Classification Trees in R

Erin Shellman

May 11, 2015

Contents

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

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

data = 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)
data$bot = factor(data$bot)
data$default_profile = factor(data$default_profile)
data$default_profile_image = factor(data$default_profile_image)
data$geo_enabled = factor(data$geo_enabled)
data$profile_background_tile = factor(data$profile_background_tile)
data$verified = factor(data$verified)

set.seed(243)
data = na.omit(data)

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

train = data[in_train, ]
test = data[-in_train, ]

# drop the ids
train$id = NULL
test$id = NULL
```

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.

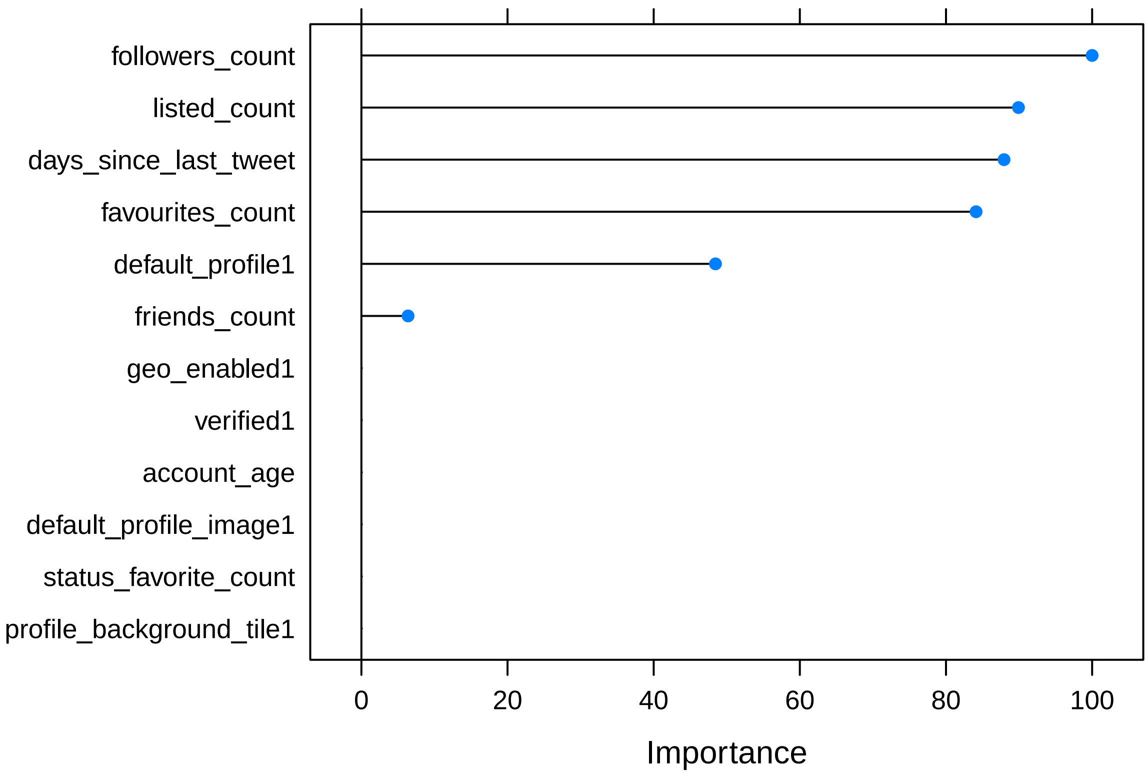
```
tree_model = train(factor(bot) ~.,
                   method = 'rpart',
                   data = train)
print(tree_model)
```

