# Tree-Based Methods on Prediction of Tropical Storms

*Zoe Chen*

*02/25/2019*

## 0. Introduction

For this data analysis case, we will focus on tree-based models to analyze a climate data set from Mann and Sabbatelli 2007. Basically, we will build regression trees to predict the number of tropical storms and classification trees to classify high frequency storms from others. Furthermore, we will build random forests and compare their performance with single-tree models.

## 1. Importing Data set

Let's import the data set first and take a look at it. This data set contains 113 rows, each row represents each year's data. Besides, it has totally 6 columns. In this case, the response variable is `Storms`(the number of tropical storms) and the predictors are `Temp`(sea surface temperature), `ENSO`(the EL Nino Index), `NAO`(North Atlantic Oscillator index).

```
df = read.csv('./TCcount.csv')
dim(df)
```

```
## [1] 113   6
```

```
colnames(df) = c('Year', 'Storms', 'Landfalling', 'Temp', 'ENSO', 'NAO')
head(df)
```

```
##   Year Storms Landfalling    Temp        ENSO     NAO
## 1 1900      7           4 27.0702  0.56385003  0.2475
## 2 1901     12           2 26.7827 -0.02228333 -0.2175
## 3 1902      5           0 26.9269  1.65433330  2.8650
## 4 1903     10           2 26.5557 -0.84288340  0.4150
## 5 1904      5           2 26.5955  1.23696670  1.7525
## 6 1905      5           0 26.7134  1.29413340  1.3750
```

## 2. Splitting Data Set

Next, we want to randomly split our original data set into two parts, specifically 80% of it as the training set and the other 20% as the test set. Thus, now we have `train.df` and `test.df` ready at hand.

```
set.seed(111)
rows = sample(nrow(df), size = .8*nrow(df), replace = F)
train.df = df[rows,]
test.df = df[-rows,]
```

# 3. Building Regression Tree

Firstly, we want to use our training set to fit a single-tree model using the R function `rpart`. Here, the formula is just like what we mentioned above. Since it's a regression problem, we set the `method` argument to be `anova`.

After we fit the model, we can use `summary`, `rpart.plot`, `printcp` to take a deeper look at the fitted tree model. In this model, it has totally 5 splits and the most important variable is `NAO`.

```
library('rpart')
```

```
## Warning: package 'rpart' was built under R version 3.5.2
```

```
tree = rpart(Storms~Temp+ENSO+NAO, data=train.df, method='anova')
summary(tree)
```

