
> predicted.knn <- prediction(as.numeric(stackpredicted.knn), as.numeric(data_test$wheat))
> perf.knn <- performance(predicted.knn, measure = "tpr", x.measure = "fpr")
> plot(perf.knn, main = "ROC curve",colorize = T)
> abline(a = 0, b = 1)
>
> auc.knn <- performance(predicted.knn, measure = "auc")
> stackpredicted.knn <- auc.knn@y.values[[1]]
> stackpredicted.knn
[1] 0.9285714
>
>
> #ROC & RUAC : NB
> predicted.nb <- prediction(as.numeric(stackpredicted.nb), as.numeric(data_test$wheat))
> perf.nb <- performance(predicted.nb, measure = "tpr", x.measure = "fpr")
> plot(perf.nb, main = "ROC curve",colorize = T)
> abline(a = 0, b = 1)
>
> auc.nb <- performance(predicted.nb, measure = "auc")
> stackpredicted.nb <- auc.knn@y.values[[1]]
> stackpredicted.nb
[1] 0.9285714
```

Model	RAUC Value
RPart	1.0
KNN	0.9285714
NB	0.9285714

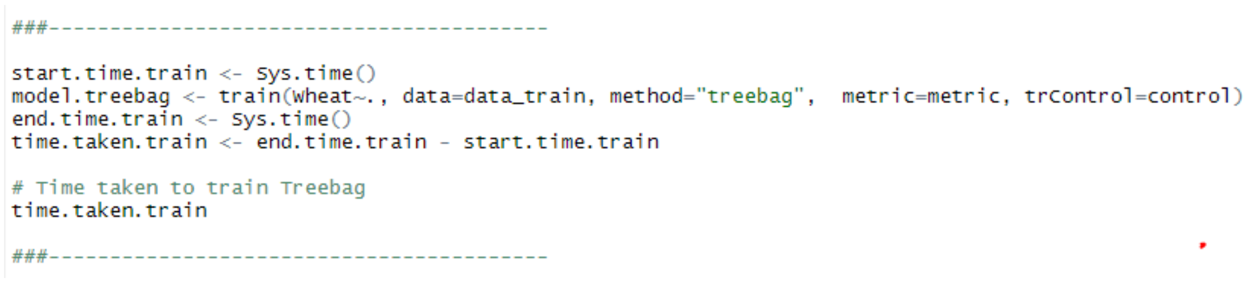
<u>Result</u>

Based on the result, we can see that RAPRT has the high ROC and RAUC while KNN and NB have similar ROC & RAUC values. Therefore, RPART is considered as well optimized model in this case.

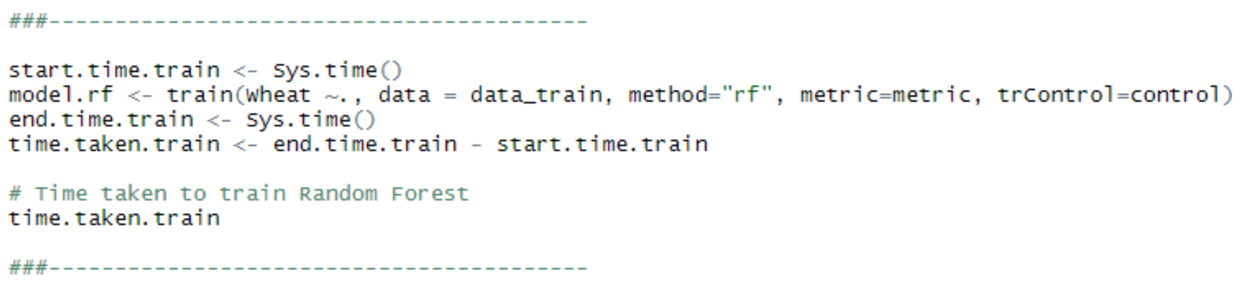
5.5 Training time

Training time indicates the time it takes to train the model. The time is recoded using the inbuilt function of using the timer.

5.5.1 Training time for bagging classifier