```
## CART
##
## 4488 samples
##   12 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 4488, 4488, 4488, 4488, 4488, 4488, ...
##
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.01409774  0.9321568  0.8643244  0.008268356  0.01647921
##   0.01879699  0.9262423  0.8526233  0.008589181  0.01720103
##   0.83364662  0.7136796  0.4000407  0.199347900  0.42497681
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.01409774.
```

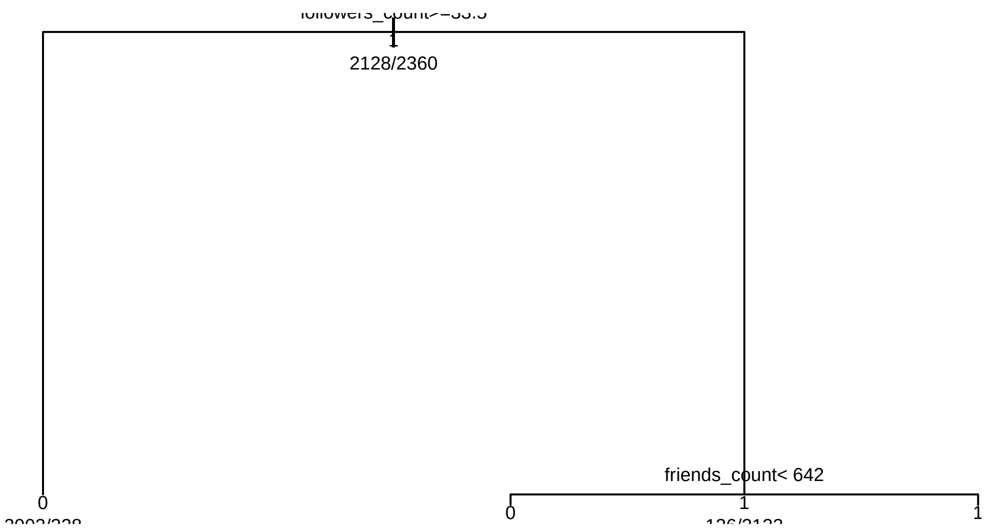
```
print(tree_model$finalModel)
```

```
## n= 4488
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 4488 2128 1 (0.47415330 0.52584670)
##   2) followers_count>=33.5 2230  228 0 (0.89775785 0.10224215) *
##   3) followers_count< 33.5 2258  126 1 (0.05580159 0.94419841)
##     6) friends_count< 642 142   51 0 (0.64084507 0.35915493) *
##     7) friends_count>=642 2116   35 1 (0.01654064 0.98345936) *
```

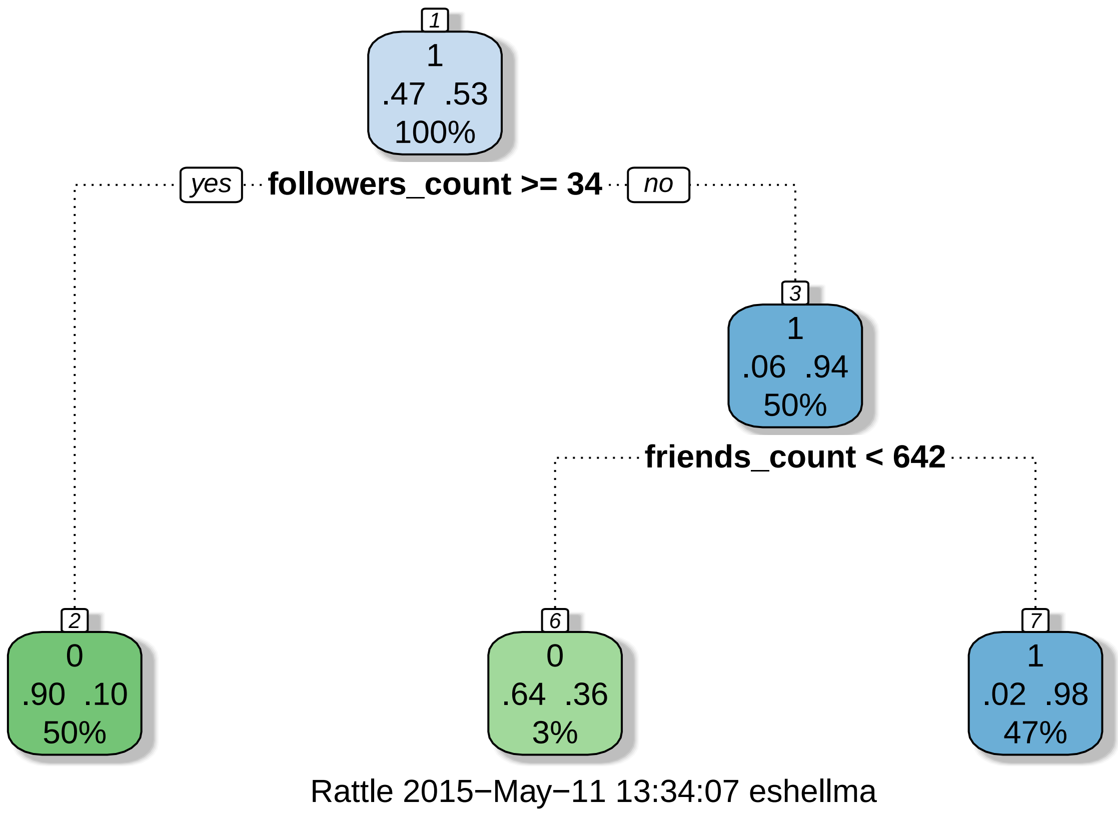
```
plot(varImp(tree_model))
```




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