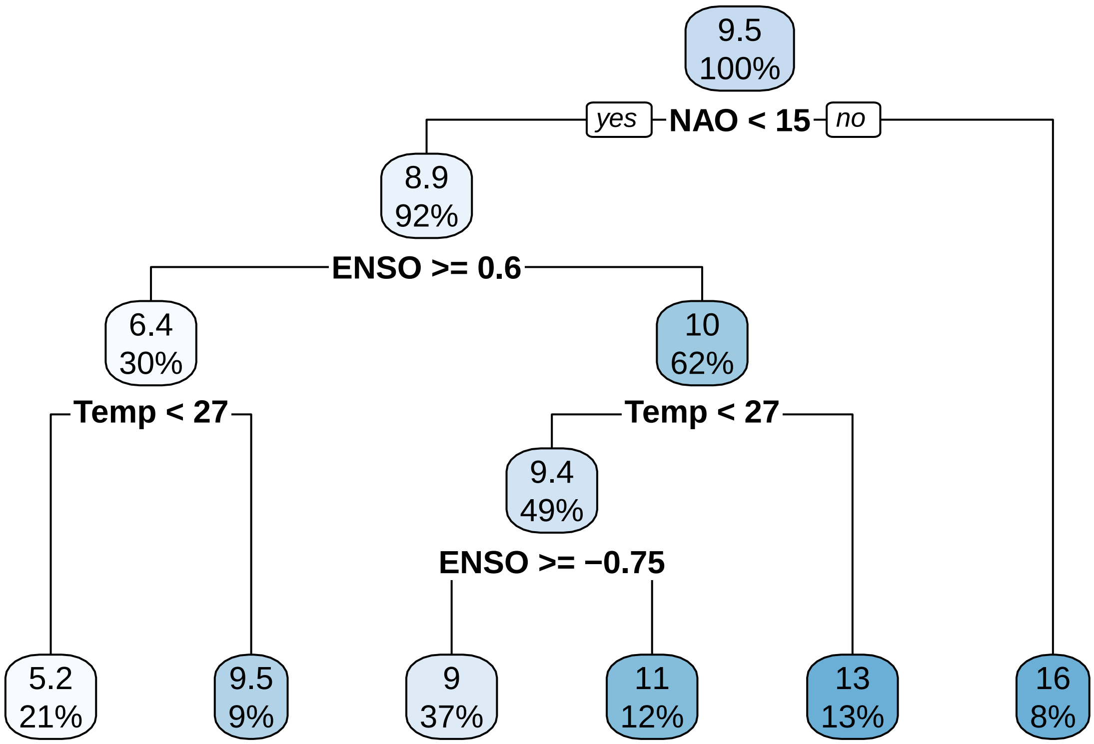
```
## Call:
## rpart(formula = Storms ~ Temp + ENSO + NAO, data = train.df,
##     method = "anova")
##   n= 90
##
##           CP nsplit rel error    xerror      xstd
## 1 0.20764151      0 1.0000000 1.0140164 0.1993746
## 2 0.15247222      1 0.7923585 1.0557666 0.2091270
## 3 0.06557991      2 0.6398863 0.9343142 0.1646788
## 4 0.06502277      3 0.5743064 0.8563238 0.1507045
## 5 0.01364910      4 0.5092836 0.8257105 0.1443453
## 6 0.01000000      5 0.4956345 0.8300152 0.1418998
##
## Variable importance
## ENSO  NAO Temp
##   46   29   25
##
## Node number 1: 90 observations,    complexity param=0.2076415
##   mean=9.488889, MSE=17.98321
##   left son=2 (83 obs) right son=3 (7 obs)
##   Primary splits:
##       NAO  < 15.0275    to the left,  improve=0.2076415, (0 missing)
##       ENSO < 14.75696   to the left,  improve=0.2076415, (0 missing)
##       Temp < 27.37925   to the left,  improve=0.1453362, (0 missing)
##   Surrogate splits:
##       ENSO < 14.75696   to the left,  agree=1.000, adj=1.000, (0 split)
##       Temp < 26.41545   to the right, agree=0.944, adj=0.286, (0 split)
##
## Node number 2: 83 observations,    complexity param=0.1524722
##   mean=8.927711, MSE=12.83815
##   left son=4 (27 obs) right son=5 (56 obs)
##   Primary splits:
##       ENSO < 0.6030456  to the right, improve=0.2315901, (0 missing)
##       Temp < 27.2664    to the left,  improve=0.1980226, (0 missing)
##       NAO  < 0.28875    to the right, improve=0.1183829, (0 missing)
##   Surrogate splits:
##       Temp < 26.74315   to the left,  agree=0.723, adj=0.148, (0 split)
```

```
##       NAO  < 2.405      to the right, agree=0.699, adj=0.074, (0 split)
##
## Node number 3: 7 observations
##   mean=16.14286, MSE=30.97959
##
## Node number 4: 27 observations,    complexity param=0.06557991
##   mean=6.444444, MSE=10.61728
##   left son=8 (19 obs) right son=9 (8 obs)
##   Primary splits:
##       Temp < 27.2356    to the left,  improve=0.37025700, (0 missing)
##       NAO  < 0.47625    to the right, improve=0.25555840, (0 missing)
##       ENSO < 0.954325   to the right, improve=0.05315615, (0 missing)
##   Surrogate splits:
##       NAO  < -0.49125   to the right, agree=0.815, adj=0.375, (0 split)
##       ENSO < 1.821212   to the left,  agree=0.741, adj=0.125, (0 split)
##
## Node number 5: 56 observations,    complexity param=0.06502277
##   mean=10.125, MSE=9.502232
##   left son=10 (44 obs) right son=11 (12 obs)
##   Primary splits:
##       Temp < 27.37925   to the left,  improve=0.19777050, (0 missing)
##       NAO  < 0.20625    to the right, improve=0.08069883, (0 missing)
##       ENSO < -0.7372568 to the right, improve=0.05048068, (0 missing)
##   Surrogate splits:
##       ENSO < 0.460807   to the left,  agree=0.804, adj=0.083, (0 split)
##
## Node number 8: 19 observations
##   mean=5.157895, MSE=3.606648
##
## Node number 9: 8 observations
##   mean=9.5, MSE=14
##
## Node number 10: 44 observations,    complexity param=0.0136491
##   mean=9.409091, MSE=6.78719
##   left son=20 (33 obs) right son=21 (11 obs)
##   Primary splits:
##       ENSO < -0.7458584 to the right, improve=0.07397260, (0 missing)
##       NAO  < 0.29375    to the right, improve=0.03970680, (0 missing)
##       Temp < 27.2893    to the left,  improve=0.03896499, (0 missing)
##   Surrogate splits:
##       Temp < 26.7073    to the right, agree=0.795, adj=0.182, (0 split)
##
## Node number 11: 12 observations
##   mean=12.75, MSE=10.6875
##
## Node number 20: 33 observations
##   mean=9, MSE=7.212121
##
## Node number 21: 11 observations
##   mean=10.63636, MSE=3.504132
```

```r
library('rpart.plot')
```

```
## Warning: package 'rpart.plot' was built under R version 3.5.2
```

```r
rpart.plot(tree)
```




```r
printcp(tree)
```

```
##
## Regression tree:
## rpart(formula = Storms ~ Temp + ENSO + NAO, data = train.df,
##     method = "anova")
##
## Variables actually used in tree construction:
## [1] ENSO NAO  Temp
##
## Root node error: 1618.5/90 = 17.983
##
## n= 90
##
##         CP nsplit rel error  xerror    xstd
## 1 0.207642      0   1.00000 1.01402 0.19937
## 2 0.152472      1   0.79236 1.05577 0.20913
## 3 0.065580      2   0.63989 0.93431 0.16468
## 4 0.065023      3   0.57431 0.85632 0.15070
## 5 0.013649      4   0.50928 0.82571 0.14435
## 6 0.010000      5   0.49563 0.83002 0.14190
```

