# Advanced Trees with Wages Data

## Contents



Trees are flexible and easy to understand but tend to overfit and do not have the same level of predictive accuracy as other prediction models. In the following sections, we will examine a number of ways to improve tree models including tuning tree hyperparameters, and ensemble models such as bagging, forests, and boosting.

## Data Description

Wages dataset is a simulated dataset based on a real dataset published in Data Analysis using Regression and Multilevel/Hierarchical Models by Andrew Gelman and Jennifer Hill

Data consists of characteristics of a set of employees and their annual earning. Variables included are:

- earn: Annual earning in dollars
- height: Height in inches
- gender: Gender (male, female)

- race: african-american, asian, hispanic, white
- ed: Years of education
- age: Age in years

# Read Data

Set your working directory to the location where the data is saved by adapting the code below.

```
setwd('c:/my_classes/best_course_ever/') # on Windows
setwd('/my_classes/best_course_ever/') # on Mac
```

```
data = read.csv('wages.csv', stringsAsFactors = T)
data = data[data$earn>0,]
```

# Split Data

```
set.seed(617)
split = sample(1:nrow(data),size = nrow(data)*0.8)
train = data[split,]
test = data[-split,]
```

In the following sections, we will build a series of models, in some cases tune them and then evaluate them on the test data.

# Tree Models

In this section, we will examine a tree with default cp of 0.01, a maximal tree, and tree tuned with 10-fold cross-validation.

## Default Tree

The default value of cp is 0.01

```
library(rpart); library(rpart.plot)
tree = rpart(earn~.,data=train)
pred = predict(tree,newdata=test)
rmse_tree = sqrt(mean((pred-test$earn)^2)); rmse_tree
```

```
## [1] 53456.15
```

## Maximal Tree

This is the largest possible tree.

```
maximalTree = rpart(earn~.,data=train,control=rpart.control(cp=0))
pred = predict(maximalTree,newdata=test)
rmse_maximalTree = sqrt(mean((pred-test$earn)^2)); rmse_maximalTree
```

```
## [1] 57082.55
```

## Tree with Tuning

Tune the complexity of a tree using 5-fold cross-validation. Here, we will examine cross-validation error for 100 different values of cp.

```r
library(caret)
trControl = trainControl(method='cv',number = 5)
tuneGrid = expand.grid(.cp = seq(from = 0.001,to = 0.1,by = 0.001))
set.seed(617)
cvModel = train(earn~.,
                data=train,
                method="rpart",
                trControl = trControl,
                tuneGrid = tuneGrid)
```

```r
cvModel$results
```

