Random Forest

Ensembled Learning

- Random Forest is an ensembled learning approach
- In ensembled learning approach, multiple predictive models are developed and results are aggregated to improve the precision

Random Forest

- In this algorithm, the observations as well as variables are sampled to create multiple decision trees
- Each observation is classified by each decision tree.
- The outcome is considered as per the majority in different trees

Algorithm
(Considering N observations & M variables)

1. Sample out of N cases with replacement from the training set, many samples. Consider each sample as root node for the decision trees to be constructed.
2. Choose some $m < M$ number of variable by sampling at each node created in step 1.
3. Grow each tree without pruning with minimum node size as 1
4. Classify the validation / test set observations by traversing them through all the grown trees.
5. Classify each outcome by a majority vote of the trees.

OOB

- Each tree is constructed using a different bootstrap sample from the original data.
- About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree.
- Put each case left out in the construction of the kth tree, traverse the kth tree to get a classification.
- In this way, a test set classification is obtained for each case in about one-third of the trees. At the end of the run, take j to be the class that got most of the votes every time case n was oob.
- The proportion of times that j is not equal to the true class of n averaged over all cases is the oob error estimate. This has proven to be unbiased in many tests.
- This is done internally with the training set

Random Forest in R

- For Traditional Decision Trees (rpart type), we have function randomForest() package randomForest
- For conditional inference trees (party type), we have cforest() in package party

Package randomForest

- randomForest() in package randomForest implements Breiman's random forest algorithm

Syntax : randomForest(formula, data, ntree, na.action, importance,…)

Where

formula : Model formula

data : data frame object with train set

ntree : Number of trees to be grown

na.action : Action to be taken on NA values

importance : whether to create importance measure while creating randomForest object

Variable Importance

- The function importance() gives the list of variables along with the measure of their importance in terms of the purity gained by them.

Syntax : importance(objRF , type, …)

Where

objRF : Object of class randomForest

type : either 1 or 2, specifying the type of importance measure (1=mean decrease in accuracy, 2=mean decrease in node impurity).

Example: Classification

- We will be using dataset churn from the package C50.
- The dataset has already been partitioned by its creators into two frames:
  - churnTrain : Training Set
  - churnTest : Validation / Test Set
- Intension is to predict the churning of the telecom customers

Attributes

- The dataset churn has the following attributes:
  - state(categorical)
  - account_length : how long account has been active
  - area_code
  - international_plan (yes/no)
  - voice_mail_plan (yes/no)
  - number_vmail_messages
  - total_day_minutes : minutes customer used service during the day
  - total_day_calls : total daily calls
  - total_day_charge : perhaps based on foregoing two variables

Attributes

- Attributes (Contd.):
  - total_eve_minutes : minutes customer used service during the evening
  - total_eve_calls : total evening calls
  - total_eve_charge : perhaps based on foregoing two variables
  - total_night_minutes : minutes customer used service during the night
  - total_night_calls : total night calls
  - total_night_charge : perhaps based on foregoing two variables
  - total_intl_minutes : minutes customer used service to make international calls
  - total_intl_calls : total international calls
  - total_intl_charge : perhaps based on foregoing two variables
  - number_customer_service_calls : Number of calls to customer service
  - churn (Response variable)

Program & Output

```
library(randomForest)
model.RF <- randomForest(churn ~ . , data = churnTrain ,
                         na.action=na.roughfix, importance=TRUE)
```

na.roughfix imputes median/mode for missing values

```
> model.RF

Call:
 randomForest(formula = churn ~ ., data = churnTrain, importance = TRUE,      na.action
= na.roughfix)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 6.48%
Confusion matrix:
    yes   no class.error
yes 397   86  0.17805383
no  130 2720  0.04561404
```