# 4. Pruning Tree

Most of the arguments in the first tree model are set as function default values. However, sometimes it would still be too complex. Thus here we would want to prune this model a little bit using R function `prune`, with `cp` which has the minimum cross validation error value.

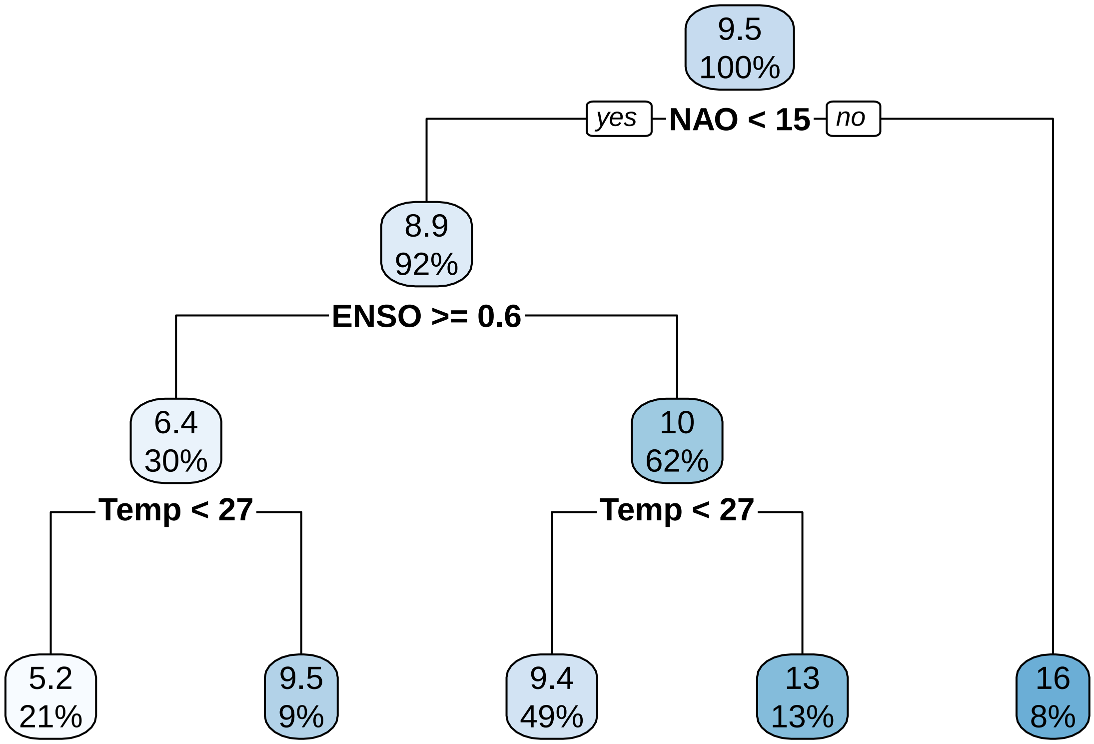
From the model result we see that the pruned tree indeed becomes simpler with less decision nodes. Now it only has 4 splits.

```
tree.pruned = prune(tree, cp=tree$cptable[which.min(tree$cptable[,'xerror']),'CP'])
summary(tree.pruned)
```

```
## Call:
## rpart(formula = Storms ~ Temp + ENSO + NAO, data = train.df,
##     method = "anova")
##   n= 90
##
##           CP nsplit rel error    xerror      xstd
## 1 0.20764151      0 1.0000000 1.0140164 0.1993746
## 2 0.15247222      1 0.7923585 1.0557666 0.2091270
## 3 0.06557991      2 0.6398863 0.9343142 0.1646788
## 4 0.06502277      3 0.5743064 0.8563238 0.1507045
## 5 0.01364910      4 0.5092836 0.8257105 0.1443453
##
## Variable importance
## ENSO  NAO Temp
##   45   29   26
##
## Node number 1: 90 observations,    complexity param=0.2076415
##   mean=9.488889, MSE=17.98321
##   left son=2 (83 obs) right son=3 (7 obs)
##   Primary splits:
##       NAO  < 15.0275    to the left,  improve=0.2076415, (0 missing)
##       ENSO < 14.75696   to the left,  improve=0.2076415, (0 missing)
##       Temp < 27.37925   to the left,  improve=0.1453362, (0 missing)
##   Surrogate splits:
##       ENSO < 14.75696   to the left,  agree=1.000, adj=1.000, (0 split)
##       Temp < 26.41545   to the right, agree=0.944, adj=0.286, (0 split)
##
## Node number 2: 83 observations,    complexity param=0.1524722
##   mean=8.927711, MSE=12.83815
##   left son=4 (27 obs) right son=5 (56 obs)
##   Primary splits:
##       