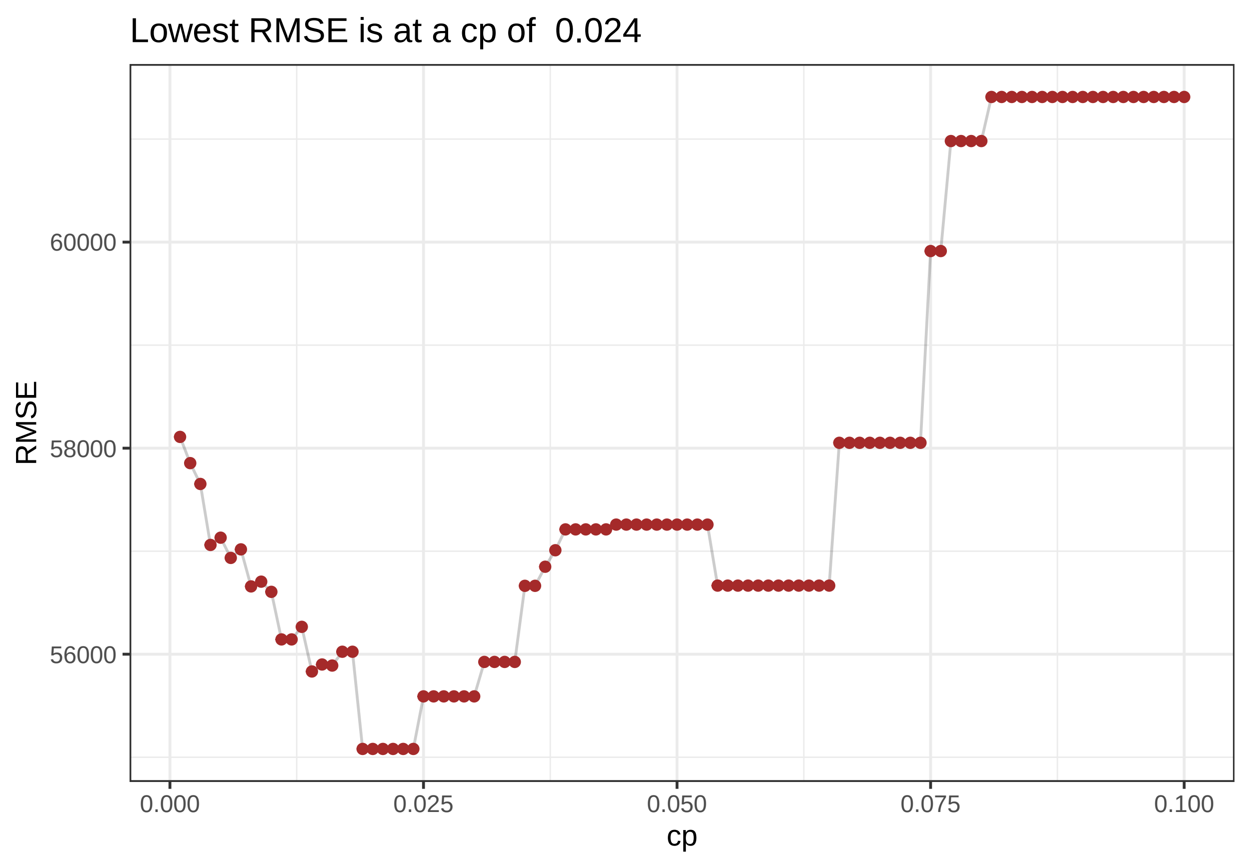
```
##       cp     RMSE  Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1  0.001 58109.24 0.2205979 38289.82 4096.196 0.07289116 1409.1612
## 2  0.002 57854.68 0.2199231 38122.92 4255.585 0.08119947 1296.6183
## 3  0.003 57652.57 0.2201436 37787.52 4114.090 0.08668704  950.9004
## 4  0.004 57060.70 0.2261675 37474.47 4554.308 0.09144301 1385.1958
## 5  0.005 57131.09 0.2219554 37442.13 4730.652 0.08875365 1543.5813
## 6  0.006 56934.97 0.2236642 37421.03 4850.358 0.09632551 1327.4715
## 7  0.007 57017.74 0.2202879 37609.53 4789.538 0.09650538 1192.7217
## 8  0.008 56658.25 0.2274475 37378.90 4468.495 0.09804062  878.6218
## 9  0.009 56704.22 0.2259959 37435.16 4548.977 0.09891671  999.5121
## 10 0.010 56605.38 0.2257526 37498.57 4477.987 0.09881983  980.0040
## 11 0.011 56143.64 0.2352185 37513.93 3601.494 0.09873190  824.1015
## 12 0.012 56143.64 0.2352185 37513.93 3601.494 0.09873190  824.1015
## 13 0.013 56265.77 0.2323064 37818.42 3685.956 0.10019309 1178.7746
## 14 0.014 55832.55 0.2399879 37540.70 2844.735 0.09855232  627.1255
## 15 0.015 55900.22 0.2412936 37536.26 3021.938 0.10532164  623.4104
## 16 0.016 55889.92 0.2403748 37443.27 3023.800 0.10654123  476.9280
## 17 0.017 56023.99 0.2372423 37664.96 3336.998 0.11906822 1242.1867
## 18 0.018 56023.99 0.2372423 37664.96 3336.998 0.11906822 1242.1867
## 19 0.019 55080.56 0.2609847 37331.31 4357.008 0.14294082 1571.5555
## 20 0.020 55080.56 0.2609847 37331.31 4357.008 0.14294082 1571.5555
## 21 0.021 55080.56 0.2609847 37331.31 4357.008 0.14294082 1571.5555
## 22 0.022 55080.56 0.2609847 37331.31 4357.008 0.14294082 1571.5555
## 23 0.023 55080.56 0.2609847 37331.31 4357.008 0.14294082 1571.5555
## 24 0.024 55080.56 0.2609847 37331.31 4357.008 0.14294082 1571.5555
## 25 0.025 55590.83 0.2550573 37817.45 5127.188 0.15048022 2104.9636
## 26 0.026 55590.83 0.2550573 37817.45 5127.188 0.15048022 2104.9636
## 27 0.027 55590.83 0.2550573 37817.45 5127.188 0.15048022 2104.9636
## 28 0.028 55590.83 0.2550573 37817.45 5127.188 0.15048022 2104.9636
## 29 0.029 55590.83 0.2550573 37817.45 5127.188 0.15048022 2104.9636
## 30 0.030 55590.83 0.2550573 37817.45 5127.188 0.15048022 2104.9636
## 31 0.031 55925.05 0.2481645 38026.19 5699.440 0.16018328 2449.1204
## 32 0.032 55925.05 0.2481645 38026.19 5699.440 0.16018328 2449.1204
## 33 0.033 55925.05 0.2481645 38026.19 5699.440 0.16018328 2449.1204
## 34 0.034 55925.05 0.2481645 38026.19 5699.440 0.16018328 2449.1204
## 35 0.035 56663.93 0.2285600 38726.06 4903.507 0.15222101 2251.3395
## 36 0.036 56663.93 0.2285600 38726.06 4903.507 0.15222101 2251.3395
## 37 0.037 56849.17 0.2246281 38848.63 4937.458 0.14980035 2156.5893
## 38 0.038 57009.20 0.2223501 39155.68 4803.357 0.14711642 1713.7947
## 39 0.039 57211.47 0.2182940 39317.12 4873.200 0.14501027 1578.7061
## 40 0.040 57211.47 0.2182940 39317.12 4873.200 0.14501027 1578.7061
## 41 0.041 57211.47 0.2182940 39317.12 4873.200 0.14501027 1578.7061
```

```
## 42  0.042 57211.47 0.2182940 39317.12 4873.200 0.14501027 1578.7061
## 43  0.043 57211.47 0.2182940 39317.12 4873.200 0.14501027 1578.7061
## 44  0.044 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 45  0.045 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 46  0.046 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 47  0.047 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 48  0.048 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 49  0.049 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 50  0.050 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 51  0.051 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 52  0.052 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 53  0.053 57258.11 0.2208563 39407.07 4836.527 0.14816870 1460.3084
## 54  0.054 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 55  0.055 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 56  0.056 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 57  0.057 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 58  0.058 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 59  0.059 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 60  0.060 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 61  0.061 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 62  0.062 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 63  0.063 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 64  0.064 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 65  0.065 56665.90 0.2307355 39257.96 4936.649 0.13656457 1366.5363
## 66  0.066 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 67  0.067 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 68  0.068 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 69  0.069 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 70  0.070 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 71  0.071 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 72  0.072 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 73  0.073 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 74  0.074 58052.07 0.1887969 39568.77 5076.402 0.11307757 1167.4040
## 75  0.075 59913.26 0.1468932 40537.95 4585.438 0.09474611 2372.1906
## 76  0.076 59913.26 0.1468932 40537.95 4585.438 0.09474611 2372.1906
## 77  0.077 60981.05 0.1152880 41022.86 4915.524 0.06462534 2092.4692
## 78  0.078 60981.05 0.1152880 41022.86 4915.524 0.06462534 2092.4692
## 79  0.079 60981.05 0.1152880 41022.86 4915.524 0.06462534 2092.4692
## 80  0.080 60981.05 0.1152880 41022.86 4915.524 0.06462534 2092.4692
## 81  0.081 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 82  0.082 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 83  0.083 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 84  0.084 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 85  0.085 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 86  0.086 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 87  0.087 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 88  0.088 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 89  0.089 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 90  0.090 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 91  0.091 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 92  0.092 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 93  0.093 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 94  0.094 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 95  0.095 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
```

```
## 96  0.096 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 97  0.097 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 98  0.098 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 99  0.099 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
## 100 0.100 61408.86 0.1011505 41432.09 4299.847 0.07159193 1957.6928
```