```
###----------------------------------------

start.time.train <- Sys.time()
model.treebag <- train(wheat~., data=data_train, method="treebag",  metric=metric, trControl=control)
end.time.train <- Sys.time()
time.taken.train <- end.time.train - start.time.train

# Time taken to train Treebag
time.taken.train

###----------------------------------------
```



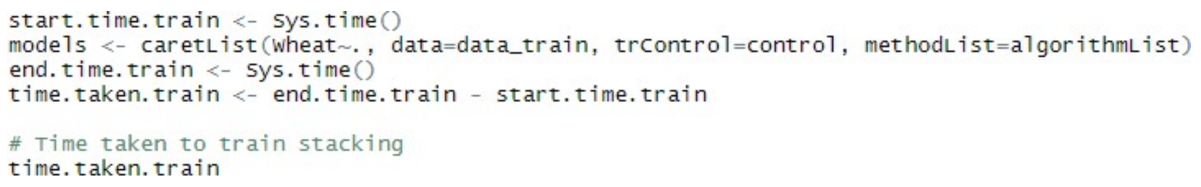
```
###-----------------------------------------

start.time.train <- Sys.time()
model.rf <- train(wheat ~., data = data_train, method="rf", metric=metric, trControl=control)
end.time.train <- Sys.time()
time.taken.train <- end.time.train - start.time.train

# Time taken to train Random Forest
time.taken.train

###-----------------------------------------
```

Model	Time taken (seconds)
Tree bag	3.706525 secs
Random forest	2.71474 secs

<u>Result</u>

In this case, Tree bag takes highest time to train which is 3.7 seconds, while Random forest has the lowest training time of 2.7 seconds. In this case, random forest is the quickest model fortraining.

5.5.2 Training time for stacking classifier



```
start.time.train <- Sys.time()
models <- caretList(wheat~., data=data_train, trControl=control, methodList=algorithmList)
end.time.train <- Sys.time()
time.taken.train <- end.time.train - start.time.train

# Time taken to train stacking
time.taken.train
```

Model	Time taken (seconds)
Stacking classifier	3.900566 secs

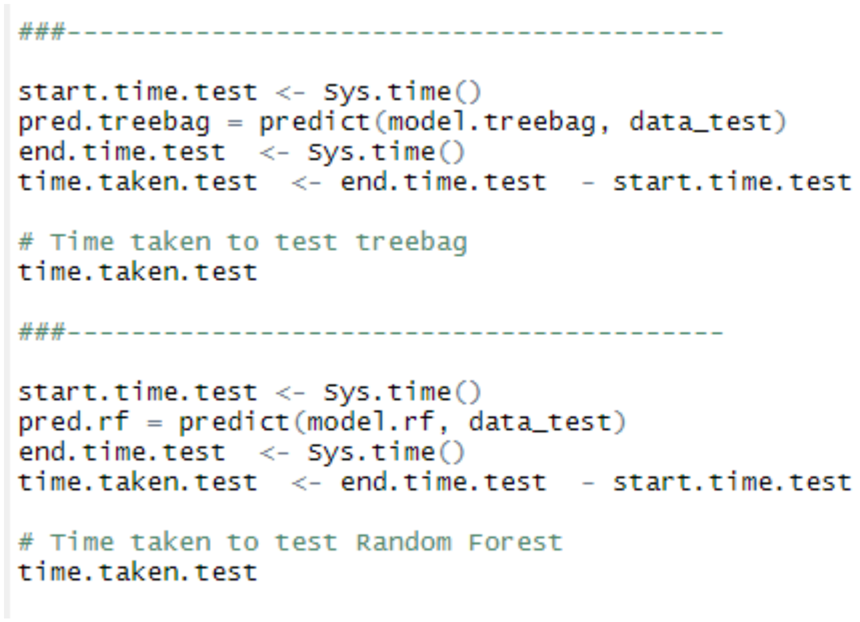
Result

The stacking classifier takes more time compared to bagging classifiers, which is 3.9 seconds.

5.6 Testing time

Testing time indicates the time it takes to train the model. The time is recoded using the inbuilt function of using the timer.

5.6.1 Testing time for bagging classifier



