GRAD

STAT 149 Spring 2016 Final Project, Process Workbook #2

Submitted by: Kendrick Lo (Harvard ID: 70984997), Amy Lee (Harvard ID: 60984077)

This is a continuation of exploration of models from Process Workbook #1, and focuses primarily on non-parametric models.

```r
train = read.csv("~/Desktop/stat149/StatKaggle/train_add3.csv", header=TRUE)
test = read.csv("~/Desktop/stat149/StatKaggle/test_add3.csv", header=TRUE)
Id = read.csv("~/Desktop/stat149/StatKaggle/id.csv", header=TRUE)
```

Random Forests

Random forests is an "ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set." (Wikipedia)

In this manner, we "automatically" achieve the benefits of cross-validation when modelling using a Random Forest, and these models are capable of discovering interactions between predictors without specifically coding them into the data.

We can also use the out-of-bag error (OOB), which R provides as output, as a measure of how well the model will perform on unseen data.

Variable conversion

The following error was issued when first attempting to fit the `randomForest` model:

> Error in randomForest.default(m, y, ...) : Can not handle categorical predictors with more than 53 categories.

It appears the `region` variable is causing issues because it has too many levels.

A quick fix would be to remove some of the regions and shrink the categorical variable. We decided to try this first, and convert the three largest ones (over 4000 counts, about twice as large as any others) into their own categorical variables, replace these three with a single level in the original data, and then reset the levels of the original `region` categorical variable.

```r
# example code
train$regA = as.factor(train$region=="reg102")
train$regB = as.factor(train$region=="reg112")
train$regC = as.factor(train$region=="reg127")
train$region[train$region=="reg102"] = NA
train$region[train$region=="reg112"] = NA
train$region[train$region=="reg127"] = NA
tempreg = as.character(train$region)
tempreg[is.na(tempreg)] = "tempbig"
```

```r
train$region = as.factor(tempreg)

test$regA = as.factor(test$region=="reg102")
test$regB = as.factor(test$region=="reg112")
test$regC = as.factor(test$region=="reg127")
test$region[test$region=="reg102"] = NA
test$region[test$region=="reg112"] = NA
test$region[test$region=="reg127"] = NA
tempreg = as.character(test$region)
tempreg[is.na(tempreg)] = "tempbig"
test$region = as.factor(tempreg)
```

However, we ultimately went with an alternative method of encoding: we used the proportion of members in each region within the training set as a substitute for the level of the factor. In this manner, we do not need to carry over a 55-factor variable into future models at all.

```r
library(randomForest)

regionprops = summary(train$region)
regionprops = regionprops/43436
tempregion = as.character(train$region)
for (i in 1:55){tempregion <- replace(tempregion, tempregion == names(regionprops[i]), regionprops[i])}
tempregion2 = as.character(test$region)
for (i in 1:55){tempregion2 <- replace(tempregion2, tempregion2 == names(regionprops[i]), regionprops[i

train2 <- train
train2$region = as.double(tempregion)
test2 <- test
test2$region = as.double(tempregion2)
```

Tuning Parameters

We note the performance of the Random Forest model is also dependent on the parameters of the model, which can be tuned:

- `ntree`: Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times
- `mtry`: Number of variables randomly sampled as candidates at each split. The default value for classification is $\sqrt{p}$ where p is number of variables in x
- `nodesize`: Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time).The default value for classification is 1.

```r
model15b.rf <- randomForest(lapsed ~ ., data=train2, ntree=500, do.trace=10)
```

```
## ntree      OOB      1      2
##    10:  32.41% 46.19% 23.42%
##    20:  30.26% 45.62% 20.24%
##    30:  29.62% 45.04% 19.55%
##    40:  29.01% 44.64% 18.81%
##    50:  28.61% 44.43% 18.29%
##    60:  28.52% 44.59% 18.03%
```

