Machine Learning Project

Nick Lukianoff

December 26, 2015

This project involves reading in a large data set, reducing it to an appropriate size, validate it, perform machine learning on it, and test the results.

• How I built my model

I chose a boosted tree model using gbm. This model provides excellent predictions with a minimum of processing. It does this by invoking efficient algorithms.

To begin with, I wanted to reduce the data set. This would speed up processing and reduce file sizes. The initial data set contained 3,139,520 items in 19,622 rows and 160 columns. I first removed any columns that contained NA values. This removed 67 columns, leaving me with 1,824,846 data items. This is a 41.88% reduction !

I next removed any non-numeric columns. These were columns 1-7, which contained identifier information that wasn't needed to data analysis. Removing these columns yielded a 7.53% reduction.

Next, I removed any columns that had very little variance. The fact that the values in these columns were so similar meant that they acted more as constants then as variables. 33 columns were thus removed, yielding a 38.37% data set reduction.

Last, I removed any rows that contained outliers. Outliers give data a large variance, and typically don't significantly impact the final result. I checked the standard deviation of each column, and removed any row that contained a value of greater than 2 standard deviations from the mean. This reduced my data set by 7.2%.

The end result of the data cleaning was a data set consisting of 18,210 rows and 53 colums. This means that I now have 965,130 data points, an incredible 69.26% reduction from the original data set !

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.2.3
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(stats)
library(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
```

```
set.seed(1234)

training <- read.csv("pml-training.csv")
dim(training)
```

```
## [1] 19622   160
```

```
#remove columns that have NA values, since these may not be as
applicable
training.nona <- training[,colSums(is.na(training)) == 0]

#remove columns that don't have data that we'll be correlating, namely
columns 1:7
cc <- ncol(training.nona)
training.noid <- training.nona[,8:cc]

#remove columns that have little variance, since they're almost a
constant
nzv <- nearZeroVar(training.noid)
training.nzv <- training.noid[,-nzv]

#remove rows that contain outliers, defined as being more than 2
standard deviations from the mean
cc <- ncol(training.nzv)
for(i in 1:(cc-1)) {
       training.norow <- training.nzv[!(abs(training.nzv[,i] -
mean(training.nzv[,i]))/sd(training.nzv[,i])) > 2,]
}

dim(training.norow)
```

```
## [1] 18210    53
```

• How I used cross validation

```
   Initially I wanted to use 10 k-folds of data, with 10 repititions
of each.  This would provide a robust validation set.  However, this
overloaded my computer, maxing out the memory, and using up all
available disk space, causing the program to crash.  I decided to use a
more realistic 5 k-fold model, with 5 repititions.  I also split the
data 70-30 into training and testing sets.
```

```
#Split the data into training and validation sets
train1 <- createDataPartition(y=training.norow$classe, p=0.7,
list=FALSE)
training.set <- training.norow[train1,]
training.test <- training.norow[-train1,]

fitControl <- trainControl(
        method = "repeatedcv",
        number = 5,
        repeats = 5)
```

```
    The results of this cross-validation were passed to the bgm
training call.
```

```
gbmFit1 <- train(training.set$classe ~ ., data = training.set,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)