```
###-----------------------------------------

start.time.test <- Sys.time()
pred.treebag = predict(model.treebag, data_test)
end.time.test  <- Sys.time()
time.taken.test  <- end.time.test  - start.time.test

# Time taken to test treebag
time.taken.test

###-----------------------------------------

start.time.test <- Sys.time()
pred.rf = predict(model.rf, data_test)
end.time.test  <- Sys.time()
time.taken.test  <- end.time.test  - start.time.test

# Time taken to test Random Forest
time.taken.test
```

Model	Time taken (seconds)
Tree bag	0.01496005 secs
Random forest	0.004987001 secs

Result

Testing time is considerably low since, the test data consists of only 20% of the entire dataset. The testing time of tree bag is higher, where 0.0149 secs. Random forest test time is considerably lower which is 0.0049 seconds. In this case, Random forest is the quickest model.

5.6.2 Testing time for stacking classifier

```
###-----------------------------------------

start.time.train <- Sys.time()
model.rf <- train(wheat ~., data = data_train, method="rf", metric=metric, trControl=control)
end.time.train <- Sys.time()
time.taken.train <- end.time.train - start.time.train

# Time taken to train Random Forest
time.taken.train

###-----------------------------------------
```

```
###-----------------------------------------

start.time.test <- Sys.time()
pred.treebag = predict(model.treebag, data_test)
end.time.test  <- Sys.time()
time.taken.test  <- end.time.test  - start.time.test

# Time taken to test treebag
time.taken.test

###-----------------------------------------

start.time.test <- Sys.time()
pred.rf = predict(model.rf, data_test)
end.time.test  <- Sys.time()
time.taken.test  <- end.time.test  - start.time.test

# Time taken to test Random Forest
time.taken.test
```

Model	Time taken (seconds)
RPart	0.004986048 secs
KNN	0.003987074 secs
NB	0.02689195 secs

Result

Testing time is considerably low since, the test data consists of only 20% of the entire dataset. As per the result, NB has the highest testing time of 0.0268 seconds. Next slowest model is identified as RPART which take 0.0049 seconds. Finally the fastest model is identified as KNN which takes 0.00398 seconds.

Conclusion

This is as interesting & challenging assignment with the freedom to choose the dataset. The first challenge of the assignment is to choosing an ideal, as well as challenging dataset. The site provides many dataset, identifying the classification is vital.

Even though the dataset does not contain, the data has other preprocessing challenge. The dataset was preprocessed to limit the instances to have only 2 types of wheats, since certain function (such as : ROCR ROC, CaretStack) expects an binary predictions.

Further, the dataset had the classes sorted as stated in the previous section of the document. When splitting for training and testing there is a high change that model is biased for the first 2 type and less biased for the 2nd class .Therefore to improve the accuracy of the model, the rows of the dataset was randomized prior to splitting the data.

Finally, the results indicate, using preprocessing and experimenting multiple parameters has produced impressive accuracy. Through out the outputs shown, we can see that values are very closer to one with high accuracy.

References

- https://rpubs.com/kangrinboqe/268745
- https://rpubs.com/whitet1/384231
- http://www.sthda.com/english/articles/35-statistical-machine-learning-essentials/140-bagging-and-random-forest-essentials/
- https://machinelearningmastery.com/machine-learning-ensembles-with-r/44
- https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
- https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/
- http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/
- https://rpubs.com/whitet1/384231
- https://stackoverflow.com/questions/8499361/easy-way-of-counting-precision-recall-and-f1-score-in-r

Appendix – Full code

```
# load libraries

library(caret)

library(klaR)

library(mlbench)

library(caretEnsemble)

library(dplyr)

library(ROCR)

library(ggplot2)

library(MLeval)

library(pROC)

library(rpart)

library(plotROC)



