Classification Trees in R

May 15, 2016

Contents

We'll be working with the same Twitter dataset again this week:

```
library(dplyr)
library(ggplot2)
library(scales)
library(caret)

twitter = read.delim('bot_or_not.tsv',
                     sep = '\t',
                     header = TRUE)
```

As usual, divide the data into test and train.

```
# tell R which variables are categorical (factors)
twitter$bot = factor(twitter$bot)
twitter$default_profile = factor(twitter$default_profile)
twitter$default_profile_image = factor(twitter$default_profile_image)
twitter$geo_enabled = factor(twitter$geo_enabled)

summary(twitter)
```

```
##  bot      statuses_count   default_profile default_profile_image
##  0:2672   Min.   :     0   0:2256          0:3077
##  1: 504   1st Qu.:   188   1: 920          1:  99
##           Median :   723
##           Mean   :  3277
##           3rd Qu.:  2646
##           Max.   :137264
##  friends_count     followers_count     favourites_count geo_enabled
##  Min.   :    11   Min.   :      0.0   Min.   :     0   0:1773
##  1st Qu.:   300   1st Qu.:     95.0   1st Qu.:    14   1:1403
##  Median :   615   Median :    288.0   Median :   122
##  Mean   :  2358   Mean   :   3709.3   Mean   :  1100
##  3rd Qu.:  1229   3rd Qu.:    830.5   3rd Qu.:   593
```

```
##  Max.   :1175187   Max.   :1396699.0   Max.   :176219
##   listed_count     account_age_hours   diversity
##  Min.   :   0.00   Min.   : 2072      Min.   :0.0050
##  1st Qu.:   4.00   1st Qu.:30285      1st Qu.:0.6254
##  Median :  16.00   Median :47484      Median :0.6963
##  Mean   :  84.77   Mean   :43664      Mean   :0.6791
##  3rd Qu.:  51.00   3rd Qu.:56718      3rd Qu.:0.7626
##  Max.   :9491.00   Max.   :78841      Max.   :1.0000
##  mean_mins_between_tweets mean_tweet_length mean_retweets
##  Min.   :    -15.7        Min.   :  8.50    Min.   :   1.000
##  1st Qu.:   1152.8        1st Qu.: 80.79    1st Qu.:   1.167
##  Median :   3851.7        Median : 91.74    Median :   1.636
##  Mean   :  14715.4        Mean   : 91.41    Mean   :   3.873
##  3rd Qu.:  10823.8        3rd Qu.:103.28    3rd Qu.:   2.424
##  Max.   :1139015.0        Max.   :287.88    Max.   :1961.300
##    reply_rate
##  Min.   :0.0000
##  1st Qu.:0.1232
##  Median :0.3137
##  Mean   :0.3411
##  3rd Qu.:0.5279
##  Max.   :1.0000
```

```r
set.seed(243)

# select the training observations
in_train = createDataPartition(y = twitter$bot,
                               p = 0.75, # 75% in train, 25% in test
                               list = FALSE)

training_set = twitter[in_train, ]
testing_set = twitter[-in_train, ]
```

Grow one tree

caret has lots of different tree models, so check 'em out. We can make a simple tree model using the `rpart` method.

```r
tree_model = train(bot ~.,
                   method = 'rpart',
                   data = training_set)

print(tree_model)
```

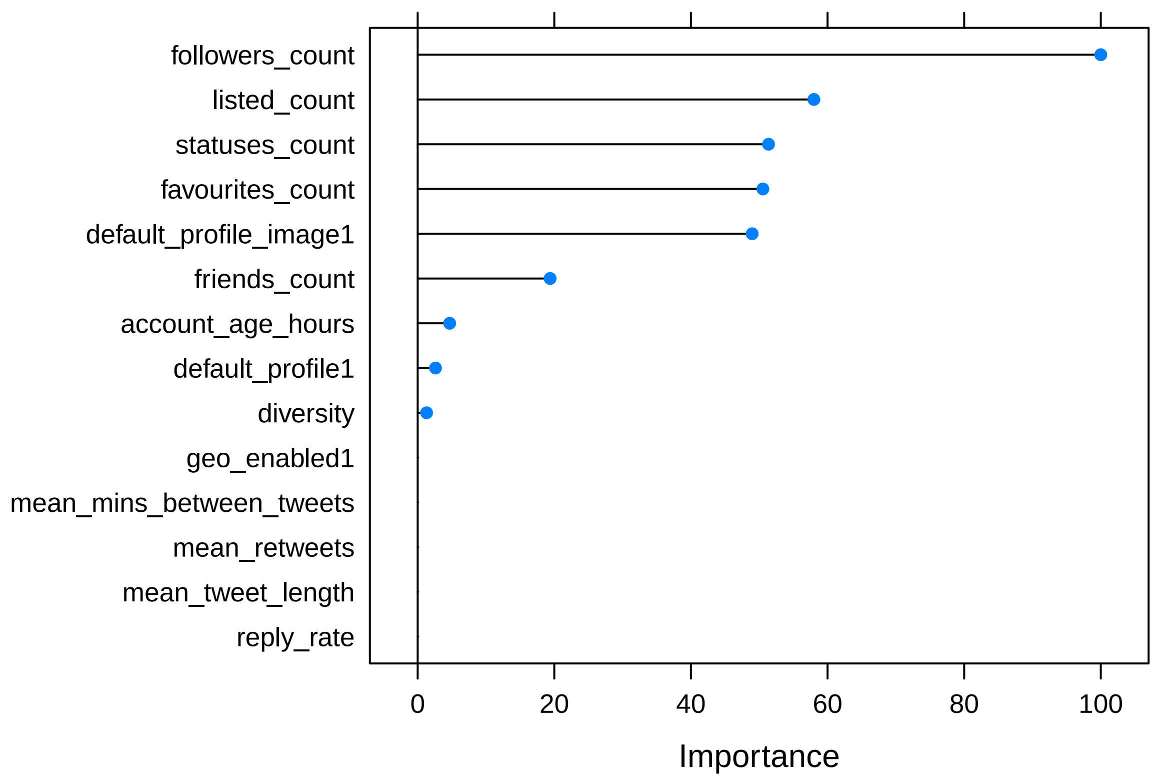
```
## CART
##
## 2382 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
```

```
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2382, 2382, 2382, 2382, 2382, 2382, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01851852  0.9060279  0.5984621
##   0.03306878  0.9006168  0.5525340
##   0.36507937  0.8652871  0.2341893
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.01851852.
```