```r
# we can do better!
library(rattle)
fancyRpartPlot(tree_model$finalModel)
```




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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 699 101
##          1  10 685
##
##                Accuracy : 0.9258
##                  95% CI : (0.9113, 0.9385)
##     No Information Rate : 0.5258
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.852
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9859
##             Specificity : 0.8715
##          Pos Pred Value : 0.8738
##          Neg Pred Value : 0.9856
##              Prevalence : 0.4742
##          Detection Rate : 0.4676
##    Detection Prevalence : 0.5351
##       Balanced Accuracy : 0.9287
```

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

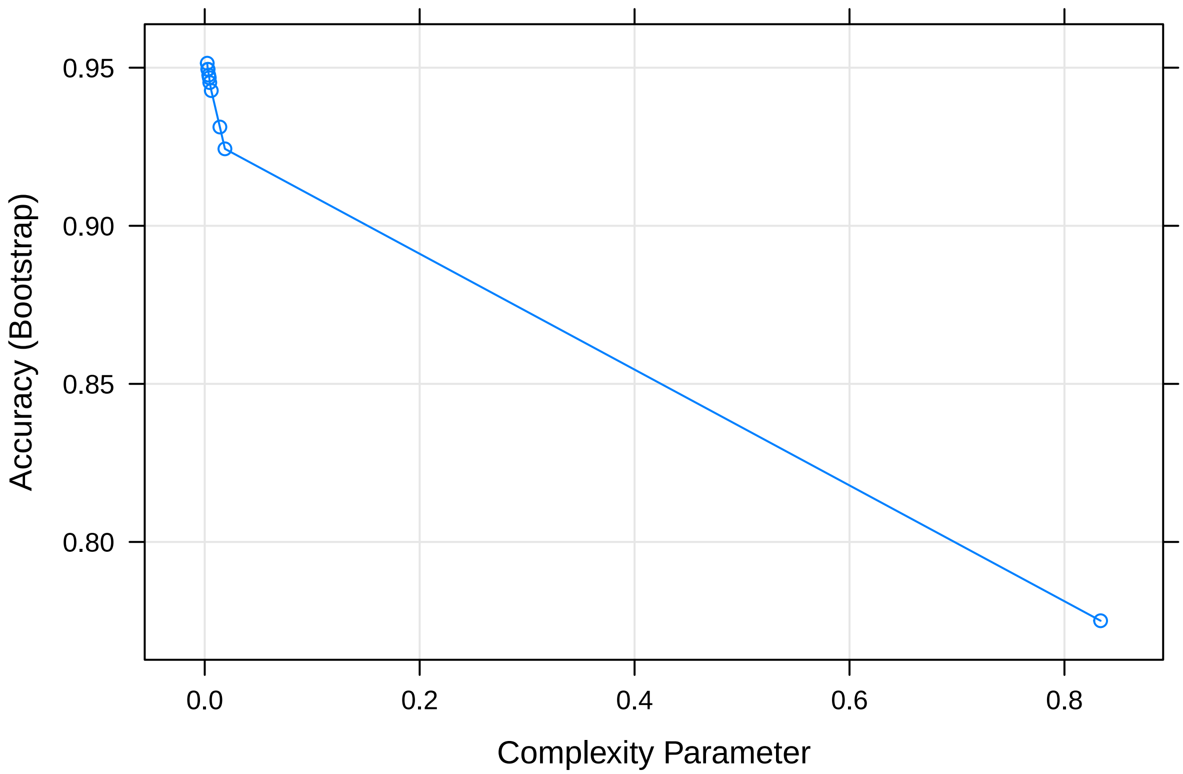
```
## CART
##
## 4488 samples
##   12 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 4488, 4488, 4488, 4488, 4488, 4488, ...
##
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.002349624  0.9514192  0.9025502  0.003887686  0.007819597
##   0.002819549  0.9494871  0.8986908  0.004378522  0.008751017
##   0.003132832  0.9494924  0.8987059  0.004628144  0.009219270
##   0.003759398  0.9476518  0.8950322  0.004474902  0.008870289
##   0.004229323  0.9468231  0.8933701  0.004413713  0.008733124
##   0.004699248  0.9452913  0.8903286  0.004212130  0.008352524
##   0.006109023  0.9427660  0.8853236  0.004810986  0.009561072
##   0.014097744  0.9312392  0.8626491  0.007800572  0.015491472
##   0.018796992  0.9243400  0.8488359  0.007446724  0.015017566
##   0.833646617  0.7750751  0.5343363  0.193970684  0.409111351
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.002349624.
```