###-------------------------------------------------


# load the dataset

dataset <- read.csv("C:/Users/jayasaigoutheman/Desktop/Datamining/Dataset/seeds.csv")


dataset = dataset[dataset$Wheat!="Canadian", ]

dataset$Wheat = factor(dataset$Wheat)


#randomise order

dataset <- dataset[sample(1:nrow(dataset)), ]


#understanding data

summary(dataset)
```

```
head(dataset)
str(dataset)
glimpse(dataset)
boxplot(dataset)

#check if dataset contains any missing value
sum(is.na(dataset))



# calculate correlation matrix
correlationMatrix <- cor(dataset[,1:7])
# summarize the correlation matrix
print(correlationMatrix)
# find attributes that are highly corrected (ideally >0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.5)
# print indexes of highly correlated attributes
print(highlyCorrelated)

# define an 80%/20% train/test split of the dataset
split=0.80
trainIndex <- createDataPartition(dataset$Wheat, p=split, list=FALSE)
data_train <- dataset[ trainIndex,]
data_test <- dataset[-trainIndex,]

###Bagging
###------------------------------------------------
set.seed(12345)
metric <- "Accuracy"
```

```
# define training control

control <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions=TRUE,
classProbs=TRUE)

model <- train(Wheat~., data=data_train, trControl=control, method="nb")

print(model)


# estimate variable importance

importance <- varImp(model, scale=FALSE)

# summarize importance

print(importance)

# plot importance

plot(importance)



model.treebag <- train(Wheat~., data=data_train, method="treebag", metric=metric, trControl=control)

model.rf <- train(Wheat ~., data = data_train, method="rf", metric=metric, trControl=control)


# Summarize the results

print(model.treebag)

print(model.rf)



bagging_results <- resamples(list(treebag=model.treebag, rf=model.rf))


summary(bagging_results)

dotplot(bagging_results)



#prediction
```

```
pred.treebag <- predict(model.treebag, data_test , type='raw')
pred.rf <- predict(model.rf, data_test)

pred.treebag
pred.rf

#confusion matrix
result.treebag <- confusionMatrix(data_test$Wheat,pred.treebag)
result.rf <- confusionMatrix(data_test$Wheat,pred.rf)

result.treebag
result.rf

#Precision & Recall estimation
result.treebag$byClass
result.rf$byClass

#Precision Recall, ROC & RUAC : treebag
pred.treebag <- prediction(as.numeric(pred.treebag), as.numeric(data_test$Wheat))
RP.treebag <- performance(pred.treebag, "prec", "rec")
pred.treebag
plot (RP.treebag)
abline(a = 0, b = 1)

ROC.treebag <- performance(pred.treebag, "tpr", "fpr");
ROC.treebag
plot (ROC.treebag)
abline(a = 0, b = 1)
```

```
auc.treebag <- performance(pred.treebag, measure = "auc")

auc.treebag <- auc.treebag@y.values[[1]]

auc.treebag

-------

pred.rf <- prediction(as.numeric(pred.rf), as.numeric(data_test$Wheat))

RP.rf <- performance(pred.rf, "prec", "rec")

plot (RP.rf)

abline(a = 0, b = 1)


ROC.rf <- performance(pred.rf, "tpr", "fpr");

plot (ROC.rf)

abline(a = 0, b = 1)


auc.rf <- performance(pred.rf, measure = "auc")

auc.rf<- auc.rf@y.values[[1]]

auc.rf




###-----------------------------------------------------


start.time.train <- Sys.time()

model.treebag <- train(Wheat~., data=data_train, method="treebag", metric=metric, trControl=control)

end.time.train <- Sys.time()

time.taken.train <- end.time.train - start.time.train


# Time taken to train Treebag

time.taken.train
```

```
###-------------------------------------------------

start.time.train <- Sys.time()
model.rf <- train(Wheat ~., data = data_train, method="rf", metric=metric, trControl=control)
end.time.train <- Sys.time()
time.taken.train <- end.time.train - start.time.train