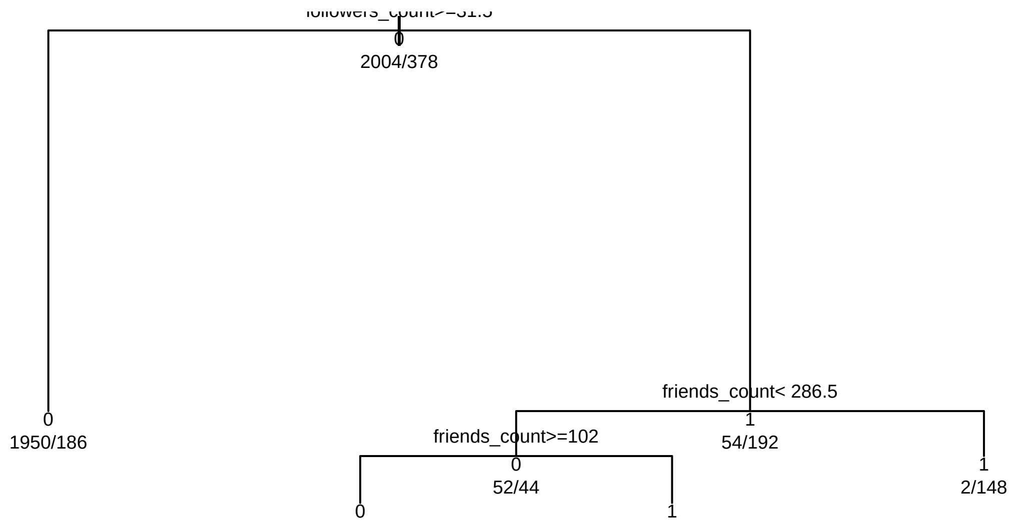
```r
print(tree_model$finalModel)
```

```
## n= 2382
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 2382 378 0 (0.84130982 0.15869018)
##    2) followers_count>=31.5 2136 186 0 (0.91292135 0.08707865) *
##    3) followers_count< 31.5 246  54 1 (0.21951220 0.78048780)
##      6) friends_count< 286.5 96  44 0 (0.54166667 0.45833333)
##       12) friends_count>=102 37   6 0 (0.83783784 0.16216216) *
##       13) friends_count< 102 59  21 1 (0.35593220 0.64406780) *
##      7) friends_count>=286.5 150   2 1 (0.01333333 0.98666667) *
```

```r
plot(varImp(tree_model))
```



```r
# plot the tree!
plot(tree_model$finalModel)
text(tree_model$finalModel, use.n = TRUE, all = TRUE, cex = 0.60)
```




```r
# test the predictions
tree_predictions = predict(tree_model, newdata = testing_set)
confusionMatrix(tree_predictions, testing_set$bot)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 656  72
##          1  12  54
##
##                Accuracy : 0.8942
##                  95% CI : (0.8707, 0.9147)
##     No Information Rate : 0.8413
##     P-Value [Acc > NIR] : 1.162e-05
##
##                   Kappa : 0.5089
##  Mcnemar's Test P-Value : 1.215e-10
##
##             Sensitivity : 0.9820
##             Specificity : 0.4286
##          Pos Pred Value : 0.9011
##          Neg Pred Value : 0.8182
##              Prevalence : 0.8413
##          Detection Rate : 0.8262
##    Detection Prevalence : 0.9169
##       Balanced Accuracy : 0.7053
##
##        'Positive' Class : 0
##
```

By default, the train function will try three values of the complexity parameter, but we can tell it to try more using the `tuneLength` argument.

```r
tree_model = train(bot ~.,
                   method = 'rpart',
                   data = training_set,
                   tuneLength = 10)
print(tree_model)
```

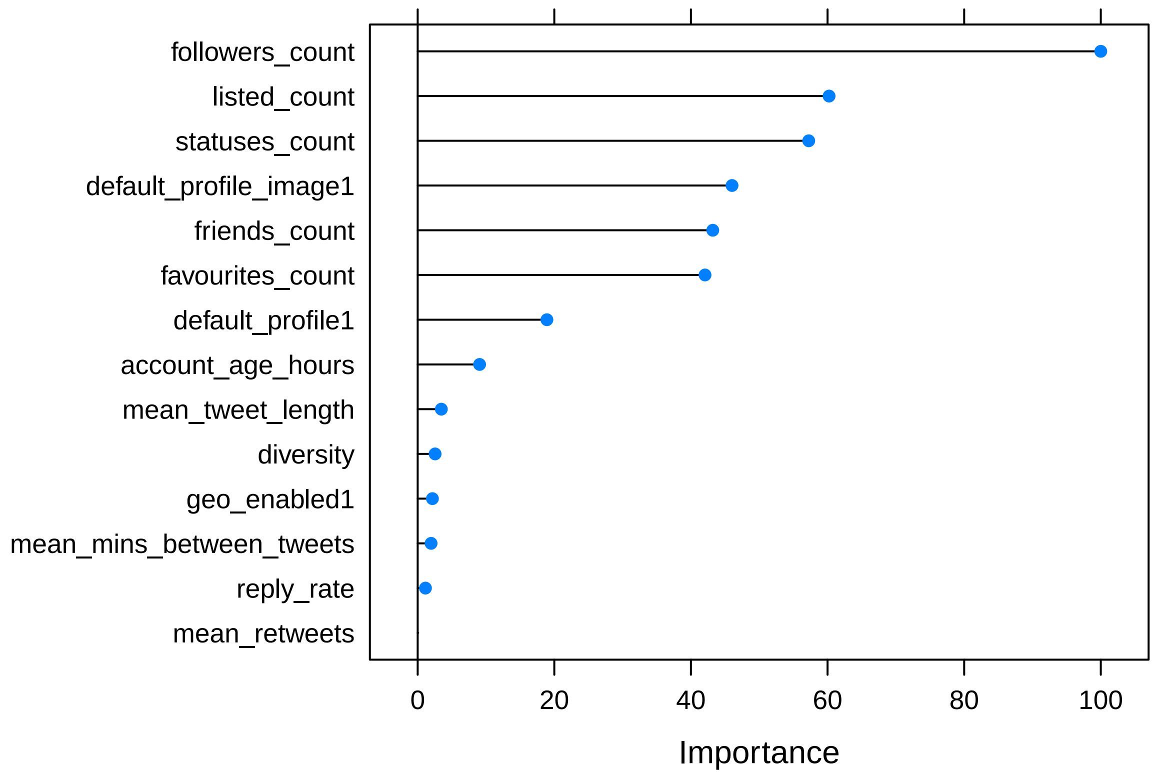
```
## CART
##
## 2382 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2382, 2382, 2382, 2382, 2382, 2382, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.002645503  0.8960147  0.5782026
##   0.003968254  0.8975288  0.5804829
##   0.005291005  0.8991581  0.5835626
##   0.006613757  0.8992922  0.5821151
##   0.009259259  0.8997177  0.5782421
##   0.015873016  0.8979459  0.5597171
##   0.016402116  0.8980368  0.5597664
##   0.018518519  0.8978292  0.5529883
##   0.033068783  0.8952924  0.5260809
##   0.365079365  0.8801403  0.3894097
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.009259259.
```

```r
print(tree_model$finalModel)
```