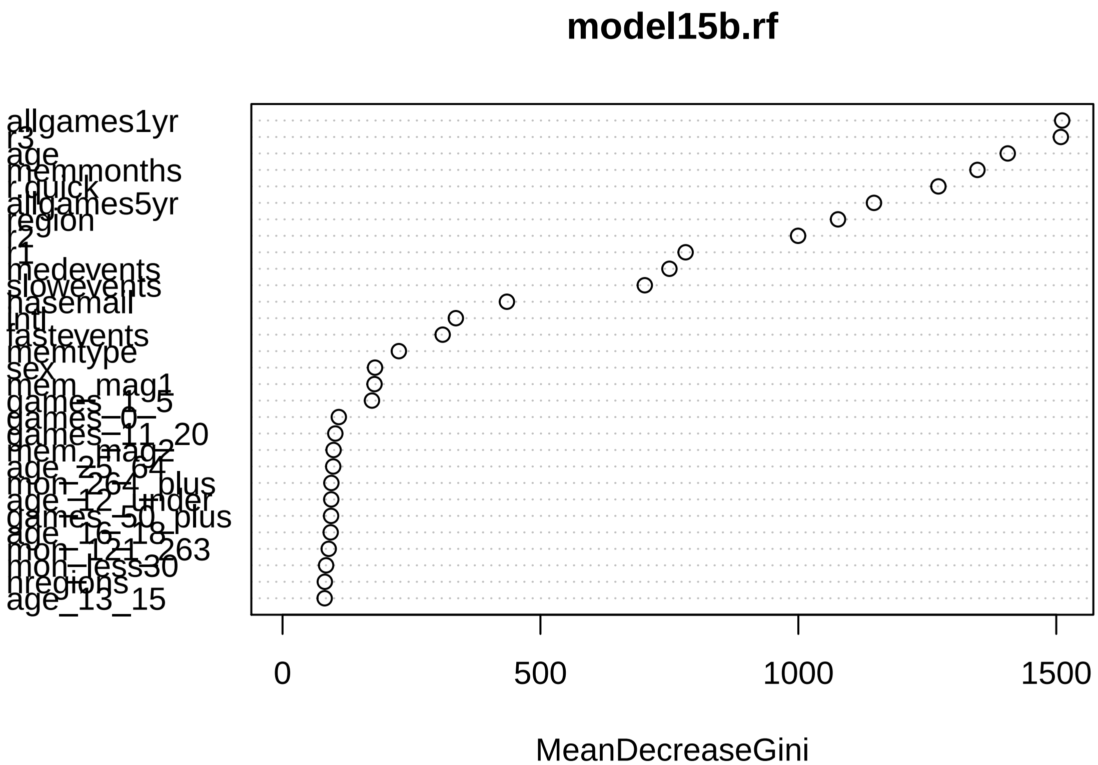
```
##    70:  28.34% 44.52% 17.79%
##    80:  28.33% 44.50% 17.78%
##    90:  28.25% 44.35% 17.75%
##   100:  28.21% 44.37% 17.67%
##   110:  28.20% 44.40% 17.63%
##   120:  28.13% 44.54% 17.43%
##   130:  28.04% 44.45% 17.34%
##   140:  28.00% 44.39% 17.30%
##   150:  27.93% 44.37% 17.20%
##   160:  27.97% 44.49% 17.20%
##   170:  27.89% 44.37% 17.14%
##   180:  27.83% 44.29% 17.10%
##   190:  27.81% 44.33% 17.03%
##   200:  27.72% 44.18% 16.98%
##   210:  27.74% 44.22% 16.99%
##   220:  27.72% 44.22% 16.96%
##   230:  27.74% 44.22% 17.00%
##   240:  27.71% 44.13% 16.99%
##   250:  27.72% 44.19% 16.97%
##   260:  27.75% 44.28% 16.96%
##   270:  27.68% 44.15% 16.93%
##   280:  27.68% 44.19% 16.90%
##   290:  27.66% 44.11% 16.93%
##   300:  27.68% 44.17% 16.92%
##   310:  27.68% 44.13% 16.94%
##   320:  27.70% 44.11% 17.00%
##   330:  27.68% 44.07% 16.99%
##   340:  27.69% 44.18% 16.94%
##   350:  27.65% 44.11% 16.90%
##   360:  27.64% 44.06% 16.93%
##   370:  27.68% 44.06% 16.99%
##   380:  27.74% 44.20% 17.00%
##   390:  27.76% 44.19% 17.04%
##   400:  27.71% 44.25% 16.92%
##   410:  27.66% 44.18% 16.88%
##   420:  27.67% 44.22% 16.88%
##   430:  27.72% 44.26% 16.92%
##   440:  27.73% 44.38% 16.86%
##   450:  27.76% 44.43% 16.88%
##   460:  27.75% 44.41% 16.89%
##   470:  27.76% 44.42% 16.89%
##   480:  27.73% 44.38% 16.87%
##   490:  27.74% 44.38% 16.88%
##   500:  27.75% 44.43% 16.87%
```

```r
print(model15b.rf)
```

```
##
## Call:
##  randomForest(formula = lapsed ~ ., data = train2, ntree = 500,      do.trace = 10)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
```

```
##         OOB estimate of  error rate: 27.75%
## Confusion matrix:
##      N     Y class.error
## N 9530  7619   0.4442825
## Y 4435 21852   0.1687146
```

```
varImpPlot(model15b.rf)
```