## Loading required package: plyr
```

• What I think the expected out of sample error is

```
    The expected sample error is 3.48%.  This number was obtained by
comparing the predicted value in the test training set to the actual
value in the test training set.  Out of 5,460 values in the training
test set, 5,270 were accurate.  I was hoping for an error rate of less
than 5%, and this was achieved.
```

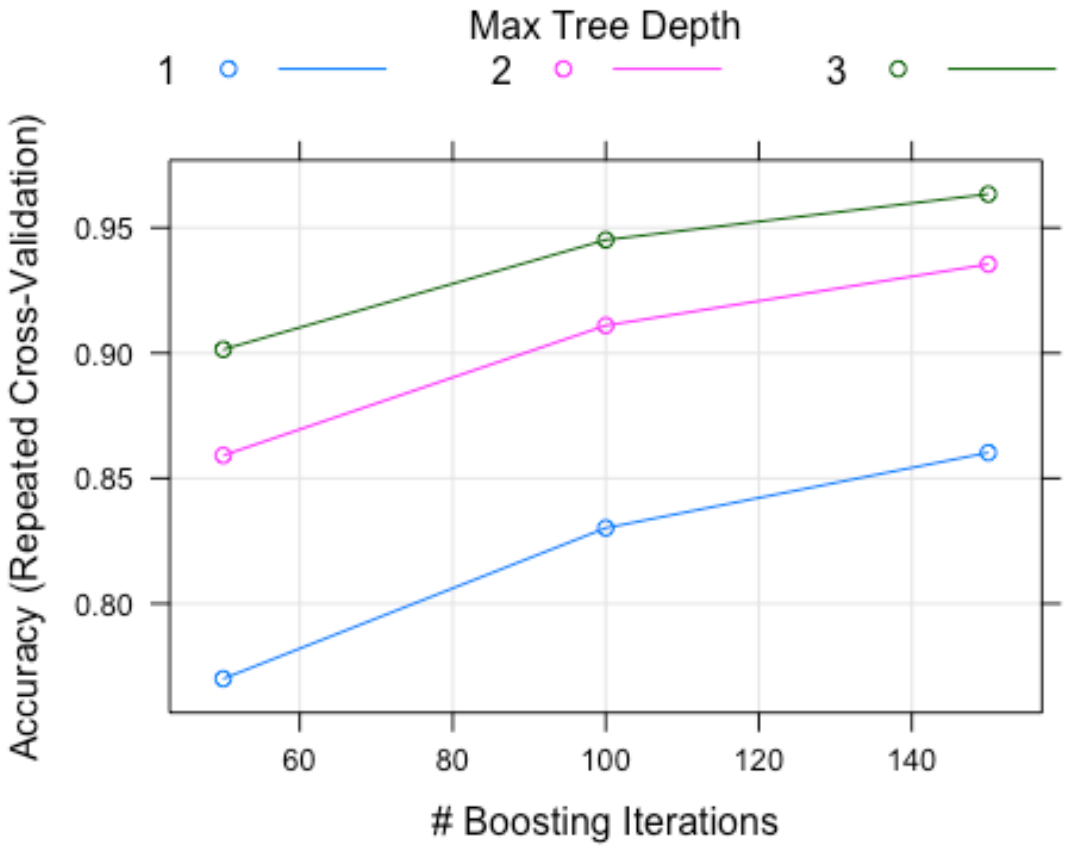
```
predicted.values <- predict(gbmFit1, training.test)
testme <- data.frame(predicted.values,training.test$classe)
testme$delta <- ifelse(testme[,1] == testme[,2], 1, 0)
error.num <- (nrow(testme)-sum(testme$delta))/nrow(testme)*100
print(paste("Error rate:", format(error.num,digits=2,nsmall=2), "%"))

## [1] "Error rate: 3.48 %"
```

• Why I made the choices that I did.

```
    I wanted to use an accurate prediction method that didn't involve
randome forest.  I suspect that most people would choose randome
forest, and so I wanted to try a different method.  I have a weak
computer, so I wanted an algorithm that wasn't too processor-intensive.
This means that a boosted algorithm was my best bet.  It produces
accurate results with minimal processing.  I chose an efficient model,
the boosted tree model. The final values used for the model were
n.trees = 150, interaction.depth = 3, shrinkage = 0.1 and
n.minobsinnode = 10.
```

• Results

This plot shows the results of training and cross-validation. It shows that the best results are obtained with a tree depth of 3, and using 150 iterations.



A table of training results shows this in tabular format. The very last line shows the highest accuracy rate of 96.34%, and the lowest standard deviation of 0.2%. This is the line that uses 150 iterations of a 3-deep tree. It is the value that is used to generate the preditions.

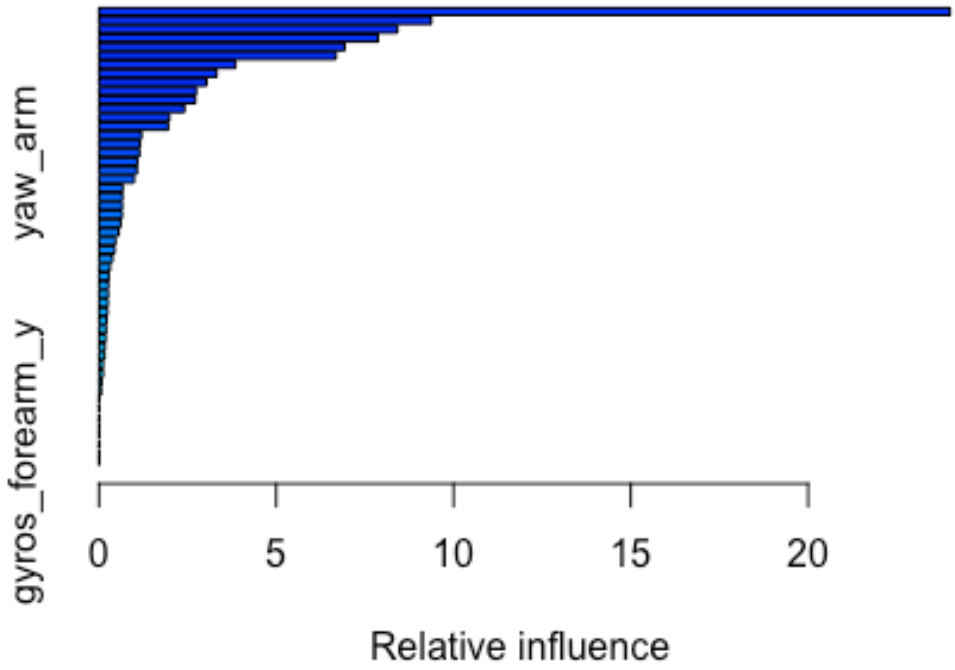
```
gbmFit1$results
```