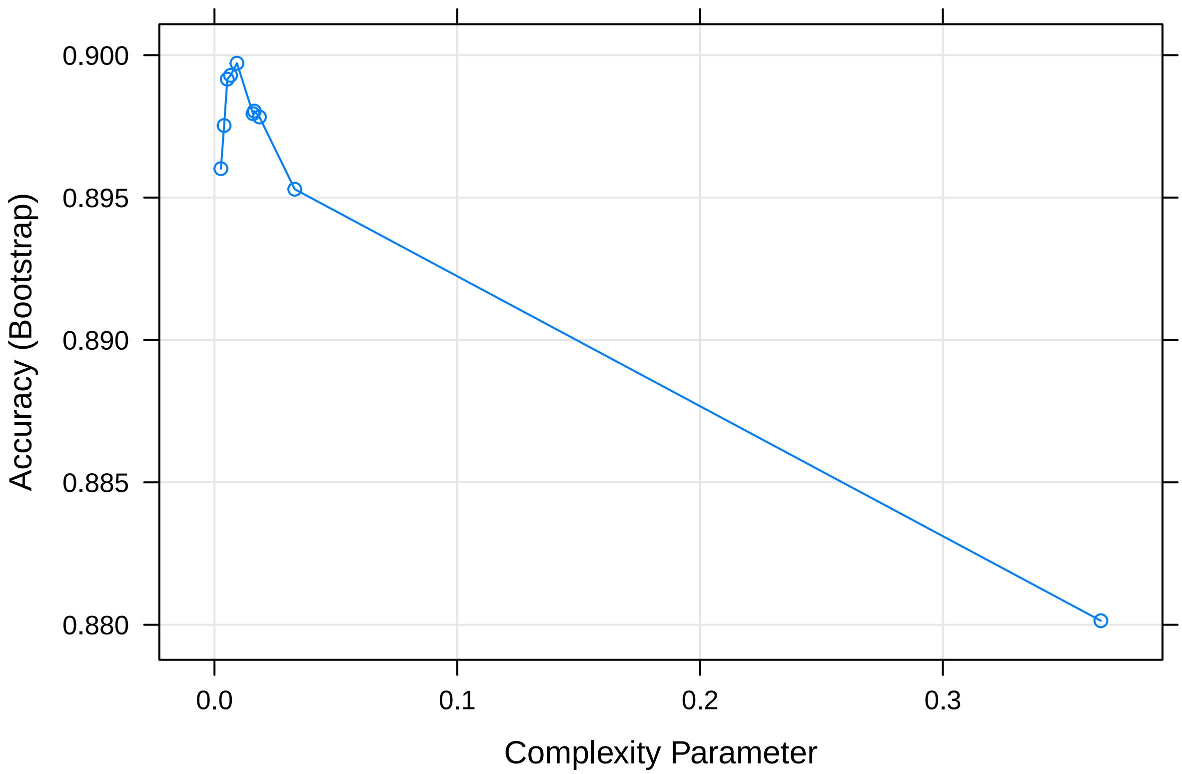
```
## n= 2382
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 2382 378 0 (0.84130982 0.15869018)
##     2) followers_count>=31.5 2136 186 0 (0.91292135 0.08707865)
##       4) friends_count>=99.5 2068 154 0 (0.92553191 0.07446809)
##         8) followers_count>=150.5 1540  67 0 (0.95649351 0.04350649) *
##         9) followers_count< 150.5 528  87 0 (0.83522727 0.16477273)
##          18) friends_count< 529 418  29 0 (0.93062201 0.06937799) *
##          19) friends_count>=529 110  52 1 (0.47272727 0.52727273)
##            38) followers_count>=75 83  34 0 (0.59036145 0.40963855)
##              76) friends_count>=968 25   4 0 (0.84000000 0.16000000) *
##              77) friends_count< 968 58  28 1 (0.48275862 0.51724138)
##               154) statuses_count>=335 20   5 0 (0.75000000 0.25000000) *
##               155) statuses_count< 335 38  13 1 (0.34210526 0.65789474) *
##            39) followers_count< 75 27   3 1 (0.11111111 0.88888889) *
```

```
##        5) friends_count< 99.5 68  32 0 (0.52941176 0.47058824)
##         10) followers_count>=73 34  10 0 (0.70588235 0.29411765) *
##         11) followers_count< 73 34  12 1 (0.35294118 0.64705882) *
##      3) followers_count< 31.5 246  54 1 (0.21951220 0.78048780)
##        6) friends_count< 286.5 96  44 0 (0.54166667 0.45833333)
##         12) friends_count>=102 37   6 0 (0.83783784 0.16216216) *
##         13) friends_count< 102 59  21 1 (0.35593220 0.64406780)
##           26) friends_count< 21.5 7   0 0 (1.00000000 0.00000000) *
##           27) friends_count>=21.5 52  14 1 (0.26923077 0.73076923) *
##        7) friends_count>=286.5 150   2 1 (0.01333333 0.98666667) *
```

```r
plot(varImp(tree_model))
```




```r
# plot accuracy by the complexity parameter
plot(tree_model)
```




```
# test the predictions
tree_predictions = predict(tree_model, newdata = testing_set)
confusionMatrix(tree_predictions, testing_set$bot)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 639  52
##          1  29  74
##
##                Accuracy : 0.898
##                  95% CI : (0.8748, 0.9182)
##     No Information Rate : 0.8413
##     P-Value [Acc > NIR] : 2.585e-06
##
##                   Kappa : 0.5874
##  Mcnemar's Test P-Value : 0.01451
##
##             Sensitivity : 0.9566
##             Specificity : 0.5873
##          Pos Pred Value : 0.9247
##          Neg Pred Value : 0.7184
##              Prevalence : 0.8413
##          Detection Rate : 0.8048
##    Detection Prevalence : 0.8703
##       Balanced Accuracy : 0.7719
##
```

```
##        'Positive' Class : 0
##
```