ntree: 500 mtry: sqrt(p) OOB error: 27.75% Kaggle score for `model15b`: 0.54805

The OOB estimate for the error rate was is approximately 27.7%. It is interesting to see that this model places `r3` as a very important variable.

```
model15c.rf <- randomForest(lapsed ~ ., data=train2, ntree=1500, do.trace=10)
print(model15c.rf)
varImpPlot(model15c.rf)
```

ntree: 1500 mtry: sqrt(p) OOB error: 27.71% Kaggle score for `model15c`: 0.54591

```
model15d.rf <- randomForest(lapsed ~ ., data=train2, ntree=3000, do.trace=10)
print(model15d.rf)
varImpPlot(model15d.rf)
```

ntree: 3000 mtry: sqrt(p) OOB error: 27.71% Kaggle score for `model15d`: 0.54510

The OOB error and performance on the test data appear to be plateauing. We experimented with tuning some other parameters:

```r
model15e.rf <- randomForest(lapsed ~ ., data=train2, ntree=3000, mtry=11, do.trace=10)
print(model15e.rf)
varImpPlot(model15e.rf)
```

ntree: 3000 mtry: 11 OOB error: 27.83% Kaggle score for `model15d`: 0.54365

Despite the slightly higher OOB error, we got better results for this model from the test set. If this was to be a final model, we could continue to tune it to further enhance performance.

```r
#Predict Output
predicted = predict(model15e.rf, test2, type="prob")
lapsed = predicted[,2]
# Write as submission file
colnames(Id) = "Id"
outputdf = cbind(Id, lapsed)
write.csv(outputdf, file="model15e.csv", row.names=FALSE)
```

```r
# saving the predictions
predicted_d = predict(model15d.rf, train2, type="prob")
values = predicted_d[,2]
write.csv(values, file="model15d_preds.csv", row.names=FALSE)

predicted_e = predict(model15e.rf, train2, type="prob")
values = predicted_e[,2]
write.csv(values, file="model15e_preds.csv", row.names=FALSE)
```

Classification Trees

Random Forests are an extension of Classification and Regression Trees (CART). We build some basic models for classification trees here.