Results from Tuning

```
library(ggplot2)
ggplot(data=cvModel$results, aes(x=cp, y=RMSE))+
  geom_line(size=0.5,alpha=0.2)+
  geom_point(color='brown')+
  theme_bw()+
  ggtitle(label=paste('Lowest RMSE is at a cp of ',cvModel$bestTune$cp))
```




Evaluate Tuned model on Test sample

```
cvTree = rpart(earn~.,data=train,cp = cvModel$bestTune$cp)
pred = predict(cvTree,newdata=test)
rmse_cvTree = sqrt(mean((pred-test$earn)^2)); rmse_cvTree
```

```
## [1] 54198.35
```

## Ensemble Models

- Bagging
- Forest
- Boosting

## Bag Models

Bootstrap Aggregation models generate a large number of bootstrapped samples. A tree is fit to each bootstrapped sample. Predictions are generating as an average of all models (for numerical outcome variables) or the majority group (for categorical outcome variables).

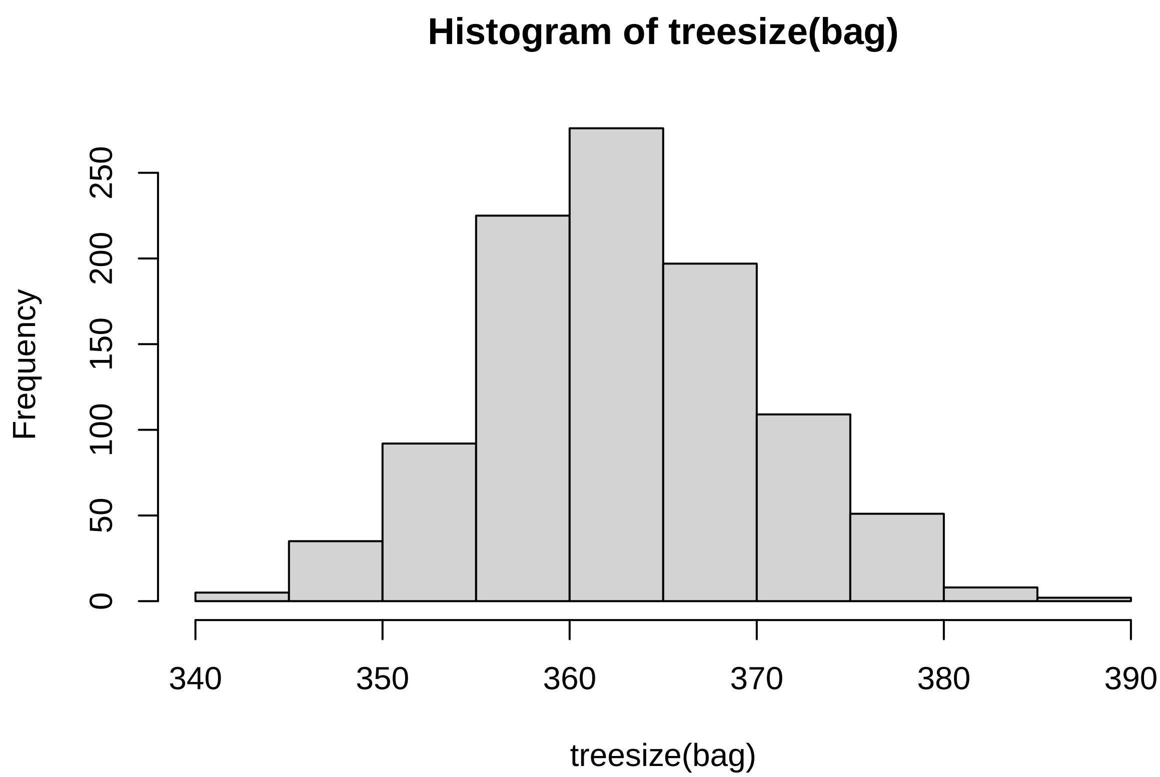
These models can be implemented using many packages including ipred, randomForest,adabag, bagEarth, treeBag, bagFDA. In this illustration, we are using a randomForest model by setting mtry to be the number of predictors, i.e., 5.

```
library(randomForest)
set.seed(617)
bag = randomForest(earn~.,data=train,mtry = ncol(train)-1,ntree=1000)
pred = predict(bag,newdata=test)
rmse_bag = sqrt(mean((pred-test$earn)^2)); rmse_bag
```

```
## [1] 53486.83
```