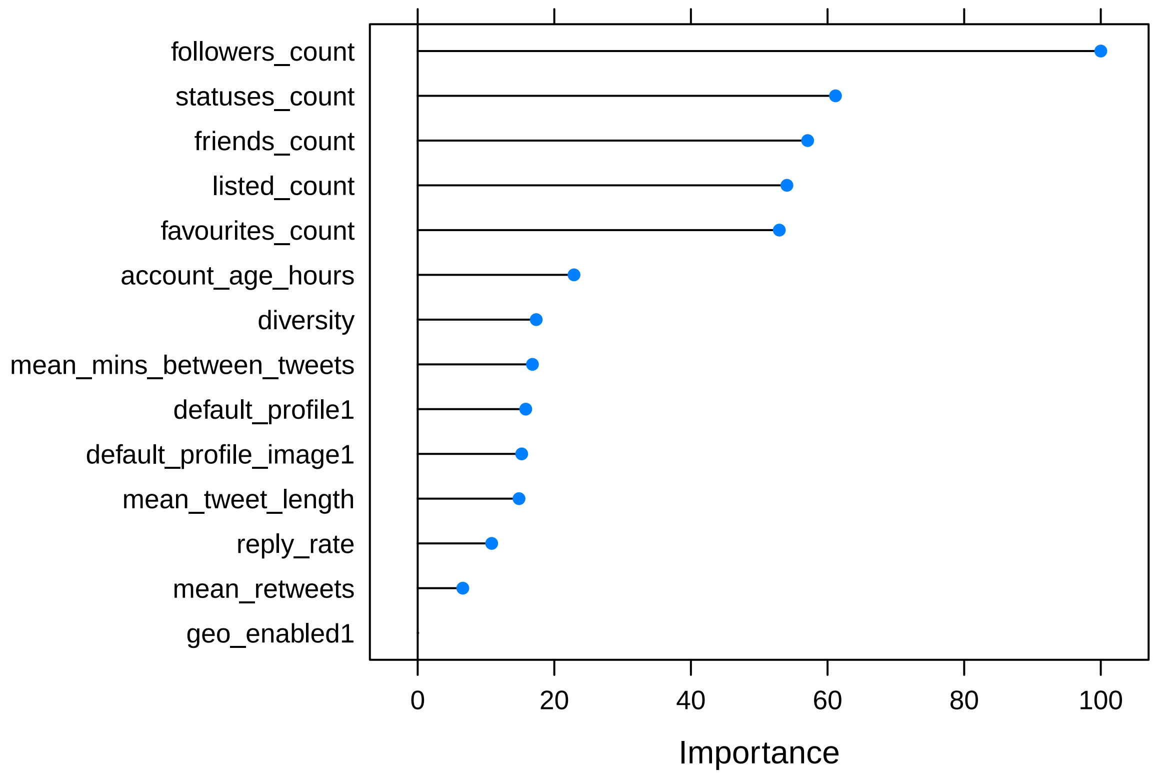
Bootstrap aggregating (bagging)

You might have to install some extra packages before this one will run. The key idea in bagging is that we resample the input data and recompute the predictions. Then, use the average or majority vote to determine the class.

```
bagged_model = train(bot ~.,
                    method = 'treebag',
                    data = training_set)

print(bagged_model)
```

```
## Bagged CART
##
## 2382 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2382, 2382, 2382, 2382, 2382, 2382, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9099259  0.6238812
##
##
```

```
plot(varImp(bagged_model))
```




```r
bagged_predictions = predict(bagged_model, testing_set)
confusionMatrix(bagged_predictions, testing_set$bot)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 652  51
##          1  16  75
##
##                Accuracy : 0.9156
##                  95% CI : (0.8941, 0.934)
##     No Information Rate : 0.8413
##     P-Value [Acc > NIR] : 4.425e-10
##
##                   Kappa : 0.6438
##  Mcnemar's Test P-Value : 3.271e-05
##
##             Sensitivity : 0.9760
##             Specificity : 0.5952
##          Pos Pred Value : 0.9275
##          Neg Pred Value : 0.8242
##              Prevalence : 0.8413
##          Detection Rate : 0.8212
##    Detection Prevalence : 0.8854
##       Balanced Accuracy : 0.7856
##
##        'Positive' Class : 0
##
```

In this case, we do get some accuracy gains from bagging.