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

ct1.fit = rpart(lapsed ~ ., data=train2, cp=0.0005, method="class", parms=list(split="information"))
```

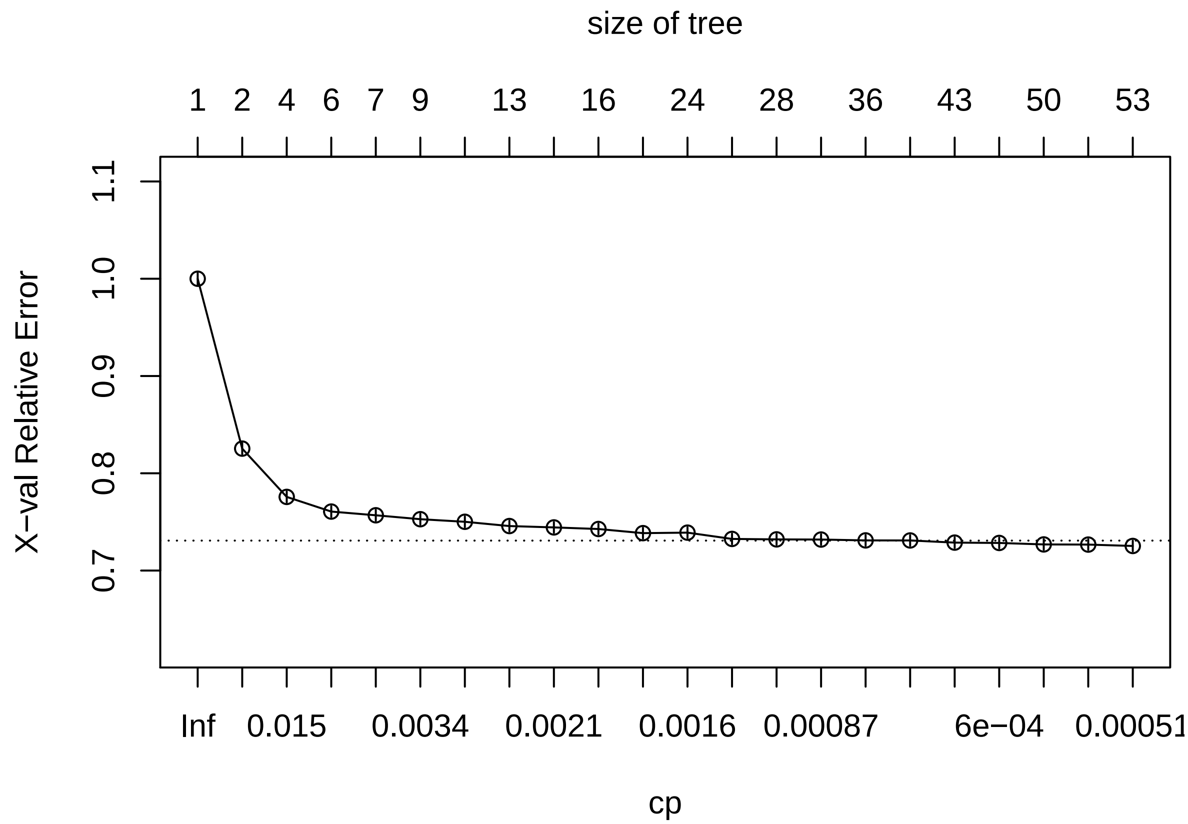
This is a very deep tree, so we proceeded with an exercise to prune it using the "1-SE rule".

```r
printcp(ct1.fit)
```

```
##
## Classification tree:
## rpart(formula = lapsed ~ ., data = train2, method = "class",
##     parms = list(split = "information"), cp = 5e-04)
##
## Variables actually used in tree construction:
##  [1] age         allgames1yr allgames5yr fastevents  hasemail
##  [6] intl        medevents   mem_mag1    memmonths   memtype
## [11] r.quick     r2          r3          region      sex
## [16] slowevents
```

```
##
## Root node error: 17149/43436 = 0.39481
##
## n= 43436
##
##            CP nsplit rel error  xerror      xstd
## 1  0.17715319      0   1.00000 1.00000 0.0059405
## 2  0.02612397      1   0.82285 0.82535 0.0056961
## 3  0.00854277      3   0.77060 0.77573 0.0056019
## 4  0.00501487      5   0.75351 0.76063 0.0055708
## 5  0.00379031      6   0.74850 0.75684 0.0055629
## 6  0.00309056      8   0.74092 0.75281 0.0055544
## 7  0.00219644      9   0.73783 0.75013 0.0055486
## 8  0.00212840     12   0.73124 0.74576 0.0055392
## 9  0.00198262     14   0.72698 0.74436 0.0055362
## 10 0.00165607     15   0.72500 0.74273 0.0055326
## 11 0.00157444     21   0.71474 0.73847 0.0055233
## 12 0.00154528     23   0.71159 0.73899 0.0055245
## 13 0.00104962     26   0.70605 0.73252 0.0055101
## 14 0.00093300     27   0.70500 0.73205 0.0055091
## 15 0.00081637     30   0.70220 0.73188 0.0055087
## 16 0.00075806     35   0.69812 0.73100 0.0055067
## 17 0.00067642     37   0.69660 0.73095 0.0055066
## 18 0.00064144     42   0.69322 0.72873 0.0055016
## 19 0.00056369     45   0.69129 0.72838 0.0055009
## 20 0.00055397     49   0.68873 0.72686 0.0054974
## 21 0.00052481     51   0.68762 0.72669 0.0054970
## 22 0.00050000     52   0.68710 0.72529 0.0054939
```

