Experiment No 3

Title: Random Forest and Parameter Tuning in R

Introduction

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method. Ensembling is nothing but a combination of weak learners (individual trees) to produce a strong learner.

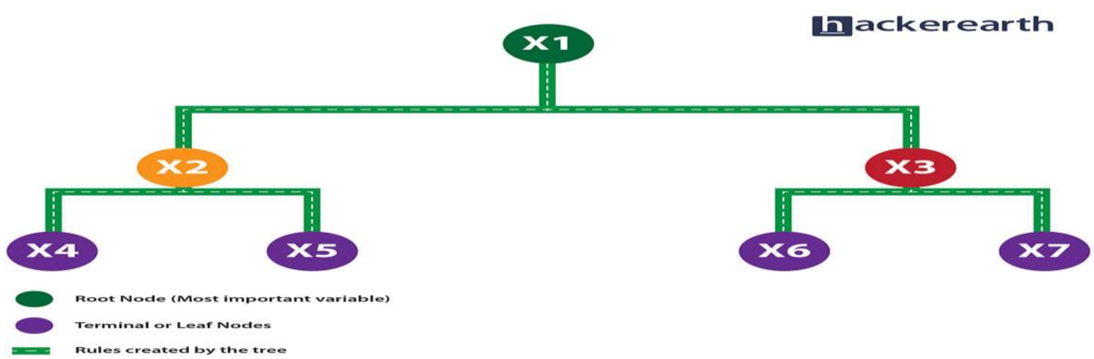
Say, you want to watch a movie. But you are uncertain of its reviews. You ask 10 people who have watched the movie. 8 of them said " the movie is fantastic." Since the majority is in favor, you decide to watch the movie. This is how we use ensemble techniques in our daily life too.

Random Forest can be used to solve regression and classification problems. In regression problems, the dependent variable is continuous. In classification problems, the dependent variable is categorical.

The random Forest algorithm was created by Leo Brieman and Adele Cutler in 2001.

How does it work? (Decision Tree, Random Forest)

To understand the working of a random forest, it's crucial to understand a **tree**. A tree works in the following way:



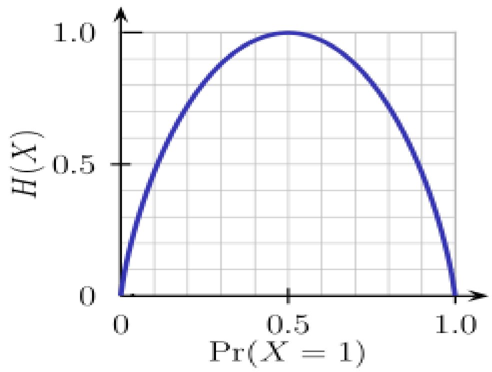
1. Given a data frame (n x p), a tree stratifies or partitions the data based on rules (if-else). Yes, a tree creates rules. These rules divide the data set into distinct and non-overlapping regions. These rules are determined by a variable's contribution to the homogeneity or pureness of the resultant child nodes (X2,X3).

2. In the image above, the variable X1 resulted in highest homogeneity in child nodes, hence it became the root node. A variable at root node is also seen as the most important variable in the data set.

3. But how is this homogeneity or pureness determined? In other words, how does the tree decide at which variable to split?

- In **regression trees** (where the output is predicted using the mean of observations in the terminal nodes), the splitting decision is based on minimizing RSS. The variable which leads to the greatest possible reduction in RSS is chosen as the root node. The tree splitting takes a **top-down greedy** approach, also known as *recursive binary splitting*. We call it "greedy" because the algorithm cares to make the best split at the current step rather than saving a split for better results on future nodes.
- In **classification trees** (where the output is predicted using mode of observations in the terminal nodes), the splitting decision is based on the following methods:
    - **Gini Index** - It's a measure of node purity. If the Gini index takes on a smaller value, it suggests that the node is pure. For a split to take place, the Gini index for a child node should be less than that for the parent node.
    - **Entropy** - Entropy is a measure of node impurity. For a binary class (a,b), the formula to calculate it is shown below. Entropy is maximum at p = 0.5. For p(X=a)=0.5 or p(X=b)=0.5 means, a new observation has a 50%-50% chance of getting classified in either classes. The entropy is minimum when the probability is 0 or 1.

Entropy = - p(a)*log(p(a)) - p(b)*log(p(b))



In a nutshell, every tree attempts to create rules in such a way that the resultant terminal nodes could be as pure as possible. Higher the purity, lesser the uncertainty to make the decision. But a decision tree suffers from high variance. "High Variance" means getting high prediction error on unseen data. We can overcome the variance problem by using more data for training. But since the data set available is limited to us, we can use resampling techniques like bagging and random forest to generate more data.