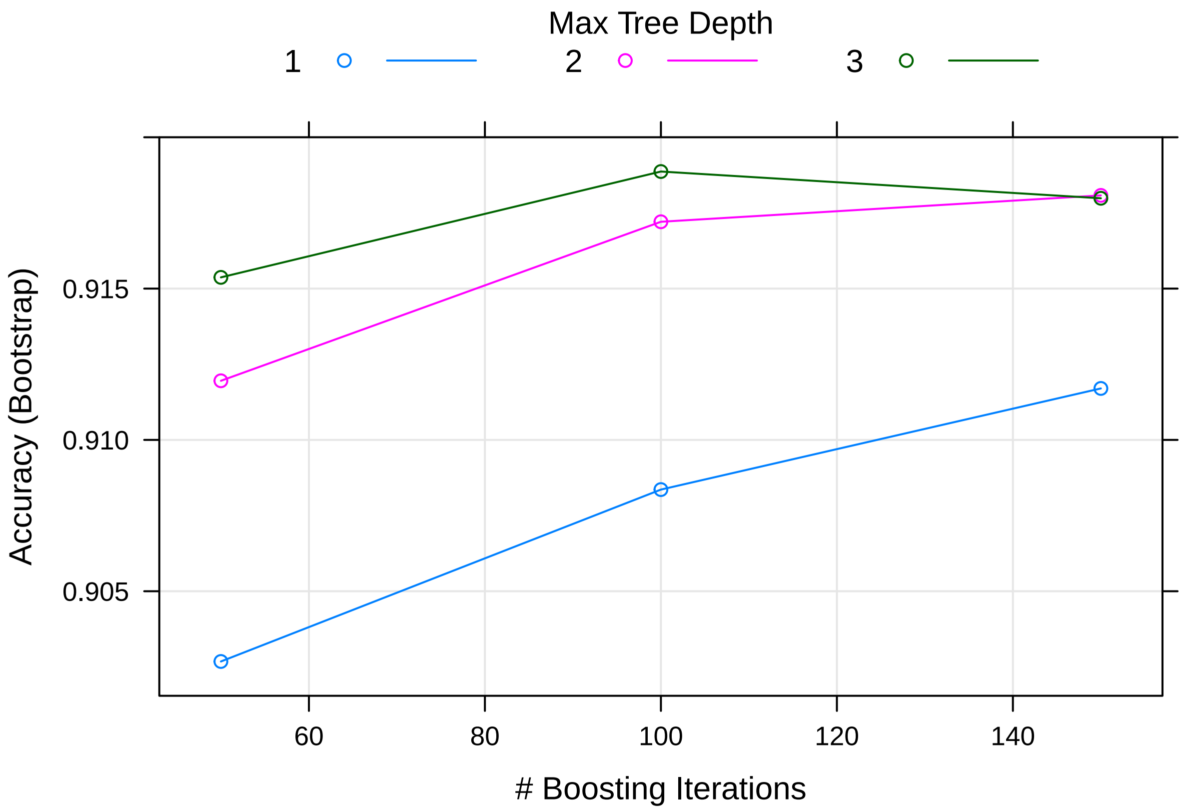
Boosting

The key idea of boosting is that we amplify the signal of weak predictors by up-weighting misclassified observations at each split point.

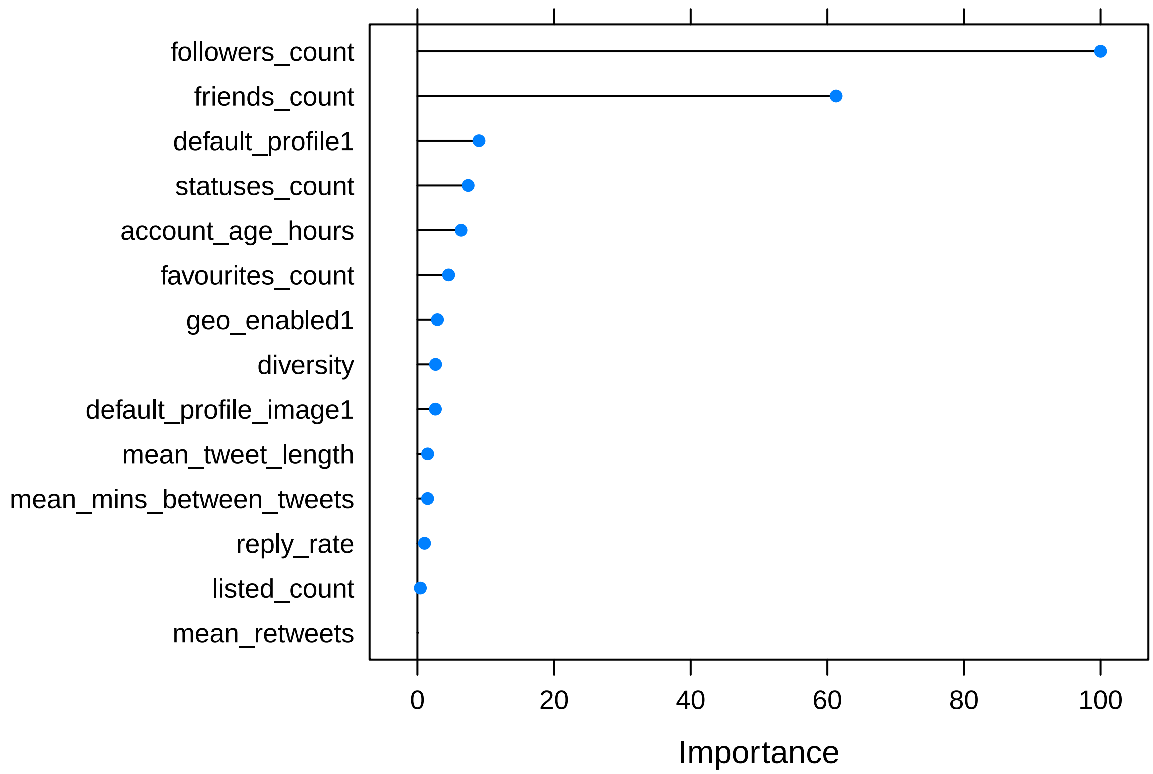
```r
boost_model = train(bot ~.,
                    method = 'gbm',
                    data = training_set,
                    verbose = FALSE)

print(boost_model)
```

```
## Stochastic Gradient Boosting
##
## 2382 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2382, 2382, 2382, 2382, 2382, 2382, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9026786  0.5660969
##   1                  100      0.9083585  0.5998731
##   1                  150      0.9117005  0.6194439
##   2                   50      0.9119541  0.6166607
##   2                  100      0.9172078  0.6519053
##   2                  150      0.9180795  0.6591229
##   3                   50      0.9153706  0.6402409
##   3                  100      0.9188683  0.6638209
##   3                  150      0.9179840  0.6639106
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 100,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

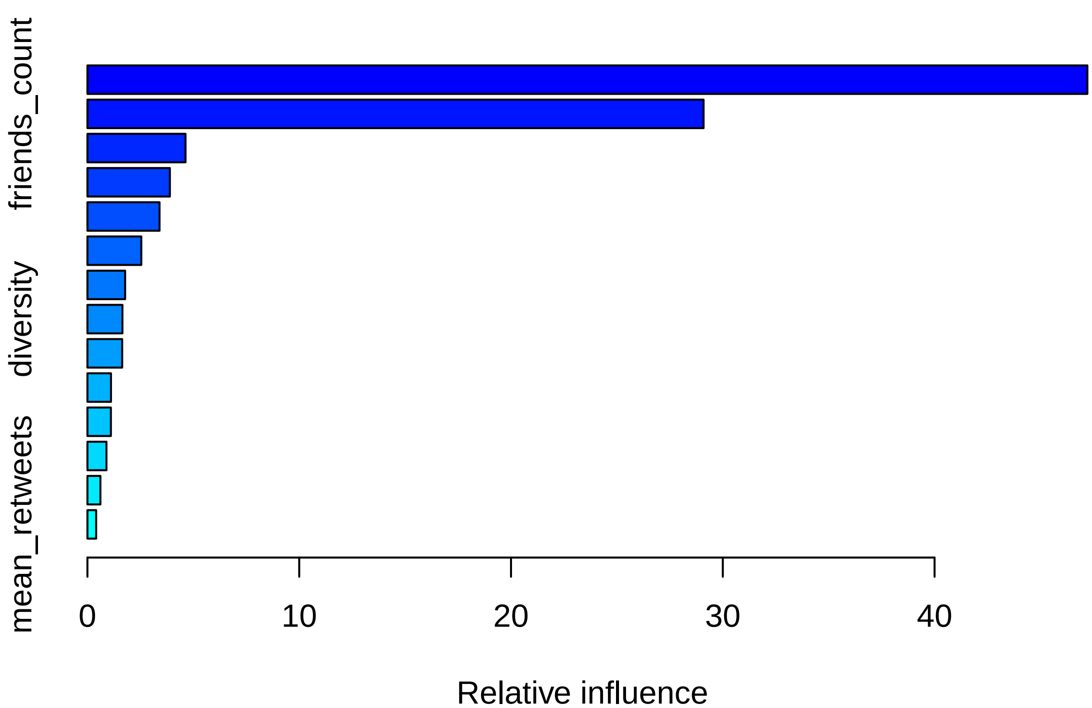
```r
plot(boost_model)
```