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

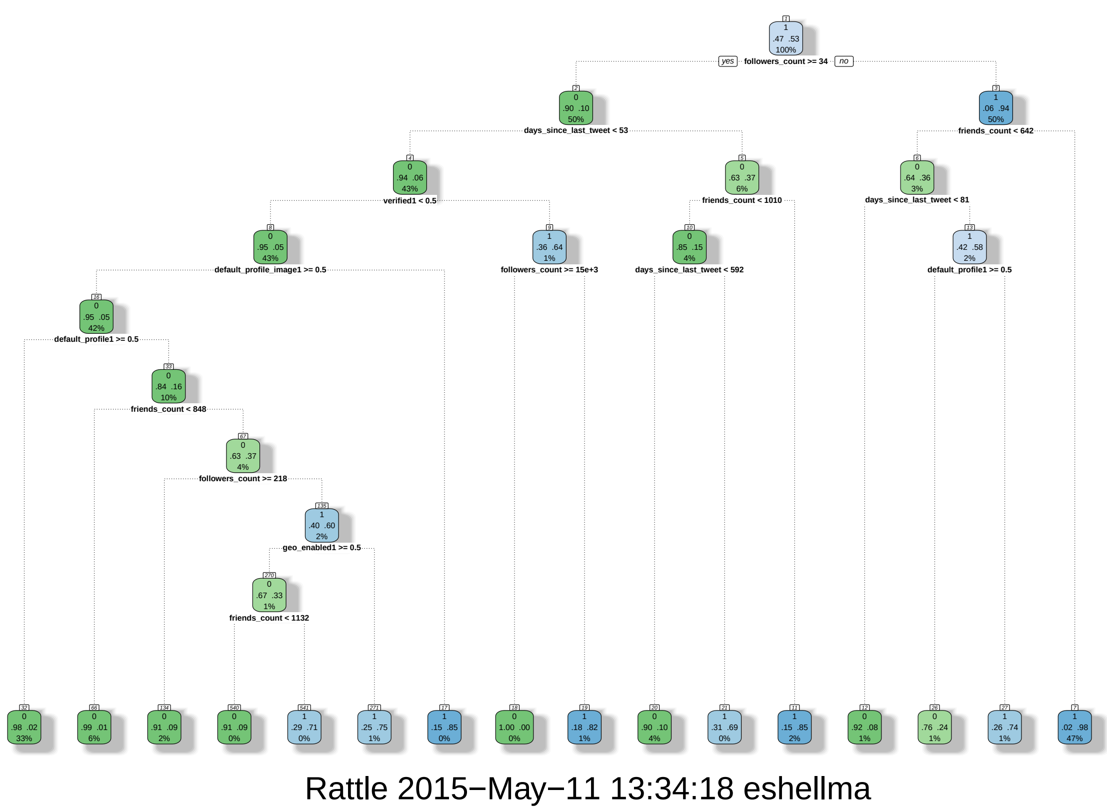
```
## n= 4488
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 4488 2128 1 (0.474153298 0.525846702)
##     2) followers_count>=33.5 2230  228 0 (0.897757848 0.102242152)
##       4) days_since_last_tweet< 53.21569 1952  126 0 (0.935450820 0.064549180)
##         8) verified1< 0.5 1916  103 0 (0.946242171 0.053757829)
```

```
##          16) default_profile_image1>=0.5 1903   92 0 (0.951655281 0.048344719)
##            32) default_profile1>=0.5 1459   23 0 (0.984235778 0.015764222) *
##            33) default_profile1< 0.5 444   69 0 (0.844594595 0.155405405)
##              66) friends_count< 847.5 263    2 0 (0.992395437 0.007604563) *
##              67) friends_count>=847.5 181   67 0 (0.629834254 0.370165746)
##               134) followers_count>=218.5 81    7 0 (0.913580247 0.086419753) *
##               135) followers_count< 218.5 100   40 1 (0.400000000 0.600000000)
##                 270) geo_enabled1>=0.5 36   12 0 (0.666666667 0.333333333)
##                   540) friends_count< 1132.5 22    2 0 (0.909090909 0.090909091) *
##                   541) friends_count>=1132.5 14    4 1 (0.285714286 0.714285714) *
##                 271) geo_enabled1< 0.5 64   16 1 (0.250000000 0.750000000) *
##          17) default_profile_image1< 0.5 13    2 1 (0.153846154 0.846153846) *
##         9) verified1>=0.5 36   13 1 (0.361111111 0.638888889)
##          18) followers_count>=15064 8    0 0 (1.000000000 0.000000000) *
##          19) followers_count< 15064 28    5 1 (0.178571429 0.821428571) *
##       5) days_since_last_tweet>=53.21569 278  102 0 (0.633093525 0.366906475)
##        10) friends_count< 1010 192   29 0 (0.848958333 0.151041667)
##          20) days_since_last_tweet< 591.677 176   18 0 (0.897727273 0.102272727) *
##          21) days_since_last_tweet>=591.677 16    5 1 (0.312500000 0.687500000) *
##        11) friends_count>=1010 86   13 1 (0.151162791 0.848837209) *
##     3) followers_count< 33.5 2258  126 1 (0.055801594 0.944198406)
##       6) friends_count< 642 142   51 0 (0.640845070 0.359154930)
##        12) days_since_last_tweet< 80.97859 63    5 0 (0.920634921 0.079365079) *
##        13) days_since_last_tweet>=80.97859 79   33 1 (0.417721519 0.582278481)
##          26) default_profile1>=0.5 25    6 0 (0.760000000 0.240000000) *
##          27) default_profile1< 0.5 54   14 1 (0.259259259 0.740740741) *
##       7) friends_count>=642 2116   35 1 (0.016540643 0.983459357) *
```