The various trees that were fit vary in their size

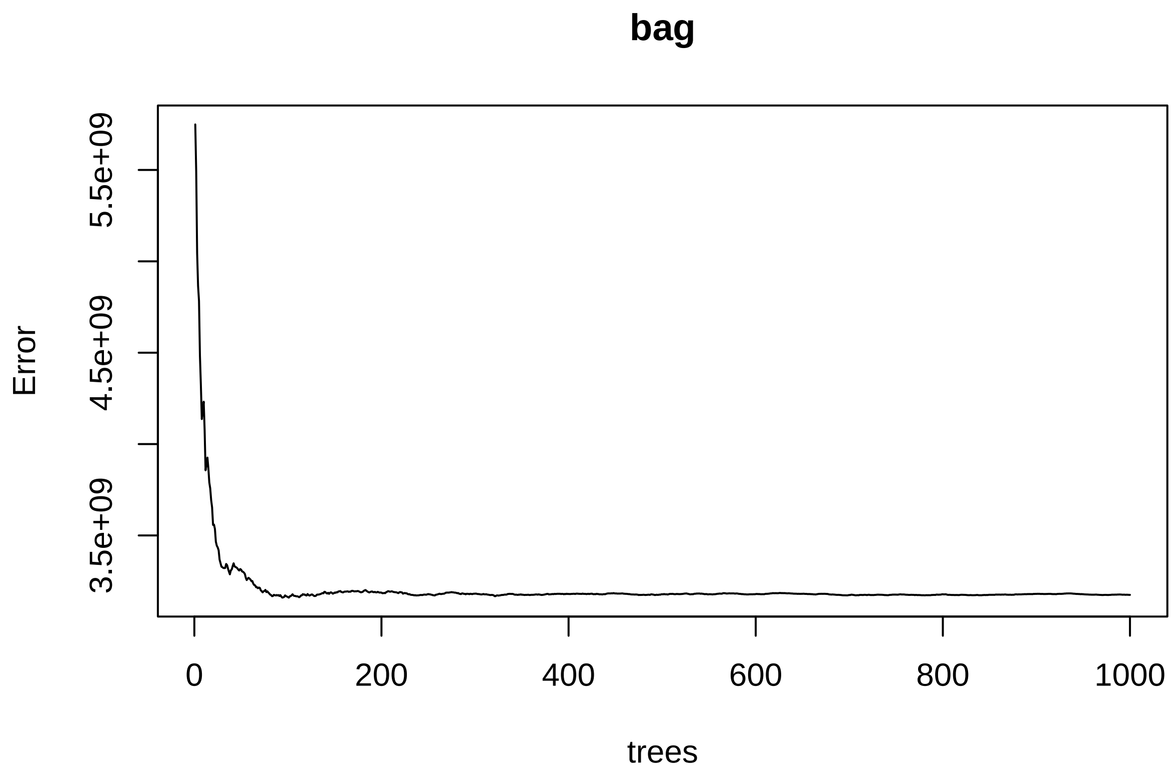
```
hist(treesize(bag))
```




```
# getTree(bag,k=100)   # One can examine a given tree
```

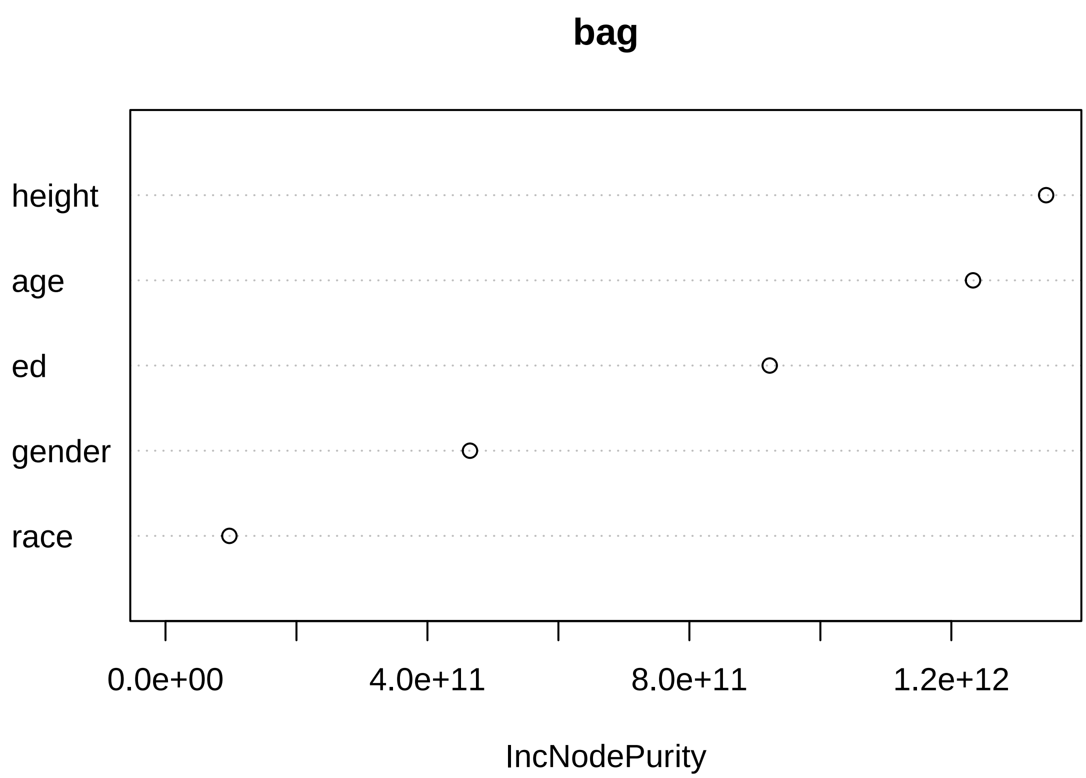
Generally speaking, error for a bag model decreases with an increase in the trees.

```
plot(bag)
```