```r
plot(varImp(boost_model))
```




```r
#TODO: remove this?
summary(boost_model$finalModel)
```




```
##                                                    var    rel.inf
## followers_count                        followers_count 47.2144335
```

```
## friends_count                          friends_count 29.0930492
## default_profile1                    default_profile1  4.6327624
## statuses_count                        statuses_count  3.8931255
## account_age_hours                  account_age_hours  3.4048035
## favourites_count                    favourites_count  2.5434848
## geo_enabled1                            geo_enabled1  1.7800263
## diversity                                  diversity  1.6530707
## default_profile_image1        default_profile_image1  1.6402545
## mean_tweet_length                  mean_tweet_length  1.1121697
## mean_mins_between_tweets mean_mins_between_tweets  1.1080210
## reply_rate                                reply_rate  0.8990138
## listed_count                            listed_count  0.6135349
## mean_retweets                          mean_retweets  0.4122503
```

```
# predict
boost_predictions = predict(boost_model, testing_set)
confusionMatrix(boost_predictions, testing_set$bot)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 659  49
##          1   9  77
##
##                Accuracy : 0.927
##                  95% CI : (0.9066, 0.9441)
##     No Information Rate : 0.8413
##     P-Value [Acc > NIR] : 3.324e-13
##
##                   Kappa : 0.686
##  Mcnemar's Test P-Value : 3.040e-07
##
##             Sensitivity : 0.9865
##             Specificity : 0.6111
##          Pos Pred Value : 0.9308
##          Neg Pred Value : 0.8953
##              Prevalence : 0.8413
##          Detection Rate : 0.8300
##    Detection Prevalence : 0.8917
##       Balanced Accuracy : 0.7988
##
##        'Positive' Class : 0
##
```

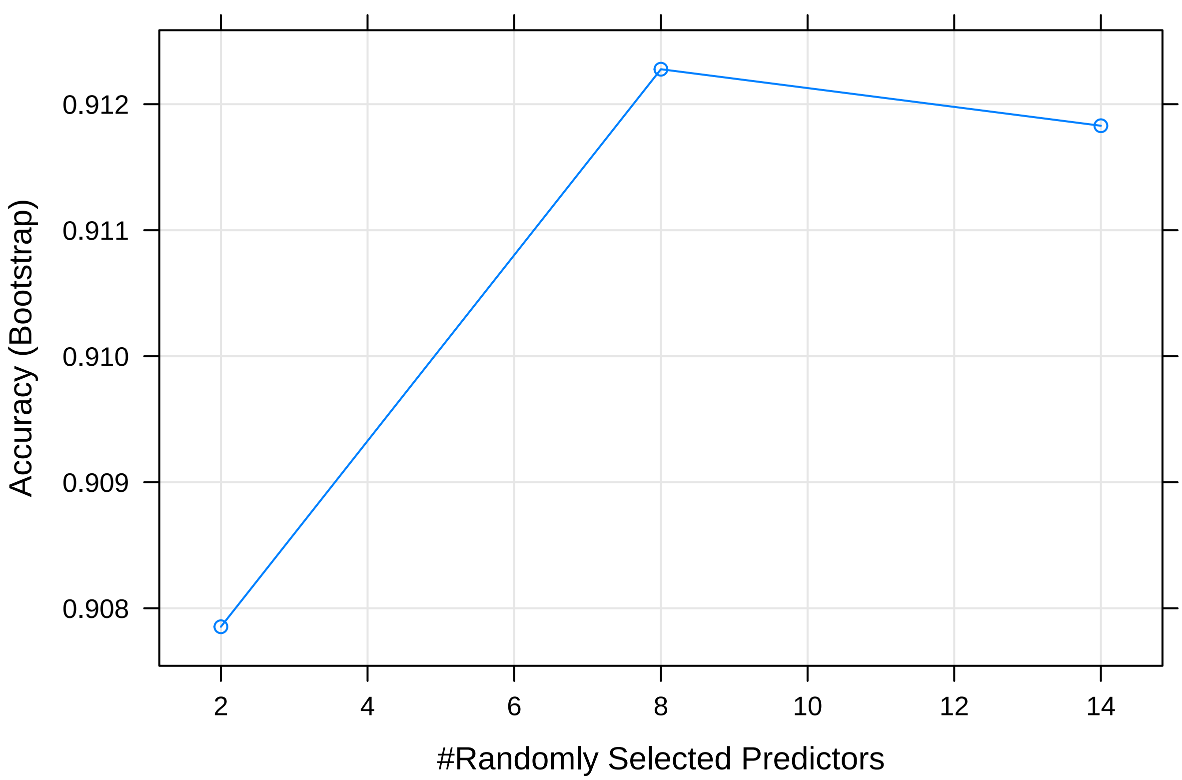
Random Forest

Random forest is a bagging method where we resample both obervations, and variables, grow multiple trees and aggregate votes. It's one of the most accurate classifiers, but can be slow. Might want to run this one at home…

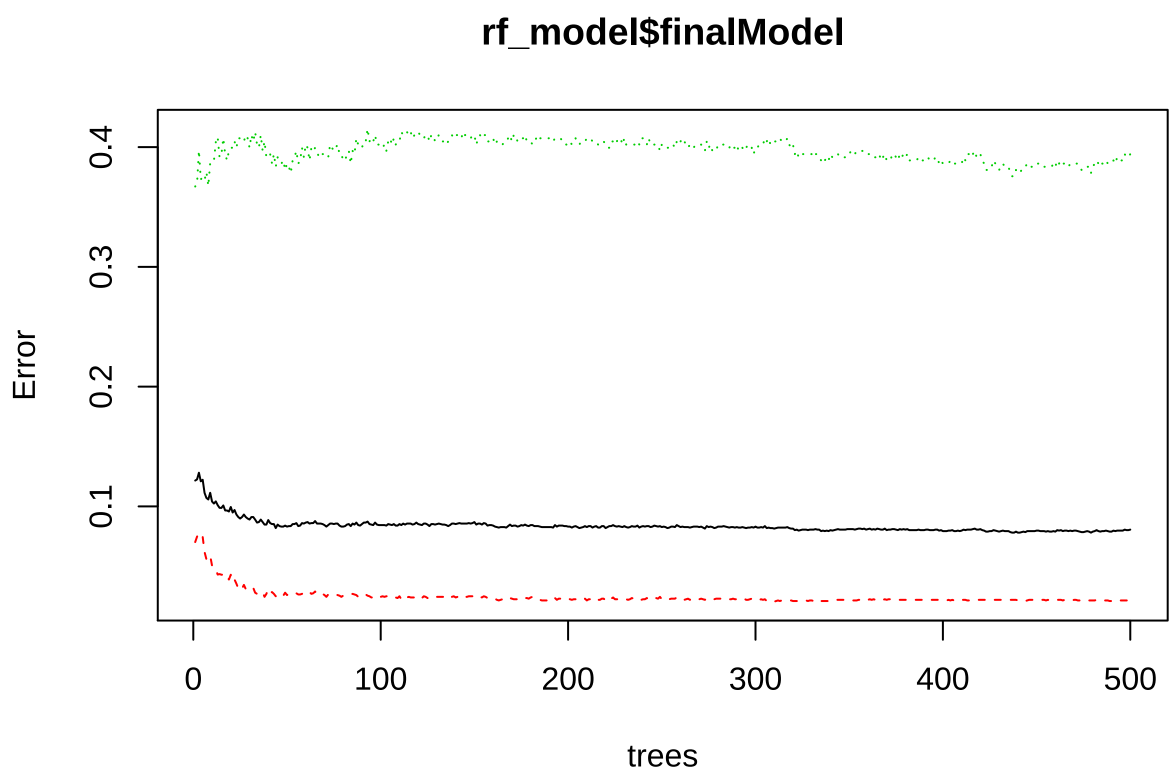
```r
rf_model = train(bot ~.,
                 data = training_set,
                 method = 'rf',
                 prox = TRUE,
                 verbose = TRUE)