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




```r
library(rattle)
fancyRpartPlot(tree_model$finalModel)
```



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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 667  40
##          1  42 746
##
##                Accuracy : 0.9452
##                  95% CI : (0.9324, 0.9561)
##     No Information Rate : 0.5258
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.89
##  Mcnemar's Test P-Value : 0.9121
##
##             Sensitivity : 0.9408
##             Specificity : 0.9491
##          Pos Pred Value : 0.9434
##          Neg Pred Value : 0.9467
##              Prevalence : 0.4742
##          Detection Rate : 0.4462
##    Detection Prevalence : 0.4729
##       Balanced Accuracy : 0.9449
##
##        '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.

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

```
## Loading required package: ipred
## Loading required package: plyr
## ------------------------------------------------------------------------
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## ------------------------------------------------------------------------
##
## Attaching package: 'plyr'
##
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```r
print(bagged_model)
```

```
## Bagged CART
##
## 4488 samples
##   12 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 4488, 4488, 4488, 4488, 4488, 4488, ...
##
## Resampling results
##
##   Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.9610437  0.9218822  0.003933348  0.007873697
##
##
```

```r
print(bagged_model$finalModel)
```

```
##
## Bagging classification trees with 25 bootstrap replications
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 685  30
##          1  24 756
##
##                Accuracy : 0.9639
##                  95% CI : (0.9531, 0.9728)
##     No Information Rate : 0.5258
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9276
##  Mcnemar's Test P-Value : 0.4962
##
##             Sensitivity : 0.9661
##             Specificity : 0.9618
##          Pos Pred Value : 0.9580
##          Neg Pred Value : 0.9692
```

```
##               Prevalence : 0.4742
##           Detection Rate : 0.4582
##     Detection Prevalence : 0.4783
##        Balanced Accuracy : 0.9640
##
##         '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 = train,
                    verbose = FALSE)
```