ENSO < 0.6030456  to the right, improve=0.2315901, (0 missing)
##       Temp < 27.2664    to the left,  improve=0.1980226, (0 missing)
##       NAO  < 0.28875    to the right, improve=0.1183829, (0 missing)
##   Surrogate splits:
##       Temp < 26.74315   to the left,  agree=0.723, adj=0.148, (0 split)
##       NAO  < 2.405      to the right, agree=0.699, adj=0.074, (0 split)
##
## Node number 3: 7 observations
##   mean=16.14286, MSE=30.97959
##
## Node number 4: 27 observations,    complexity param=0.06557991
```

```
##   mean=6.444444, MSE=10.61728
##   left son=8 (19 obs) right son=9 (8 obs)
##   Primary splits:
##       Temp < 27.2356    to the left,  improve=0.37025700, (0 missing)
##       NAO  < 0.47625    to the right, improve=0.25555840, (0 missing)
##       ENSO < 0.954325   to the right, improve=0.05315615, (0 missing)
##   Surrogate splits:
##       NAO  < -0.49125   to the right, agree=0.815, adj=0.375, (0 split)
##       ENSO < 1.821212   to the left,  agree=0.741, adj=0.125, (0 split)
##
## Node number 5: 56 observations,    complexity param=0.06502277
##   mean=10.125, MSE=9.502232
##   left son=10 (44 obs) right son=11 (12 obs)
##   Primary splits:
##       Temp < 27.37925   to the left,  improve=0.19777050, (0 missing)
##       NAO  < 0.20625    to the right, improve=0.08069883, (0 missing)
##       ENSO < -0.7372568 to the right, improve=0.05048068, (0 missing)
##   Surrogate splits:
##       ENSO < 0.460807   to the left,  agree=0.804, adj=0.083, (0 split)
##
## Node number 8: 19 observations
##   mean=5.157895, MSE=3.606648
##
## Node number 9: 8 observations
##   mean=9.5, MSE=14
##
## Node number 10: 44 observations
##   mean=9.409091, MSE=6.78719
##
## Node number 11: 12 observations
##   mean=12.75, MSE=10.6875
```

```
rpart.plot(tree.pruned)
```