```r
plotcp(ct1.fit) # different results produced on each run
```



We use cp=0.0013:

```
ct2.fit = prune(ct1.fit, cp=0.0013)
print(ct2.fit)
```

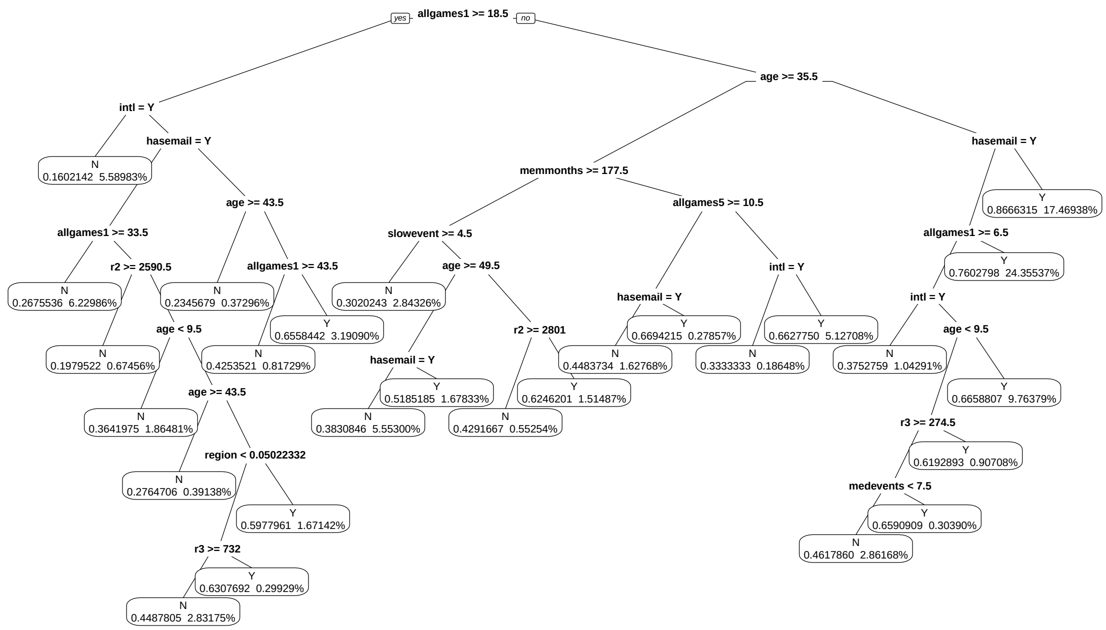
```
## n= 43436
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 43436 17149 Y (0.3948108 0.6051892)
##     2) allgames1yr>=18.5 10396  3679 N (0.6461139 0.3538861)
##       4) intl=Y 2428   389 N (0.8397858 0.1602142) *
##       5) intl=N 7968  3290 N (0.5870984 0.4129016)
##        10) hasemail=Y 6065  2192 N (0.6385820 0.3614180)
##          20) allgames1yr>=33.5 2706   724 N (0.7324464 0.2675536) *
##          21) allgames1yr< 33.5 3359  1468 N (0.5629652 0.4370348)
##            42) r2>=2590.5 293    58 N (0.8020478 0.1979522) *
##            43) r2< 2590.5 3066  1410 N (0.5401174 0.4598826)
##              86) age< 9.5 810   295 N (0.6358025 0.3641975) *
##              87) age>=9.5 2256  1115 N (0.5057624 0.4942376)
##               174) age>=43.5 170    47 N (0.7235294 0.2764706) *
##               175) age< 43.5 2086  1018 Y (0.4880153 0.5119847)
##                 350) region< 0.05022332 1360   634 N (0.5338235 0.4661765)
##                   700) r3>=732 1230   552 N (0.5512195 0.4487805) *
##                   701) r3< 732 130    48 Y (0.3692308 0.6307692) *
##                 351) region>=0.05022332 726   292 Y (0.4022039 0.5977961) *
##        11) hasemail=N 1903   805 Y (0.4230163 0.5769837)
```