Relative importance of predictors

```
varImpPlot(bag)
```




```
importance(bag)
```

```
##        IncNodePurity
## height  1.344749e+12
## gender  4.649394e+11
## race    9.759492e+10
## ed      9.227044e+11
## age     1.233138e+12
```

## Random Forest Models

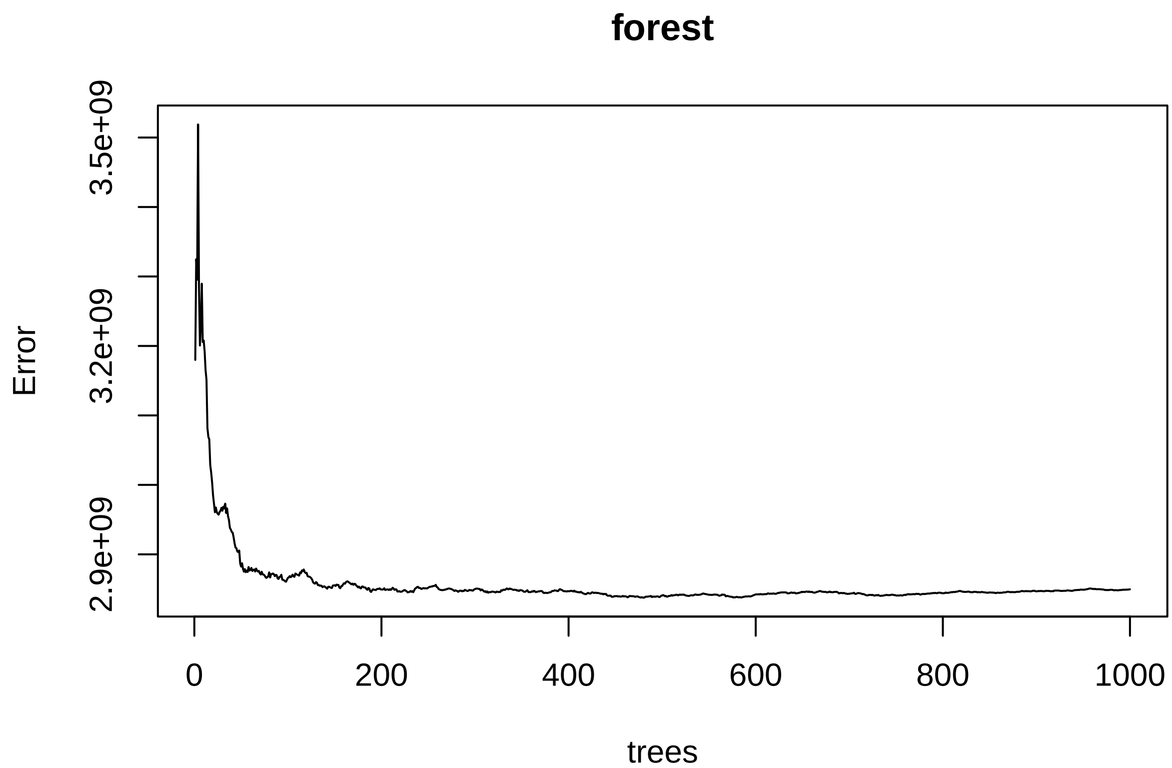
Unlike bag models which consider all predictors in constructing each tree, randomForest models consider a subset of predictors (default is p/3 for numerical outcomes or sqrt(p) for categorical outcomes) for constructing each tree. In `library(randomForest)`, mtry controls number of variables considered for each tree.

```
library(randomForest)
set.seed(617)
forest = randomForest(earn~.,data=train,ntree = 1000)
pred = predict(forest,newdata=test)
rmse_forest = sqrt(mean((pred-test$earn)^2)); rmse_forest
```

```
## [1] 50062.28
```