## 5. Comparing Training Errors and Test Errors

Now we not only want to compare the performance of original tree and pruned tree, we also want to understand the difference between in-sample errors and out-sample errors. First, from the below table we can see that the pruned model's RMSE is higher than original tree model's, meaning that the pruned tree may be too simple, thus resulting in a higher bias. Second, we also see that the in-sample errors are lower than out-sample errors. It is reasonable since the model doesn't see the data in the test set before, it will have higher errors in test set.

```
library(MLmetrics)
```

```
## Warning: package 'MLmetrics' was built under R version 3.5.2
```

```
##
## Attaching package: 'MLmetrics'
```

```
## The following object is masked from 'package:base':
##
##     Recall
```

```
train.prediction.1 = predict(tree, train.df)
test.prediction.1 = predict(tree, newdata=test.df)
train.prediction.2 = predict(tree.pruned, train.df)
```

```
test.prediction.2 = predict(tree.pruned, newdata=test.df)

training.rmse = c(RMSE(train.df$Storms, train.prediction.1), RMSE(train.df$Storms, train.prediction.2))
test.rmse = c(RMSE(test.df$Storms, test.prediction.1), RMSE(test.df$Storms, test.prediction.2))
model = c('tree', 'pruned tree')
table.1 = data.frame(model, training.rmse, test.rmse)
table.1
```

```
##         model training.rmse test.rmse
## 1        tree      2.985481  4.089886
## 2 pruned tree      3.026310  4.197393
```