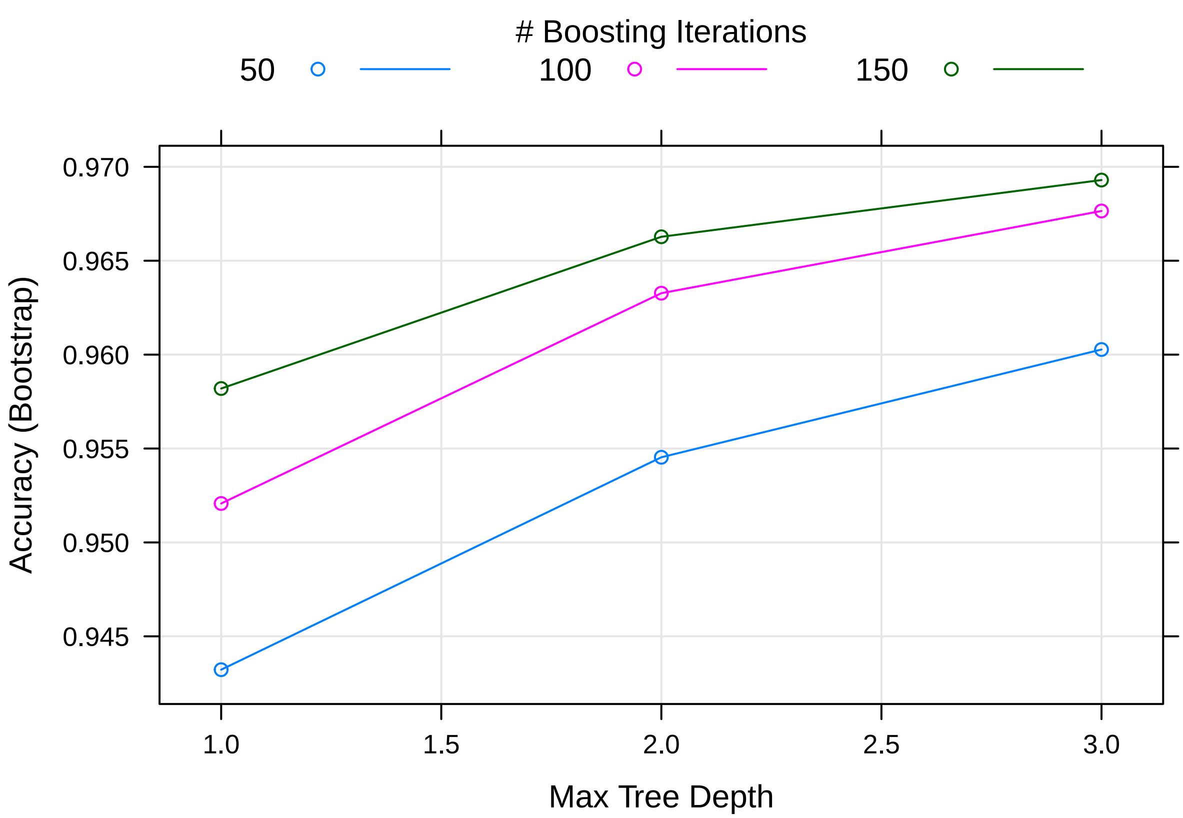
```
## Loading required package: gbm
## Loading required package: survival
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.1
```

```r
print(boost_model)
```