```
##          22) age>=43.5 162    38 N (0.7654321 0.2345679) *
##          23) age< 43.5 1741   681 Y (0.3911545 0.6088455)
##            46) allgames1yr>=43.5 355   151 N (0.5746479 0.4253521) *
##            47) allgames1yr< 43.5 1386   477 Y (0.3441558 0.6558442) *
##     3) allgames1yr< 18.5 33040 10432 Y (0.3157385 0.6842615)
##       6) age>=35.5 8410  4090 N (0.5136742 0.4863258)
##        12) memmonths>=177.5 5274  2189 N (0.5849450 0.4150550)
##          24) slowevents>=4.5 1235   373 N (0.6979757 0.3020243) *
##          25) slowevents< 4.5 4039  1816 N (0.5503838 0.4496162)
##            50) age>=49.5 3141  1302 N (0.5854823 0.4145177)
##             100) hasemail=Y 2412   924 N (0.6169154 0.3830846) *
##             101) hasemail=N 729   351 Y (0.4814815 0.5185185) *
##            51) age< 49.5 898   384 Y (0.4276169 0.5723831)
##             102) r2>=2801 240   103 N (0.5708333 0.4291667) *
##             103) r2< 2801 658   247 Y (0.3753799 0.6246201) *
##        13) memmonths< 177.5 3136  1235 Y (0.3938138 0.6061862)
##          26) allgames5yr>=10.5 828   398 N (0.5193237 0.4806763)
##            52) hasemail=Y 707   317 N (0.5516266 0.4483734) *
##            53) hasemail=N 121    40 Y (0.3305785 0.6694215) *
##          27) allgames5yr< 10.5 2308   805 Y (0.3487868 0.6512132)
##            54) intl=Y 81    27 N (0.6666667 0.3333333) *
##            55) intl=N 2227   751 Y (0.3372250 0.6627750) *
##       7) age< 35.5 24630  6112 Y (0.2481527 0.7518473)
##        14) hasemail=Y 17042  5100 Y (0.2992607 0.7007393)
##          28) allgames1yr>=6.5 6463  2564 Y (0.3967198 0.6032802)
##            56) intl=Y 453   170 N (0.6247241 0.3752759) *
##            57) intl=N 6010  2281 Y (0.3795341 0.6204659)
##             114) age< 9.5 1769   864 Y (0.4884115 0.5115885)
##               228) r3>=274.5 1375   661 N (0.5192727 0.4807273)
##                 456) medevents< 7.5 1243   574 N (0.5382140 0.4617860) *
##                 457) medevents>=7.5 132    45 Y (0.3409091 0.6590909) *
##               229) r3< 274.5 394   150 Y (0.3807107 0.6192893) *
##             115) age>=9.5 4241  1417 Y (0.3341193 0.6658807) *
##          29) allgames1yr< 6.5 10579  2536 Y (0.2397202 0.7602798) *
##        15) hasemail=N 7588  1012 Y (0.1333685 0.8666315) *
```

Let's plot the decision tree:

```r
prp(ct2.fit, type=0, extra=106, digits=0,
    main="Pruned classification tree")
```

Pruned classification tree



Again, it appears that the predictors `allgames1yr`, `age`, and `memmonths` are important predictors. It also seems like `hasemail` is an important is an important predictor, which makes sense as it would be easier to communicate with these members (e.g. to encourage them to renew).

We now generate our predictions on the test set:

```
predicted = predict(ct2.fit, test2)
lapsed = predicted[,2]
# Write as submission file
colnames(Id) = "Id"
outputdf = cbind(Id, lapsed)
write.csv(outputdf, file="model_ct2.csv", row.names=FALSE)
```

Kaggle score on classification tree model: 0.57371

It appears that our Random Forest model performs better (and we are likely to trust those results more as the fits tends to be smoother than the CART models).

Gradient Boosting Models

We experimented with Gradient Boosting Models. In contrast to random forests, boosting methods are based on a different, constructive strategy: in boosting, we add new models to construct an ensemble sequentially. At each particular iteration, a new model is trained with respect to the error of the whole ensemble learnt so far.