# 6. Building Classification Tree

Next, we add a column to the data set which is 1 if the number of tropical storms is greater than the 80th percentile value of the data and 0 otherwise. Then we perform a 10-fold cross validation in which we build a classification tree to predict whether this new variable is a 1 or 0.

```
value = quantile(df$Storms, 0.8)
train.df$Status = ifelse(train.df$Storms >= value, 1, 0)
test.df$Status = ifelse(test.df$Storms >= value, 1, 0)

set.seed(121)
k = 10
train.df$Fold = sample(1:k, size=nrow(train.df), replace=T)
head(train.df)
```

```
##     Year Storms Landfalling    Temp        ENSO     NAO Status Fold
## 68  1967      8        3.00 27.0749 -0.68640003 -0.0175      0    4
## 82  1981     12        0.00 27.2491 -0.02304858  0.2475      0   10
## 42  1941      6        5.00 27.2343  1.05591670 -0.4575      0    6
## 57  1956      8        2.00 26.9621 -0.35223333  1.6500      0    8
## 111 2010     19       26.09 26.7800 27.64000000 28.1500      1    6
## 46  1945     11        8.00 27.3145 -0.15655000  0.4400      0    5
```

In each fold, we not only fit the model, but also record the training accuracy and validation accuracy. From the below table and accuracy plot, we can see that the training accuracy is much more stable than validation accuracy. Moreover, we find that in most of time the validation accuracy is lower than training accuracy. Also, the average validation accuracy in these 10 iterations is also lower than the average of training accuracy.

```
train.acc = c()
val.acc = c()

for (i in 1:k){
  train = train.df[-which(train.df$Fold==i),]
  val = train.df[which(train.df$Fold==i),]
  tree = rpart(Status ~ Temp+ENSO+NAO, data=train, method='class')
  train.acc = c(train.acc, Accuracy(predict(tree, type='class'), train$Status))
  val.acc = c(val.acc, Accuracy(predict(tree, newdata=val, type='class'), val$Status))
}
```

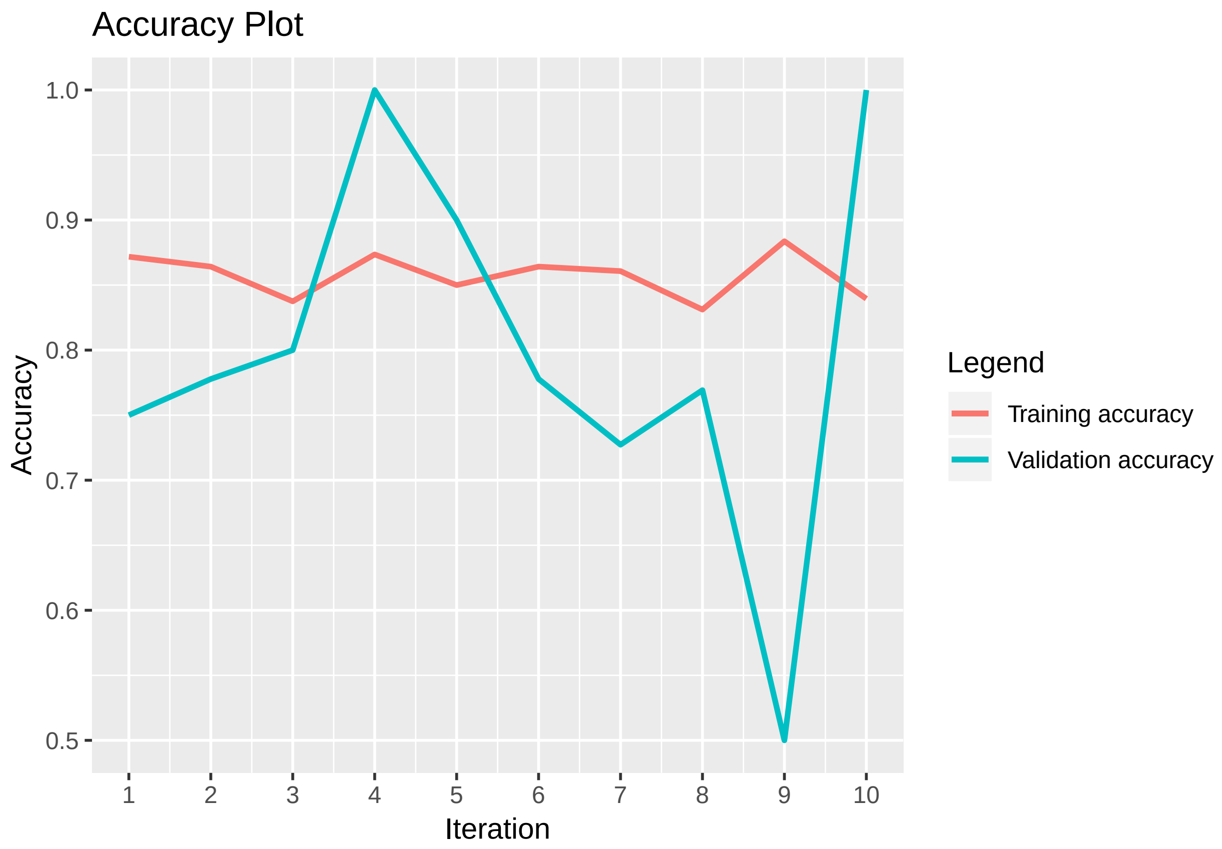
```r
itr = c(1:10)
acc.df = data.frame(itr, train.acc, val.acc)
acc.df
```

```
##    itr train.acc   val.acc
## 1    1 0.8717949 0.7500000
## 2    2 0.8641975 0.7777778
## 3    3 0.8375000 0.8000000
## 4    4 0.8735632 1.0000000
## 5    5 0.8500000 0.9000000
## 6    6 0.8641975 0.7777778
## 7    7 0.8607595 0.7272727
## 8    8 0.8311688 0.7692308
## 9    9 0.8837209 0.5000000
## 10  10 0.8395062 1.0000000
```