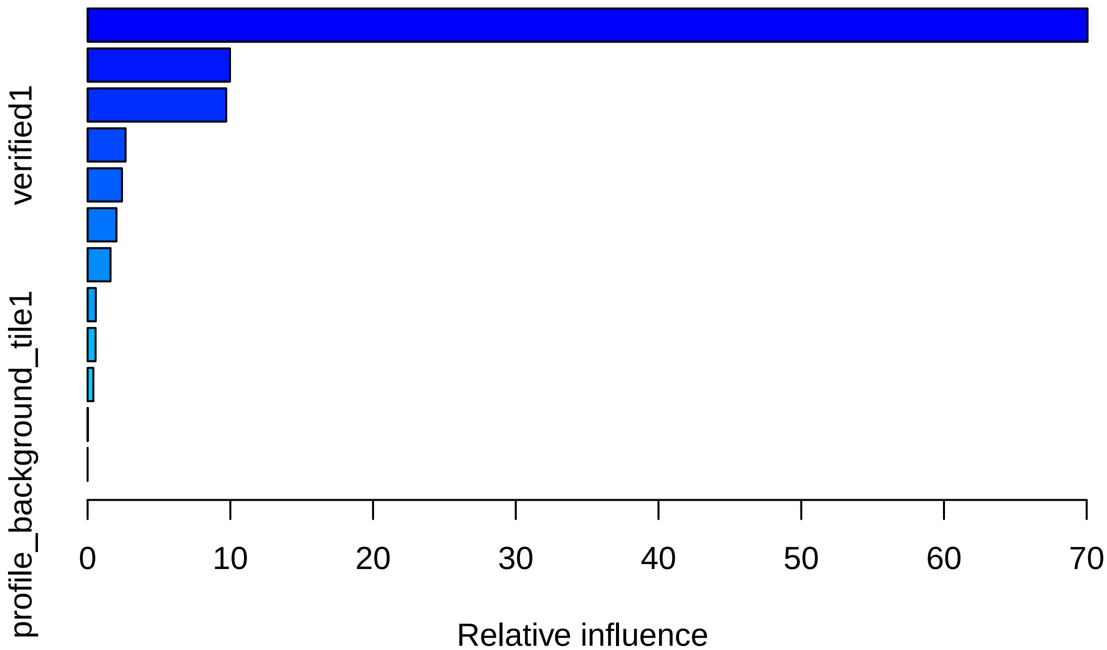
```
## Stochastic Gradient Boosting
##
## 4488 samples
##   12 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 4488, 4488, 4488, 4488, 4488, 4488, ...
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa      Accuracy SD
##   1                   50      0.9432229  0.8863083  0.005503738
##   1                  100      0.9520733  0.9039854  0.005621995
```

```
##    1                    150       0.9581941  0.9162224  0.004475931
##    2                     50       0.9545352  0.9089731  0.005311355
##    2                    100       0.9632713  0.9263912  0.004658999
##    2                    150       0.9662743  0.9323858  0.004011909
##    3                     50       0.9602718  0.9203928  0.004798436
##    3                    100       0.9676486  0.9351325  0.004302051
##    3                    150       0.9692963  0.9384304  0.003773682
##    Kappa SD
##    0.011058789
##    0.011292777
##    0.008980821
##    0.010605139
##    0.009331382
##    0.008038162
##    0.009595433
##    0.008609035
##    0.007537283
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3 and shrinkage = 0.1.
```

```r
plot(boost_model)
```




```r
summary(boost_model$finalModel)
```




```
##                                               var    rel.inf
## followers_count                   followers_count 70.0602266
## friends_count                       friends_count  9.9759629
## days_since_last_tweet       days_since_last_tweet  9.7128842
## verified1                               verified1  2.6618917
## listed_count                         listed_count  2.4089257
## default_profile1                 default_profile1  2.0222701
## default_profile_image1     default_profile_image1  1.6068213
## geo_enabled1                         geo_enabled1  0.5748611
## favourites_count                 favourites_count  0.5578347
## account_age                           account_age  0.4037014
## status_favorite_count       status_favorite_count  0.0146202
## profile_background_tile1 profile_background_tile1  0.0000000
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 691  27