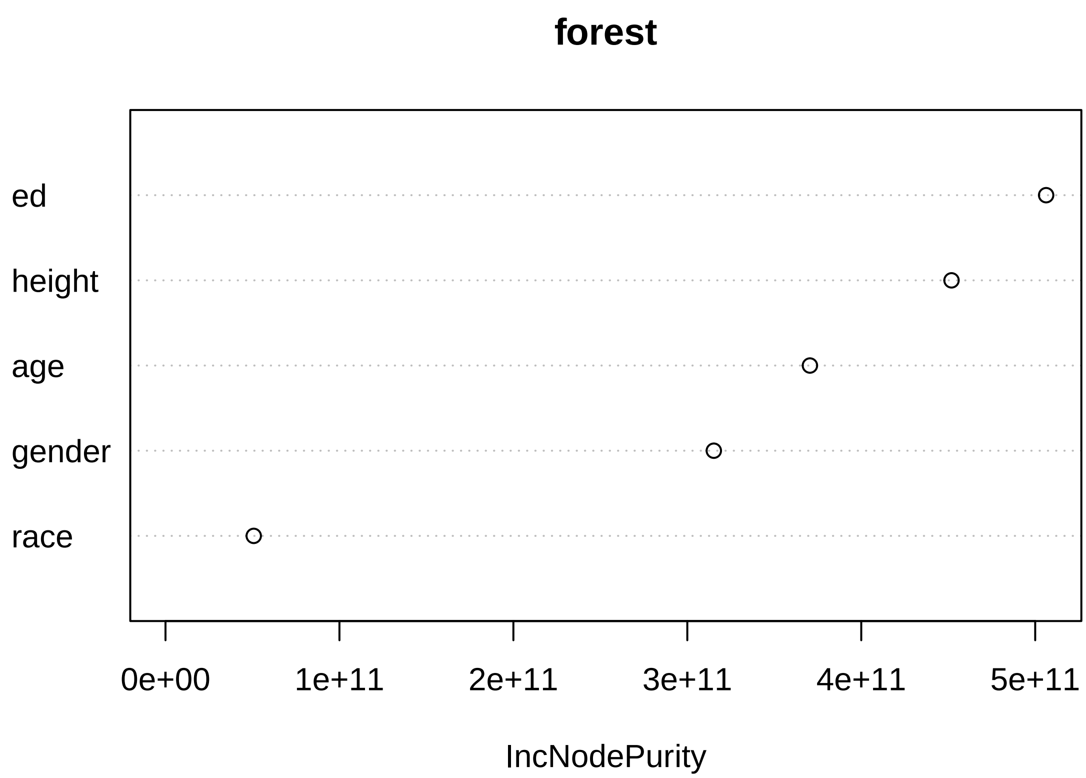
As in the case for bags, more trees reduces error to a certain point, after which the graph asymptotes.

```
plot(forest)
```




Relative importance of predictors

```
varImpPlot(forest)
```




### Tuned Random Forest

The mtry parameter of a randomForest model can be tuned to improve model predictions. Here, we will use 5-fold cross validation to examine four values of mtry.

```
trControl=trainControl(method="cv",number=5)
tuneGrid = expand.grid(mtry=1:4)
set.seed(617)
cvModel = train(earn~.,data=train,
                 method="rf",ntree=1000,trControl=trControl,tuneGrid=tuneGrid )
cvModel
```

```
## Random Forest
##
## 1094 samples
##    5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 875, 875, 875, 876, 875
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   1     55791.64  0.2891217  36982.68
##   2     53265.49  0.3015901  35135.91
##   3     54021.84  0.2856715  35466.80
##   4     55117.06  0.2680037  36034.66
```

```
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

Now, let us use the best mtry of 2.

```
cvForest = randomForest(earn~.,data=train,ntree = 1000,mtry=cvModel$bestTune$mtry)
pred = predict(cvForest,newdata=test)
rmse_cv_forest = sqrt(mean((pred-test$earn)^2)); rmse_cv_forest
```

```
## [1] 52363.92
```

### Forest with Ranger

ranger is popular R library for running random forest models.

```
library(ranger)
```

```
##
## Attaching package: 'ranger'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
forest_ranger = ranger(earn~.,data=train,num.trees = 1000)
pred = predict(forest_ranger, data =test,num.trees = 1000)
rmse_forest_ranger = sqrt(mean((pred$predictions-test$earn)^2)); rmse_forest_ranger
```

```
## [1] 52381.18
```

### Tuned Forest Ranger

Tuning ranger model for different values of mtry, splitrule and min.node.size

```
trControl=trainControl(method="cv",number=5)
tuneGrid = expand.grid(mtry=1:4,
                       splitrule = c('variance','extratrees','maxstat'),
                       min.node.size = c(2,5,10,15,20,25))
set.seed(617)
cvModel = train(earn~.,
                data=train,
                method="ranger",
                num.trees=1000,
                trControl=trControl,
                tuneGrid=tuneGrid )
cv_forest_ranger = ranger(earn~.,
                          data=train,
                          num.trees = 1000,
                          mtry=cvModel$bestTune$mtry,
                          min.node.size = cvModel$bestTune$min.node.size,
                          splitrule = cvModel$bestTune$splitrule)
pred = predict(cv_forest_ranger, data =test, num.trees = 1000)
rmse_cv_forest_ranger = sqrt(mean((pred$predictions-test$earn)^2)); rmse_cv_forest_ranger
```

```
## [1] 51741.44
```

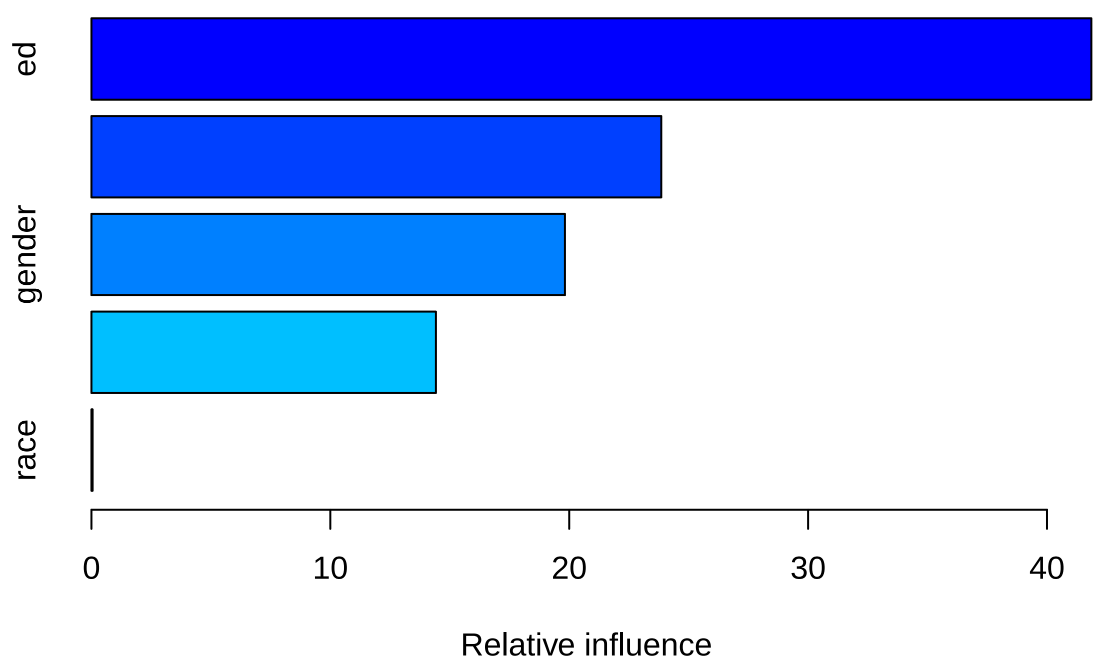
## Boosting Models

Like bag and forest models, boosting models are ensemble models that derive predictions from a number of trees. The key difference is that in boosting, trees are grown sequentially, each tree is grown using information from previously grown trees. Thus, boosting can be seen as a slow learning evolutionary model. Since we are predicting a numerical variable, earn, the distribution is set to 'gaussian'. Had the goal been to predict a binary outcome, we would have set distribution to 'bernoulli'.