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

```r
ggplot(acc.df, aes(itr)) + geom_line(aes(y=train.acc, colour='Training accuracy'), size=1) + geom_line(a
```



```r
sum(train.acc)/length(train.acc)
```

```
## [1] 0.8576409
```

```r
sum(val.acc)/length(val.acc)
```

```
## [1] 0.8002059
```

# 7. Building Random Forests

So far, we have built regression trees and classification trees. However, this kind of trees has low bias but high variance. Thus, in this part we want to further build random forest models and compare their performance with single tree models. In R, we can use `randomForest` to build the random forest models.

In each fold iteration, we build two models, one is just a single regression tree and the other one is a random forest. From the result we see that the random forest's RMSE are usually lower than single-tree's RMSE. This show that random forests is reducing the variance by aggregating different tree models and thus have a better performance.

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.5.2
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

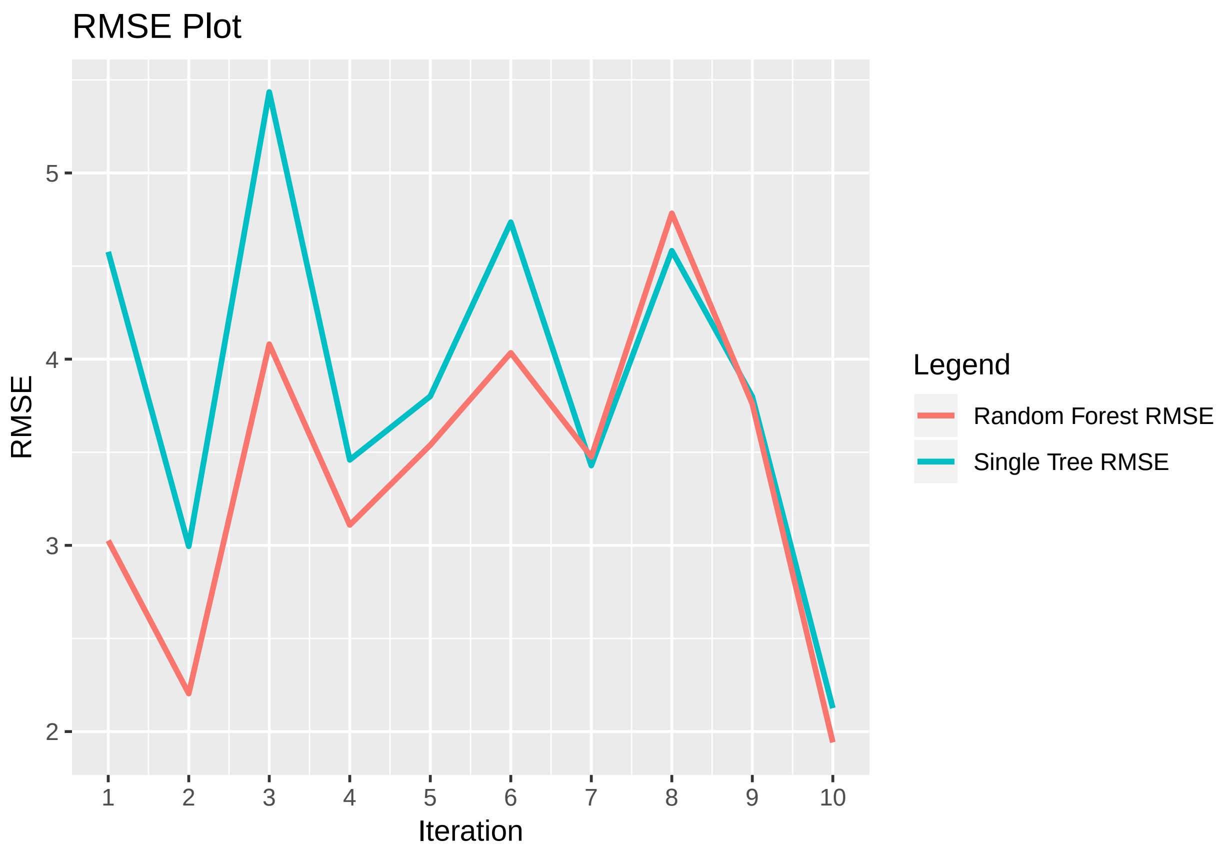
```r
tree.rmse = c()
forest.rmse = c()

for (i in 1:k){
  train = train.df[-which(train.df$Fold==i),]
  val = train.df[which(train.df$Fold==i),]
  tree = rpart(Storms~Temp+ENSO+NAO, data=train)
  forest = randomForest(Storms~Temp+ENSO+NAO, data=train, ntree=500)
  tree.rmse = c(tree.rmse, RMSE(val$Storms, predict(tree, newdata=val)))
  forest.rmse = c(forest.rmse, RMSE(val$Storms, predict(forest, newdata=val)))
}

acc.df.2 = data.frame(itr, tree.rmse, forest.rmse)
acc.df.2
```

```
##    itr tree.rmse forest.rmse
## 1    1  4.576339    3.025845
## 2    2  2.995247    2.203959
## 3    3  5.434580    4.080366
## 4    4  3.458995    3.109322
## 5    5  3.800695    3.538647
## 6    6  4.735517    4.034010
## 7    7  3.428357    3.475415
## 8    8  4.582325    4.783280
## 9    9  3.799113    3.759644
## 10  10  2.125914    1.941779
```

```
ggplot(acc.df.2, aes(itr)) + geom_line(aes(y=tree.rmse, colour='Single Tree RMSE'), size=1) + geom_line
```




```
sum(tree.rmse)/length(tree.rmse)
```

```
## [1] 3.893708
```

```
sum(forest.rmse)/length(forest.rmse)
```

```
## [1] 3.395227
```

Lastly, we just build the random forest model and then compute the test RMSE to make a final evaluation of this model.

```r
rforest = randomForest(Storms~Temp+ENSO+NAO, data=train.df)
# training RMSE
RMSE(train.df$Storms, predict(rforest))
```

```
## [1] 3.625005
```

```r
# test RMSE
RMSE(test.df$Storms, predict(rforest, newdata=test.df))
```

```
## [1] 3.90974
```