# Time taken to train Random Forest
time.taken.train

###-------------------------------------------------

start.time.test <- Sys.time()
pred.treebag = predict(model.treebag, data_test)
end.time.test <- Sys.time()
time.taken.test <- end.time.test - start.time.test

# Time taken to test treebag
time.taken.test

###-------------------------------------------------

start.time.test <- Sys.time()
pred.rf = predict(model.rf, data_test)
end.time.test <- Sys.time()
time.taken.test <- end.time.test - start.time.test

# Time taken to test Random Forest
time.taken.test
```

```
###Stacking

###--------------------------------------------------


algorithmList <- c('rpart', 'knn', 'nb')

set.seed(12345)

models <- caretList(Wheat~., data=data_train, trControl=control, methodList=algorithmList)

results <- resamples(models)


summary(results)

dotplot(results)


# correlation between results

modelCor(results)

splom(results)


# stack using knn

stack <- caretStack(models, method="knn", metric="Accuracy", trControl=control)

stack




#prediction: rpart

stackpredicted.rpart <- predict(models$rpart, data_test)

result.rpart <- confusionMatrix(stackpredicted.rpart, data_test$Wheat)


#prediction: knn

stackpredicted.knn <- predict(models$knn, data_test)
```

```
result.knn <- confusionMatrix(stackpredicted.knn, data_test$Wheat)

#prediction: nb
stackpredicted.nb <- predict(models$nb, data_test)
result.nb <- confusionMatrix(stackpredicted.nb, data_test$Wheat)

#confusion table
result.rpart
result.knn
result.nb

#Precision & Recall estimation
result.rpart$byClass
result.knn$byClass
result.nb$byClass



#ROC & RUAC : RPART
predicted.rpart <- prediction(as.numeric(stackpredicted.rpart), as.numeric(data_test$Wheat))
perf.treebag <- performance(predicted.rpart, measure = "tpr", x.measure = "fpr")
plot(perf.treebag, main = "ROC curve",colorize = T)
abline(a = 0, b = 1)

auc.treebag <- performance(predicted.rpart, measure = "auc")
pred.treebag <- auc.treebag@y.values[[1]]
pred.treebag
```

```
#ROC & RUAC : KNN

predicted.knn <- prediction(as.numeric(stackpredicted.knn), as.numeric(data_test$Wheat))

perf.knn <- performance(predicted.knn, measure = "tpr", x.measure = "fpr")

plot(perf.knn, main = "ROC curve",colorize = T)

abline(a = 0, b = 1)


auc.knn <- performance(predicted.knn, measure = "auc")

stackpredicted.knn <- auc.knn@y.values[[1]]

stackpredicted.knn



#ROC & RUAC : NB

predicted.nb <- prediction(as.numeric(stackpredicted.nb), as.numeric(data_test$Wheat))

perf.nb <- performance(predicted.nb, measure = "tpr", x.measure = "fpr")

plot(perf.nb, main = "ROC curve",colorize = T)

abline(a = 0, b = 1)


auc.nb <- performance(predicted.nb, measure = "auc")

stackpredicted.nb <- auc.knn@y.values[[1]]

stackpredicted.nb



###-------------------------------------------------



start.time.train <- Sys.time()

models <- caretList(Wheat~., data=data_train, trControl=control, methodList=algorithmList)

end.time.train <- Sys.time()

time.taken.train <- end.time.train - start.time.train
```

```
# Time taken to train stacking

time.taken.train


start.time.test <- Sys.time()

stackpredicted.rpart <- predict(models$rpart, data_test)

end.time.test <- Sys.time()

time.taken.test <- end.time.test - start.time.test


# Time taken to test treebag

time.taken.test


start.time.test <- Sys.time()

stackpredicted.knn <- predict(models$knn, data_test)

end.time.test <- Sys.time()

time.taken.test <- end.time.test - start.time.test


# Time taken to test treebag

time.taken.test


start.time.test <- Sys.time()

stackpredicted.nb <- predict(models$nb, data_test)

end.time.test <- Sys.time()

time.taken.test <- end.time.test - start.time.test


# Time taken to test treebag

time.taken.test
```