```
# Reference: http://www.stat.missouri.edu/~speckman/stat461/boost.R

gbm1.fit <- gbm(lapsed ~ ., data=train_gbm, dist="adaboost", n.tree=400, cv.folds=10)
```

```r
# print(gbm1.fit)

# gbm.perf(gbm1.fit)
```

```r
predicted = predict(gbm1.fit, newdata=test2, n.tree=400, type="response")
lapsed = predicted
# Write as submission file
colnames(Id) = "Id"
outputdf = cbind(Id, lapsed)
write.csv(outputdf, file="model_gbm1.csv", row.names=FALSE)

predicted = predict(gbm1.fit, newdata=test2, n.tree=171, type="response")
lapsed = predicted
# Write as submission file
colnames(Id) = "Id"
outputdf = cbind(Id, lapsed)
write.csv(outputdf, file="model_gbm2.csv", row.names=FALSE)
```

Kaggle score for `model_gbm1` (400 trees): 0.54186 Kaggle score for `model_gbm2` (171 trees): 0.54271

```r
gbm3.fit <- gbm(lapsed ~ ., data=train_gbm, dist="adaboost", n.tree = 1000, shrinkage = 1, cv.folds=10)

# print(gbm3.fit)
```

```r
predicted = predict(gbm3.fit, newdata=test2, n.tree=1000, type="response")
lapsed = predicted
# Write as submission file
colnames(Id) = "Id"
outputdf = cbind(Id, lapsed)
write.csv(outputdf, file="model_gbm3.csv", row.names=FALSE)
```

Kaggle score for `model_gbm3` (1000 trees): 0.56713

We could continue tuning, but the performance seems on par with Random Forests.

```r
# saving the predictions
predicted_gbm1 = predict(gbm1.fit, train_gbm, n.tree=400, type="response")
write.csv(predicted_gbm1, file="modelgbm1_preds.csv", row.names=FALSE)

predicted_gbm2 = predict(gbm1.fit, train_gbm, n.tree=171, type="response")
write.csv(predicted_gbm2, file="modelgbm2_preds.csv", row.names=FALSE)
```

Neural Networks

We also experimented with neural network modeling in R.

```r
## one hot encoding of response variable
dummies <- dummyVars(lapsed~ ., data = train)
lapsed = train$lapsed
train2 = cbind(lapsed, data.frame(predict(dummies, newdata = train)))
```

```
## Warning in model.frame.default(Terms, newdata, na.action = na.action, xlev
## = object$lvls): variable 'lapsed' is not a factor
```

```r
temp2 = cbind(Id, test)
dummies2 <- dummyVars(Id~ ., data = temp2)
test2 = data.frame(predict(dummies2, newdata = temp2))
```

```r
model <- train(lapsed~ ., data=train2, method='nnet', trace = TRUE) # train

prediction <- predict(model, newdata=test2, type="prob")
head(prediction)

lapsed = prediction[,2]

# Write as submission file
outputdf = cbind(Id, lapsed)
# summary(outputdf)
write.csv(outputdf, file="nn1.csv", row.names=FALSE)
```

Kaggle score for `nn1`: 0.55629

```r
# re-run with cross-validation of tuning parameters

model2 <- train(lapsed~ ., data=train2, method='nnet', trace = FALSE,
                maxit=500, MaxNWts=2000,
                tuneGrid=expand.grid(.size=c(1, 3, 5, 8, 10),
                                     .decay=c(0, 0.01, 0.001, 0.1))) # train
model2

prediction3 <- predict(model2, newdata=test2, type="prob")

lapsed = prediction3[,2]

# Write as submission file
outputdf = cbind(Id, lapsed)
# summary(outputdf)
write.csv(outputdf, file="nn3.csv", row.names=FALSE)

predict_train <- predict(model2, data=train2, type="prob")
predicts <- predict_train[,2]
write.csv(predicts, file="train_predicts_for_nn3.csv", row.names=FALSE)
```

Kaggle score for `nn3` (tuned): 0.53759