```
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy
Kappa
## 1       0.1                 1             10      50 0.7700234
0.7085096
## 4       0.1                 2             10      50 0.8591537
0.8217554
## 7       0.1                 3             10      50 0.9014440
0.8753550
## 2       0.1                 1             10     100 0.8301813
0.7850910
## 5       0.1                 2             10     100 0.9109811
```

```
0.8874531
## 8       0.1                 3             10     100 0.9451610
0.9306770
## 3       0.1                 1             10     150 0.8602361
0.8232841
## 6       0.1                 2             10     150 0.9354670
0.9184179
## 9       0.1                 3             10     150 0.9634198
0.9537701
##    AccuracySD     KappaSD
## 1 0.007933415 0.010167018
## 4 0.006681726 0.008514649
## 7 0.005471947 0.006939525
## 2 0.007979429 0.010146566
## 5 0.005464202 0.006915869
## 8 0.003551447 0.004497536
## 3 0.006281837 0.007955230
## 6 0.004876600 0.006176061
## 9 0.002774331 0.003509710

gbmFit1$bestTune

##   n.trees interaction.depth shrinkage n.minobsinnode
## 9     150                 3       0.1             10
```

• Observations

```
    I created a table of observations and their relative weights.  I
found out that the roll_belt variable accounted for a full 24% of the
total prediction of the score !  This is by far the most significant
variable.  The top 4 variables accounted for 50% of the scoring
weights, and the top 10 variables accounted for 75%.  The top 20
accounted for 90%.  The bottom 8 didn't contribute anything, and in
hindsight, could have been removed as part of the data cleaning
process.  The bottom 25 variables each contributed less than 1% to the
solution.  This means that half of all the values contributed only 10%.
The following graph and table illustrate this.


summary(gbmFit1)
```




```
##                                     var     rel.inf
## roll_belt                     roll_belt 24.00250352
## pitch_forearm             pitch_forearm  9.37510203
## yaw_belt                       yaw_belt  8.42180875
## magnet_dumbbell_y       magnet_dumbbell_y  7.86905463
## roll_forearm                 roll_forearm  6.92771394
## magnet_dumbbell_z       magnet_dumbbell_z  6.68622049
## magnet_belt_z               magnet_belt_z  3.85275202
## pitch_belt                     pitch_belt  3.31929838
## gyros_belt_z                 gyros_belt_z  3.03444405
## accel_forearm_x           accel_forearm_x  2.74487134
## roll_dumbbell               roll_dumbbell  2.72050280
## accel_dumbbell_y         accel_dumbbell_y  2.41242956
## magnet_arm_x                 magnet_arm_x  1.98722534
## gyros_dumbbell_y         gyros_dumbbell_y  1.95108428
## magnet_arm_z                 magnet_arm_z  1.19993496
## accel_dumbbell_z         accel_dumbbell_z  1.16102673
## accel_dumbbell_x         accel_dumbbell_x  1.14311973
## yaw_arm                           yaw_arm  1.08126128
## magnet_belt_y               magnet_belt_y  1.07053851
## magnet_dumbbell_x       magnet_dumbbell_x  0.99193801
## accel_forearm_z           accel_forearm_z  0.66996304
## yaw_forearm                   yaw_forearm  0.66351977
```

```
## magnet_belt_x               magnet_belt_x  0.65758025
## accel_belt_z                 accel_belt_z  0.65485908
## magnet_forearm_x         magnet_forearm_x  0.61100339
## magnet_arm_y                 magnet_arm_y  0.55884550
## gyros_dumbbell_x         gyros_dumbbell_x  0.46664756
## pitch_dumbbell             pitch_dumbbell  0.44716015
## magnet_forearm_y         magnet_forearm_y  0.38395774
## total_accel_arm           total_accel_arm  0.31325743
## accel_arm_x                   accel_arm_x  0.27568021
## pitch_arm                       pitch_arm  0.26391895
## gyros_arm_x                   gyros_arm_x  0.25942500
## accel_forearm_y           accel_forearm_y  0.25313253
## total_accel_dumbbell total_accel_dumbbell  0.24351711
## gyros_belt_y                 gyros_belt_y  0.21789369
## total_accel_forearm   total_accel_forearm  0.20165930
## accel_belt_y                 accel_belt_y  0.19156735
## accel_arm_z                   accel_arm_z  0.17171091
## gyros_forearm_z           gyros_forearm_z  0.16051758
## magnet_forearm_z         magnet_forearm_z  0.12184070
## roll_arm                         roll_arm  0.11975370
## gyros_dumbbell_z         gyros_dumbbell_z  0.07547867
## gyros_arm_y                   gyros_arm_y  0.06428004
## total_accel_belt         total_accel_belt  0.00000000
## gyros_belt_x                 gyros_belt_x  0.00000000
## accel_belt_x                 accel_belt_x  0.00000000
## gyros_arm_z                   gyros_arm_z  0.00000000
## accel_arm_y                   accel_arm_y  0.00000000
## yaw_dumbbell                 yaw_dumbbell  0.00000000
## gyros_forearm_x           gyros_forearm_x  0.00000000
## gyros_forearm_y           gyros_forearm_y  0.00000000
```