Building many **decision trees** results in a **forest**. A random forest works the following way:

1. First, it uses the Bagging (Bootstrap Aggregating) algorithm to create random samples. Given a data set D1 (n rows and p columns), it creates a new dataset (D2) by sampling n cases at random with replacement from the original data. About 1/3 of the rows from D1 are left out, known as Out of Bag(OOB) samples.
2. Then, the model trains on D2. OOB sample is used to determine unbiased estimate of the error.
3. Out of p columns, P << p columns are selected at each node in the data set. The P columns are selected at random. Usually, the default choice of P is p/3 for regression tree and P is sqrt(p) for classification tree.
4. Unlike a tree, no pruning takes place in random forest; i.e, each tree is grown fully. In decision trees, pruning is a method to avoid overfitting. Pruning means selecting a subtree that leads to the lowest test errror rate. We can use cross validation to determine the test error rate of a subtree.
5. Several trees are grown and the final prediction is obtained by averaging or voting.

Each tree is grown on a different sample of original data. Since random forest has the feature to calculate OOB error internally, cross validation doesn't make much sense in random forest.

Advantages and Disadvantages of Random Forest

Advantages are as follows:

1. It is robust to correlated predictors.
2. It is used to solve both regression and classification problems.
3. It can be also used to solve unsupervised ML problems.
4. It can handle thousands of input variables without variable selection.
5. It can be used as a feature selection tool using its variable importance plot.
6. It takes care of missing data internally in an effective manner.

Disadvantages are as follows:

1. The Random Forest model is difficult to interpret.
2. It tends to return erratic predictions for observations out of range of training data. For example, the training data contains two variable x and y. The range of x variable is 30 to 70. If the test data has x = 200, random forest would give an unreliable prediction.
3. It can take longer than expected time to computer a large number of trees.

R Program

```
install.packages("randomForest")

library(randomForest)

data1 <- read.csv(file.choose(), header = TRUE)

head(data1)

str(data1)

summary(data1)

# Split into Train and Validation sets

# Training Set : Validation Set = 70 : 30 (random)
```

```
set.seed(100)

train <- sample(nrow(data1), 0.7*nrow(data1), replace = FALSE)

TrainSet <- data1[train,]

ValidSet <- data1[-train,]

summary(TrainSet)

summary(ValidSet)

# Create a Random Forest model with default parameters

model1 <- randomForest(Condition ~ ., data = TrainSet, importance = TRUE)

model1

# Fine tuning parameters of Random Forest model

model2 <- randomForest(Condition ~ ., data = TrainSet, ntree = 500, mtry = 6, importance =
TRUE)

model2

# Predicting on train set

predTrain <- predict(model2, TrainSet, type = "class")

# Checking classification accuracy

table(predTrain, TrainSet$Condition)

# Predicting on Validation set

predValid <- predict(model2, ValidSet, type = "class")

# Checking classification accuracy

mean(predValid == ValidSet$Condition)

table(predValid,ValidSet$Condition)

# To check important variables

importance(model2)

varImpPlot(model2)
```