##          1  18 759
##
##                Accuracy : 0.9699
##                  95% CI : (0.9599, 0.978)
##     No Information Rate : 0.5258
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9397
##  Mcnemar's Test P-Value : 0.233
```

```
##
##             Sensitivity : 0.9746
##             Specificity : 0.9656
##          Pos Pred Value : 0.9624
##          Neg Pred Value : 0.9768
##              Prevalence : 0.4742
##          Detection Rate : 0.4622
##    Detection Prevalence : 0.4803
##       Balanced Accuracy : 0.9701
##
##        '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 = train,
                 method = 'rf',
                 prox = TRUE,
                 verbose = TRUE)

print(rf_model)
summary(rf_model)
plot(rf_model)
plot(rf_model$finalModel)

# pull a tree out of the forest
getTree(rf_model$finalModel, k = 5)

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

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

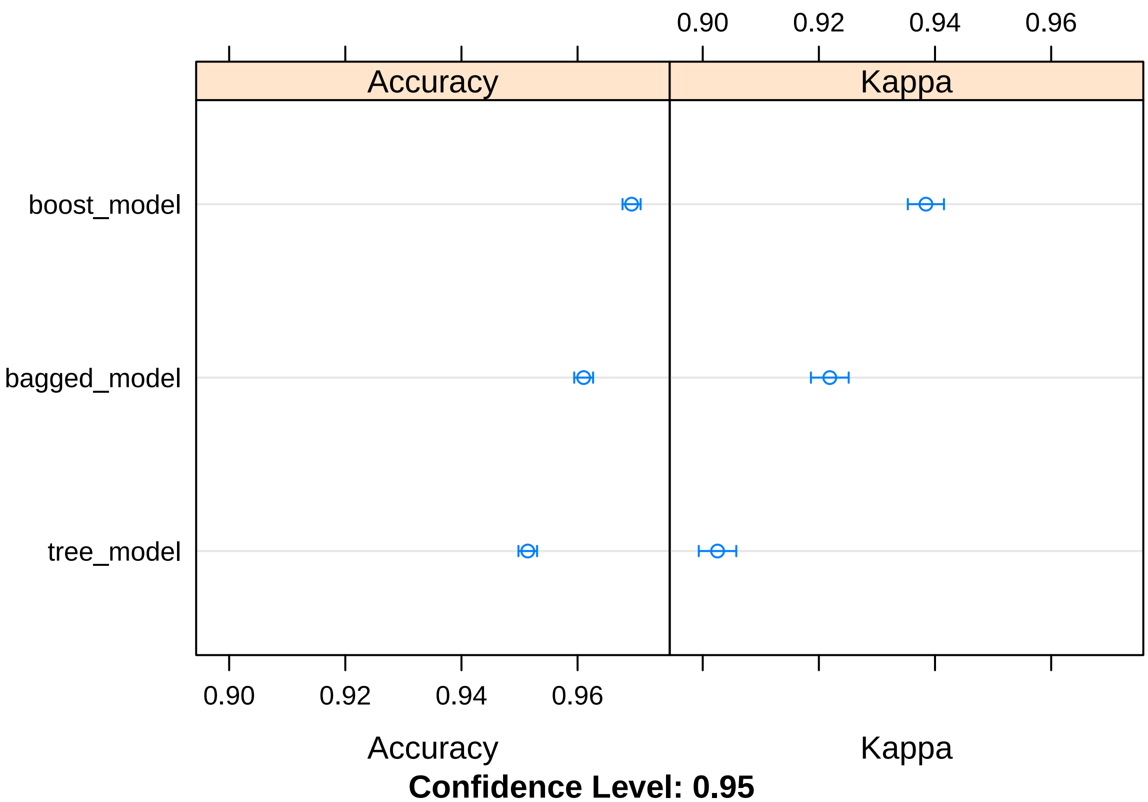
```r
# compare
results = resamples(list(tree_model = tree_model,
                         bagged_model = bagged_model,
                         boost_model = boost_model))

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

```
##
## Call:
## summary.resamples(object = results)
##
## Models: tree_model, bagged_model, boost_model
```

```
## Number of resamples: 25
##
## Accuracy
##                Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## tree_model   0.9451  0.9480 0.9507 0.9514  0.9538 0.9617    0
## bagged_model 0.9536  0.9586 0.9602 0.9610  0.9634 0.9709    0
## boost_model  0.9601  0.9676 0.9694 0.9693  0.9718 0.9776    0
##
## Kappa
##                Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## tree_model   0.8899  0.8958 0.9010 0.9026  0.9075 0.9232    0
## bagged_model 0.9070  0.9172 0.9203 0.9219  0.9264 0.9416    0
## boost_model  0.9200  0.9351 0.9387 0.9384  0.9436 0.9551    0
```

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



How do the tree models compare with logistic regression?