print(rf_model)
```

```
## Random Forest
##
## 2382 samples
##   14 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2382, 2382, 2382, 2382, 2382, 2382, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9078537  0.5868531
##    8    0.9122775  0.6217937
##   14    0.9118290  0.6279741
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 8.
```

```r
plot(rf_model)
```



```r
plot(rf_model$finalModel)
```




```r
# pull a tree out of the forest
head(getTree(rf_model$finalModel, k = 5, labelVar = TRUE))
```

```
##   left daughter right daughter       split var split point status
## 1             2              3    listed_count         3.5      1
## 2             4              5   friends_count       529.0      1
## 3             6              7   friends_count        99.5      1
## 4             8              9   friends_count       100.5      1
## 5            10             11 followers_count        77.5      1
## 6            12             13    listed_count        18.0      1
##   prediction
## 1       <NA>
## 2       <NA>
## 3       <NA>
## 4       <NA>
## 5       <NA>
## 6       <NA>
```

```r
# predict
rf_predictions = predict(rf_model, testing_set)
confusionMatrix(rf_predictions, testing_set$bot)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1
##          0 656  55
##          1  12  71
##
##                Accuracy : 0.9156
##                  95% CI : (0.8941, 0.934)
##     No Information Rate : 0.8413
##     P-Value [Acc > NIR] : 4.425e-10
##
##                   Kappa : 0.6332
##  Mcnemar's Test P-Value : 2.880e-07
##
##             Sensitivity : 0.9820
##             Specificity : 0.5635
##          Pos Pred Value : 0.9226
##          Neg Pred Value : 0.8554
##              Prevalence : 0.8413
##          Detection Rate : 0.8262
##    Detection Prevalence : 0.8955
##       Balanced Accuracy : 0.7728
##
##        'Positive' Class : 0
##
```

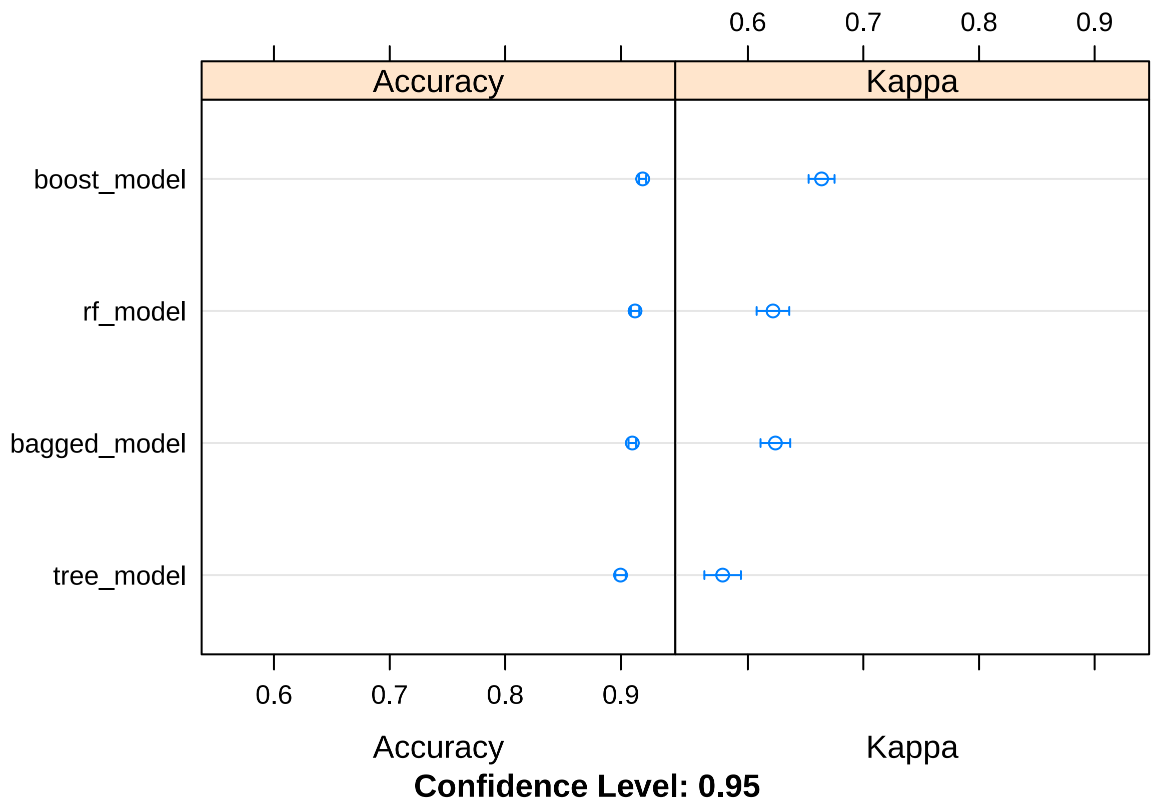
As always, we can compare the models with the `resamples` function.