```
library(gbm)
set.seed(617)
boost = gbm(earn~.,
            data=train,
            distribution="gaussian",
            n.trees = 500,
            interaction.depth = 2,
            shrinkage = 0.01)
```

Boosting models offer a way to examine the relative influence of variables.

```
summary(boost)
```




```
##            var     rel.inf
## ed          ed 41.85723381
## age        age 23.85348758
## gender gender 19.82020900
## height height 14.41961849
## race      race  0.04945111
```

Let us examine RMSE for the train sample.

```r
pred = predict(boost,n.trees = 500)
rmse_boost_train = sqrt(mean((pred-train$earn)^2)); rmse_boost_train
```

```
## [1] 50425.2
```

Now, examine RMSE for test sample.

```r
pred = predict(boost,newdata=test,n.trees = 500)
rmse_boost = sqrt(mean((pred-test$earn)^2)); rmse_boost
```

```
## [1] 50876.59
```

### Boosting with cross-validation

Boosting models are notorious for overfitting training data. A very simple way to see this borne out is to see the effect of increasing number of trees on train and test rmse. Specifically, in the model above, try running the models with the following number of trees: 200, 1e3, 1e4, 1e6. To avoid the folly of overfitting, it is best to tune the model using cross-validation. In the code that follows, we will tune a gradient boosting model using interaction depth, shrinkage and minobsinnode.

The code below uses a garbage collector to hide from us the long list of models tested.

```r
library(caret)
set.seed(617)
trControl = trainControl(method="cv",number=5)
tuneGrid = expand.grid(n.trees = 500,
                       interaction.depth = c(1,2,3),
                       shrinkage = (1:100)*0.001,
                       n.minobsinnode=c(5,10,15))
garbage = capture.output(cvModel <- train(earn~.,
                                          data=train,
                                          method="gbm",
                                          trControl=trControl,
                                          tuneGrid=tuneGrid))
set.seed(617)
cvBoost = gbm(earn~.,
              data=train,
              distribution="gaussian",
              n.trees=cvModel$bestTune$n.trees,
              interaction.depth=cvModel$bestTune$interaction.depth,
              shrinkage=cvModel$bestTune$shrinkage,
              n.minobsinnode = cvModel$bestTune$n.minobsinnode)
pred = predict(cvBoost,test,n.trees=500)
rmse_cv_boost = sqrt(mean((pred-test$earn)^2)); rmse_cv_boost
```

```
## [1] 51234.84
```

### Boosting with xgboost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.

The algorithm is also a bit picky about the format of variables used. All factor class variables need to be dummy coded and fed into the model as a matrix. To do this, we will dummy code using library(vtreat)