Output of the program and analysis

```
> library(randomForest)
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
> data1 <- read.csv(file.choose(), header = TRUE)

> head(data1)
   Buying   maint   doors   persons  lug_boot   safety   Condition
1   vhigh   vhigh   2       2        small      low      unacc
2   vhigh   vhigh   2       2        small      med      unacc
3   vhigh   vhigh   2       2        small      high     unacc
4   vhigh   vhigh   2       2        med        low      unacc
5   vhigh   vhigh   2       2        med        med      unacc
6   vhigh   vhigh   2       2        med        high     unacc
> str(data1)
'data.frame':    1728 obs. of  7 variables:
 $ buying   : Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ maint    : Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ doors    : Factor w/ 4 levels "2","3","4","5more": 1 1 1 1 1 1 1 1 1 1 ...
 $ persons  : Factor w/ 3 levels "2","4","more": 1 1 1 1 1 1 1 1 1 2 ...
 $ lug_boot : Factor w/ 3 levels "big","med","small": 3 3 3 2 2 2 1 1 1 3 ...
 $ safety   : Factor w/ 3 levels "high","low","med": 2 3 1 2 3 1 2 3 1 2 ...
 $ Condition: Factor w/ 4 levels "acc","good","unacc",..: 3 3 3 3 3 3 3 3 3 3 ...
> summary(data1)
   buying     maint      doors     persons    lug_boot
 high :432   high :432   2    :432   2   :576   big  :576
 low  :432   low  :432   3    :432   4   :576   med  :576
 med  :432   med  :432   4    :432   more:576   small:576
 vhigh:432   vhigh:432   5more:432

  safety    Condition
 high:576   acc  : 384
 low :576   good :  69
 med :576   unacc:1210
            vgood:  65
> set.seed(100)
> train <- sample(nrow(data1), 0.7*nrow(data1), replace = FALSE)
> TrainSet <- data1[train,]
> ValidSet <- data1[-train,]
> summary(TrainSet)
   buying     maint      doors     persons    lug_boot
 high :298   high :303   2    :312   2   :407   big  :406
 low  :300   low  :302   3    :298   4   :409   med  :393
```

```
 med  :306   med  :312   4   :299   more:393   small:410
 vhigh:305   vhigh:292   5more:300
  safety    Condition
 high:396   acc  :260
 low :412   good : 46
 med :401   unacc:856
            vgood: 47
> summary(ValidSet)
   buying     maint      doors     persons    lug_boot
 high :134   high :129   2   :120   2   :169   big  :170
 low  :132   low  :130   3   :134   4   :167   med  :183
 med  :126   med  :120   4   :133   more:183   small:166
 vhigh:127   vhigh:140   5more:132
  safety    Condition
 high:180   acc  :124
 low :164   good : 23
 med :175   unacc:354
            vgood: 18


> model1 <- randomForest(Condition ~ ., data = TrainSet, importance = TRUE)
> model1

Call:
 randomForest(formula = Condition ~ ., data = TrainSet, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 3.64%
Confusion matrix:
        acc   good   unacc   vgood   class.error
acc     255   2      2       1       0.01923077
good     7    35     0       4       0.23913043
unacc   20    2      834     0       0.02570093
vgood   6      0     0       41      0.12765957
```

By default, number of trees is 500 and number of variables tried at each split is 2 in this case. Err or rate is 3.6%

```
> model2 <- randomForest(Condition ~ ., data = TrainSet, ntree = 500, mtry = 6, importance = TR
UE)
> model2
```

Ntree: Number of trees to grow. This should not be set to too small a number, to ensure that ever y input row gets predicted at least a few times.

Mtry: Number of variables randomly sampled as candidates at each split. Note that the default va lues are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)

```
Call:
 randomForest(formula = Condition ~ ., data = TrainSet, ntree = 500,      mtry = 6, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 6

        OOB estimate of  error rate: 2.32%
Confusion matrix:
      acc good unacc vgood class.error
acc   248    6     5     1  0.04615385
good    4   42     0     0  0.08695652
unacc   8    2   846     0  0.01168224
vgood   2    0     0    45  0.04255319
```

When we have increased the mtry to 6 from 2, error rate has reduced from 3.6% to 2.32%. We wi ll now predict on the train dataset first and then predict on validation dataset.

```
> predTrain <- predict(model2, TrainSet, type = "class")
> table(predTrain, TrainSet$Condition)

predTrain acc good unacc vgood
    acc   260    0     0     0
    good    0   46     0     0
    unacc   0    0   856     0
    vgood   0    0     0    47
> predValid <- predict(model2, ValidSet, type = "class")
> mean(predValid == ValidSet$Condition)
[1] 0.9845857
> table(predValid,ValidSet$Condition)

predValid acc good unacc vgood
    acc   120    1     2     1
    good    1   22     0     0
    unacc   3    0   352     0
    vgood   0    0     0    17
```

In case of prediction on train dataset, there is zero misclassification; however, in the case of valid ation dataset, 6 data points are misclassified and accuracy is 98.84%. We can also use function to check important variables. The below functions show the drop in mean accuracy for each of the v ariables.

```
> importance(model2)
```

```
             acc     good    unacc    vgood
buying   144.13929 75.96633 111.10092  80.70126
maint    134.88327 69.62648 104.31162  50.07345
doors     32.35052 17.55486  47.57988  20.17438
persons  150.37837 50.89904 186.53684  57.04931
lug_boot  85.05941 55.85293  83.13938  63.39719
safety   176.85992 82.14649 201.91053 110.32306
         MeanDecreaseAccuracy MeanDecreaseGini
buying               200.4809         68.49384
maint                182.8435         91.02632
doors                 57.3249         33.93850
persons              237.0746        122.51556
lug_boot             144.7800         75.31990
safety               277.8490        151.59471
> varImpPlot(model2
```