```r
# compare the three methods
results = resamples(list(tree_model = tree_model,
                         bagged_model = bagged_model,
                         boost_model = boost_model,
                         rf_model = rf_model))

# compare accuracy and kappa
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: tree_model, bagged_model, boost_model, rf_model
## Number of resamples: 25
##
## Accuracy
##                Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## tree_model   0.8781  0.8916 0.8996 0.8997  0.9061 0.9195    0
## bagged_model 0.8920  0.9051 0.9121 0.9099  0.9156 0.9240    0
## boost_model  0.9082  0.9128 0.9194 0.9189  0.9234 0.9352    0
## rf_model     0.8954  0.9051 0.9121 0.9123  0.9176 0.9308    0
##
## Kappa
##                Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## tree_model   0.5085  0.5502 0.5750 0.5782  0.6050 0.6528    0
## bagged_model 0.5717  0.5988 0.6288 0.6239  0.6496 0.6756    0
```

```
## boost_model  0.6110  0.6432 0.6634 0.6638  0.6825 0.7134    0
## rf_model     0.5548  0.5936 0.6190 0.6218  0.6498 0.6817    0
```

```
# plot results
dotplot(results)
```

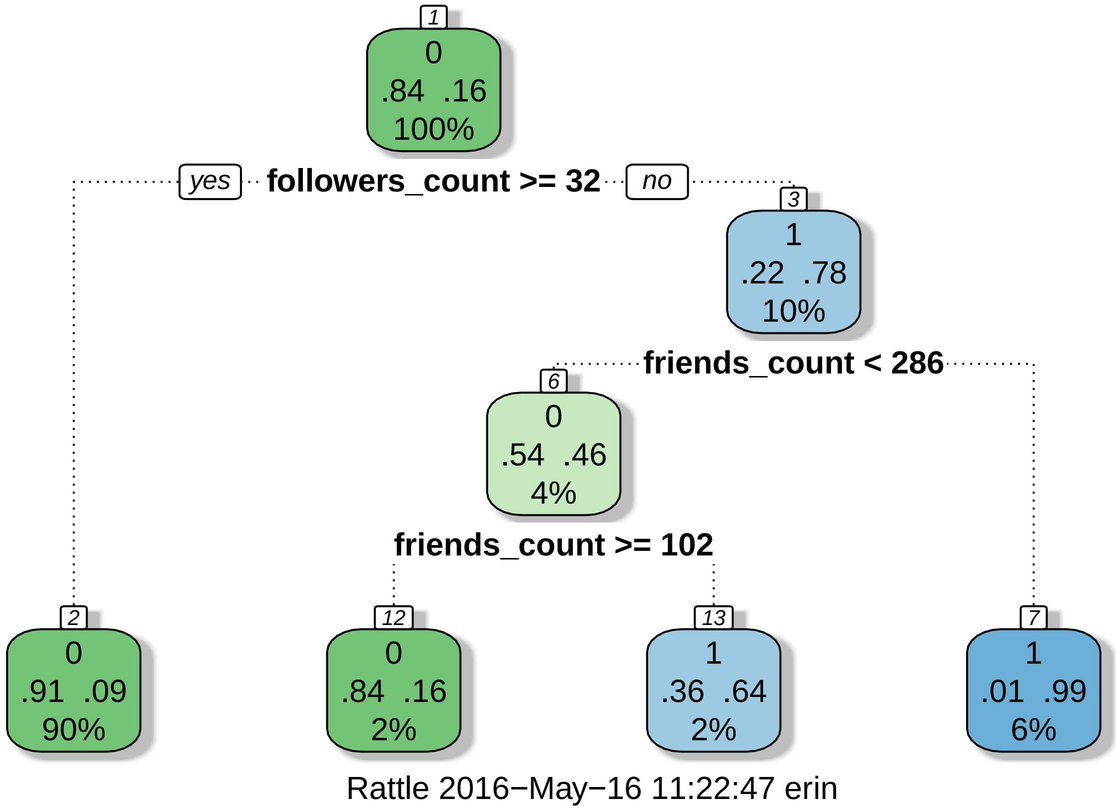


How do the tree models compare with logistic regression?

Making prettier trees

If you want to make your trees look a little prettier, try out the `rattle` package.

```
library(rattle)

tree_model = train(bot ~.,
                   method = 'rpart',
                   data = training_set)

# plot the final model
fancyRpartPlot(tree_model$finalModel)
```


1
0
.84 .16
100%
yes
followers_count >= 32
no
3
1
.22 .78
10%
friends_count < 286
6
0
.54 .46
4%
friends_count >= 102
2
0
.91 .09
90%
12
0
.84 .16
2%
13
1
.36 .64
2%
7
1
.01 .99
6%
Rattle 2016−May−16 11:22:47 erin