```r
library(vtreat)
```

```
## Loading required package: wrapr
```

```r
trt = designTreatmentsZ(dframe = train,
                        varlist = names(train)[2:6])
```

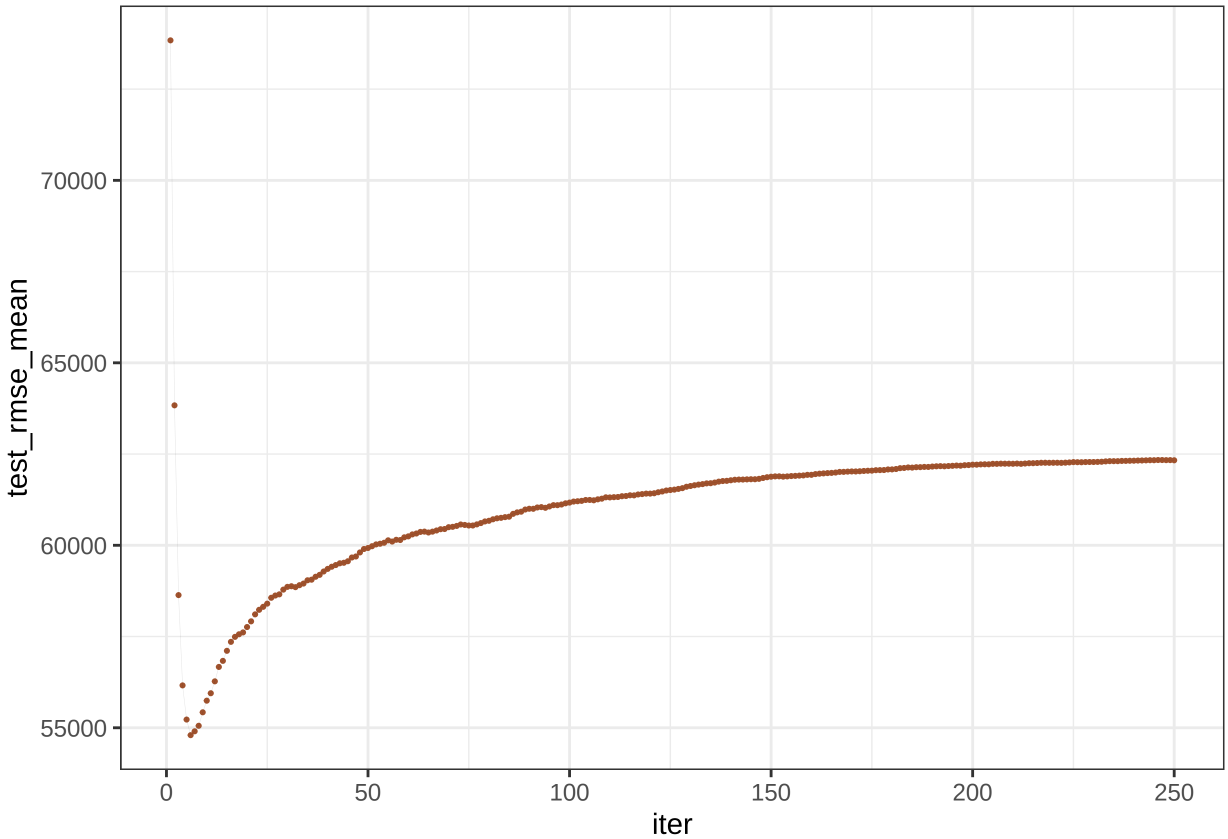
```
## [1] "vtreat 1.6.3 inspecting inputs Wed Sep 01 17:09:53 2021"
## [1] "designing treatments Wed Sep 01 17:09:53 2021"
## [1] " have initial level statistics Wed Sep 01 17:09:53 2021"
## [1] " scoring treatments Wed Sep 01 17:09:53 2021"
## [1] "have treatment plan Wed Sep 01 17:09:53 2021"
```

```r
newvars = trt$scoreFrame[trt$scoreFrame$code%in% c('clean','lev'),'varName']

train_input = prepare(treatmentplan = trt,
                      dframe = train,
                      varRestriction = newvars)
test_input = prepare(treatmentplan = trt,
                     dframe = test,
                     varRestriction = newvars)
head(train_input)
```

```
##   height ed age gender_lev_x_female gender_lev_x_male
## 1  65.98 14  47                   1                 0
## 2  64.07 16  64                   1                 0
## 3  59.61 16  92                   1                 0
## 4  63.28 12  42                   1                 0
## 5  71.34 11  22                   0                 1
## 6  67.70 14  29                   1                 0
##   race_lev_x_african_minus_american race_lev_x_asian race_lev_x_hispanic
## 1                                 0                0                   0
## 2                                 0                0                   0
## 3                                 0                1                   0
## 4                                 0                0                   0
## 5                                 1                0                   0
## 6                                 0                0                   0
##   race_lev_x_white
## 1                1
## 2                1
## 3                0
## 4                1
## 5                0
## 6                1
```

Like all boosting models, xgboost can overfit the train data. To identify the optimal nrounds, we use a cross-validation function within xgboost.

```r
library(xgboost); library(caret)
set.seed(617)
tune_nrounds = xgb.cv(data=as.matrix(train_input),
                      label = train$earn,
                      nrounds=250,
                      nfold = 5,
                      verbose = 0)

ggplot(data=tune_nrounds$evaluation_log, aes(x=iter, y=test_rmse_mean))+
  geom_point(size=0.4, color='sienna')+
  geom_line(size=0.1, alpha=0.1)+
  theme_bw()
```




As is obvious from the above chart, the optimal nrounds is a rather small number

```
which.min(tune_nrounds$evaluation_log$test_rmse_mean)
```

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

Next, we use xgboost to fit the train data with nrounds = 6 and apply the model to the test data.

```
xgboost2= xgboost(data=as.matrix(train_input),
                  label = train$earn,
                  nrounds=6,
                  verbose = 0)
pred = predict(xgboost2,
               newdata=as.matrix(test_input))
rmse_xgboost = sqrt(mean((pred - test$earn)^2)); rmse_xgboost
```

```
## [1] 51597.19
```

Now, it is possible to tune an xgboost model using other parameters such as eta, max_depth, and gamma using the `train` function from library(caret)

## Results

Here are the results from the different models that were run

```
data.frame(models = c('Tree', 'Maximal Tree', 'Tuned Tree', 'Bag','Forest','Tuned Forest','Ranger','Tun
           RMSE = c(rmse_tree, rmse_maximalTree, rmse_cvTree, rmse_bag, rmse_forest, rmse_cv_forest, rm
rmse_cv_forest_ranger, rmse_boost, rmse_cv_boost, rmse_xgboost))
```

```
##          models     RMSE
```

```
## 1           Tree 53456.15
## 2  Maximal Tree 57082.55
## 3    Tuned Tree 54198.35
## 4            Bag 53486.83
## 5         Forest 50062.28
## 6  Tuned Forest 52363.92
## 7         Ranger 52381.18
## 8  Tuned Ranger 51741.44
## 9          Boost 50876.59
## 10  Tuned Boost 51234.84
## 11      XGBoost 51597.19
```

## And the Winner is ???