```
Neural Network

43436 samples
146 predictor
2 classes: 'N', 'Y'

No pre-processing
Resampling: Bootstrapped (25 reps)
```

```
Summary of sample sizes: 43436, 43436, 43436, 43436, 43436, 43436, ...
Resampling results across tuning parameters:

  size  decay  Accuracy   Kappa      Accuracy SD  Kappa SD
1     0.000  0.7097156  0.3650203  0.022207564  0.081307501
1     0.001  0.7150336  0.3840959  0.004288479  0.014392487
1     0.010  0.7166744  0.3859578  0.002715953  0.006214260
1     0.100  0.7167520  0.3863466  0.002721783  0.006218872
3     0.000  0.7096349  0.3761473  0.004901399  0.011774349
3     0.001  0.7143386  0.3840064  0.004485911  0.015023195
3     0.010  0.7168440  0.3902266  0.004169898  0.009380195
3     0.100  0.7210172  0.3983600  0.002722972  0.005459505
5     0.000  0.7011880  0.3508344  0.021181565  0.075503129
5     0.001  0.7092429  0.3737054  0.004685641  0.014040129
5     0.010  0.7141405  0.3842377  0.004613420  0.012483010
5     0.100  0.7175368  0.3920005  0.003232986  0.007437774
8     0.000  0.7005524  0.3561363  0.006241760  0.013943082
8     0.001  0.7045692  0.3650478  0.004720755  0.014013950
8     0.010  0.7080148  0.3733012  0.003392562  0.009111642
8     0.100  0.7118594  0.3809774  0.003468838  0.007833833
10     0.000  0.6958247  0.3515721  0.005768513  0.010624254
10     0.001  0.7021615  0.3643075  0.004164125  0.009595991
10     0.010  0.7065354  0.3714839  0.004554720  0.008850249
10     0.100  0.7093673  0.3769377  0.003046364  0.006976055

Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were size = 3 and decay = 0.1.
```

```r
model3 <- train(lapsed~ ., data=train2, method='nnet', trace = FALSE,
                maxit=2000, MaxNWts=5000,
                tuneGrid=expand.grid(.size=c(1, 2, 3, 4, 5),
                                     .decay=c(0.01, 0.1))) # train
model3

prediction3b <- predict(model3, newdata=test2, type="prob")
lapsed = prediction3b[,2]
# Write as submission file
outputdf = cbind(Id, lapsed)
# summary(outputdf)
write.csv(outputdf, file="nn4.csv", row.names=FALSE)
predict_train <- predict(model3, data=train2, type="prob")
predicts <- predict_train[,2]
write.csv(predicts, file="train_predicts_for_nn3b.csv", row.names=FALSE)

model4 <- train(lapsed~ ., data=train2, method='nnet', trace = FALSE,
                maxit=2000, MaxNWts=5000,
                tuneGrid=expand.grid(.size=c(2, 3, 4),
                                     .decay=c(1.0))) # train

prediction5 <- predict(model4, newdata=test2, type="prob")
lapsed = prediction5[,2]
# Write as submission file
outputdf = cbind(Id, lapsed)
# summary(outputdf)
```

```r
write.csv(outputdf, file="nn5.csv", row.names=FALSE)
predict_train <- predict(model4, data=train2, type="prob")
predicts <- predict_train[,2]
write.csv(predicts, file="train_predicts_for_nn5.csv", row.names=FALSE)
```

Kaggle score for `nn4` (further tuned): 0.53730 Kaggle score for `nn5` (tuned decay): 0.53944

```r
model5 <- train(lapsed~ ., data=train2, method='nnet', trace = FALSE,
                maxit=5000, MaxNWts=5000,
                tuneGrid=expand.grid(.size=c(2, 3, 4),
                                     .decay=c(1.0))) # train

prediction5b <- predict(model5, newdata=test2, type="prob")
lapsed = prediction5b[,2]
# Write as submission file
outputdf = cbind(Id, lapsed)
# summary(outputdf)
write.csv(outputdf, file="nn5b.csv", row.names=FALSE)
```

Kaggle score for `nn5b` (tuned decay): 0.54027 We are beginning to overfit the training set.