Importance

```
> importance(model.RF,type = 2)
                              MeanDecreaseGini
state                               110.643986
account_length                       20.877809
area_code                             4.566489
international_plan                   56.297338
voice_mail_plan                      14.256491
number_vmail_messages                21.713792
total_day_minutes                   104.049412
total_day_calls                      21.995356
total_day_charge                    102.438042
total_eve_minutes                    44.623016
total_eve_calls                      19.105654
total_eve_charge                     44.944149
total_night_minutes                  25.375495
total_night_calls                    20.275212
total_night_charge                   24.307408
total_intl_minutes                   30.511743
total_intl_calls                     35.962697
total_intl_charge                    30.884003
number_customer_service_calls        93.451117
```

Evaluation

```
> pred <- predict(model, newdata = churnTest[,-20])
>
> confusionMatrix( table(pred, churnTest$churn) )
Confusion Matrix and Statistics


pred   yes   no
  yes  185   40
  no    39 1403

               Accuracy : 0.9526
                 95% CI : (0.9413, 0.9623)
    No Information Rate : 0.8656
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7967
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8259
            Specificity : 0.9723
         Pos Pred Value : 0.8222
         Neg Pred Value : 0.9730
             Prevalence : 0.1344
         Detection Rate : 0.1110
   Detection Prevalence : 0.1350
      Balanced Accuracy : 0.8991

       'Positive' Class : yes
```

Example: Numerical Response

- Consider the dataset Boston in package MASS with the following attributes:
  - crim: per capita crime rate by town.
  - zn: proportion of residential land zoned for lots over 25,000 sq.ft.
  - indus: proportion of non-retail business acres per town.
  - chas: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
  - nox: nitrogen oxides concentration (parts per 10 million).
  - rm: average number of rooms per dwelling.
  - age: proportion of owner-occupied units built prior to 1940.
  - dis: weighted mean of distances to five Boston employment centres.
  - rad: index of accessibility to radial highways.
  - tax: full-value property-tax rate per \$10,000.
  - ptratio: pupil-teacher ratio by town.
  - black: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
  - lstat: lower status of the population (percent).
  - medv: median value of owner-occupied homes in \$1000s.

Program & Output

```
library(randomForest)
model.RF <- randomForest(medv ~ . , data = training ,
                         na.action=na.roughfix, importance=TRUE)
```

```
> model.RF

Call:
 randomForest(formula = medv ~ ., data = training, importance = TRUE,      na.action = 
 na.roughfix)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 12.3705
                    % Var explained: 86.08
```

Variable Importance

```
> importance(model.RF,type = 2)
        IncNodePurity
crim       1873.85795
zn          153.04752
indus      1661.92180
chas         85.53658
nox        2249.01481
rm         9214.13244
age         808.49074
dis        1773.60915
rad         295.72698
tax        1221.70373
ptratio    1876.61594
black       647.32592
lstat      9258.32671
```

Evaluation

```
> pred.RF <- predict(model.RF, newdata = validation[,-14])
>
> postResample(pred.RF , validation$medv)
     RMSE  Rsquared
2.9496507 0.8871838
>
> MAPE <- function(y, yhat) {
+   mean(abs((y - yhat)/y))
+ }
>
> MAPE(validation$medv , pred.RF)
[1] 0.1020432
>
> RMSPE<- function(y, yhat) {
+   sqrt(mean((y-yhat)/y)^2)
+ }
>
> RMSPE(validation$medv , pred.RF)
[1] 0.04299174
```

Using cforest()

```
library(party)
model.RF <- cforest(medv ~ ., data = Boston,
                    control = cforest_unbiased(ntree = 50))
```

```
> pred.RF <- predict(model.RF, newdata = validation[,-14])
>
> postResample(pred.RF , validation$medv)
     RMSE  Rsquared
2.9072959 0.8913023
>
> MAPE <- function(y, yhat) {
+   mean(abs((y - yhat)/y))
+ }
>
> MAPE(validation$medv , pred.RF)
[1] 0.09490211
>
> RMSPE<- function(y, yhat) {
+   sqrt(mean((y-yhat)/y)^2)
+ }
>
> RMSPE(validation$medv , pred.RF)
[1] 0.04144711
```