Predicting Incorrect Exercises

Carlos Mercado

July 11, 2017

The Problem

Using accelerometer data from 4 places during an exercise (arm, forearm,belt, and the dumbell itself) across six participants, can the **quality** of the exercise be predicted?

This is the “classe” variable in the Weight Lifting Exercises Dataset. Thank you to,

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human ’13) . Stuttgart, Germany: ACM SIGCHI, 2013.

For creating the dataset under creative commons.

Explaining my Model Choices

Seeing the dimensions of the data I was immediately wary of overfitting, but upon examining pieces of the data and cleaning it of unusable features I narrowed it down to 53 features (removing the NA, character, time components and user names). I did this under the assumption that the time dependencies of the accelerometer would be negligible in predicting grossly incorrect exercises (as explained by the experimenters, a professional was there to monitor the exercise and ensure safe, but dramatic incorrect motions).

I started with a computationally friendly Linear Discriminant Analysis to benchmark my in-sample error (70%). From there, I used parallel computing (as suggested by the discussion boards) to create a more accurate Random Forest Model. Upon seeing that it had perfect in-sample accuracy, I decided against creating an ensemble model and went straight to testing the RF model on a validation set I had already separated. At 99.34% accuracy, it would succeed in getting all of the 20 final tests correct around 90% of the time.

Downloading and Cleaning the data

```
library(ggplot2)
library(caret)
```

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

```
trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"

#the test set will be used at the end
testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

download.file(trainURL, destfile = "./trainWLED.csv")

weightlift <- read.csv("./trainWLED.csv",stringsAsFactors = FALSE, na.strings = " ")
```

When downloading the data a large amount of NAs and character classes form (when stringsAsFactors = FALSE). So to check if those values mean anything, I've found the columns that become characters and I'll identify if they're useful.

```
#160 is classe predictor variable
weightlift[,160] <- as.factor(weightlift[,160])

charcolumnlist = NULL
for(i in 1:159){
if(class(weightlift[,i]) == "character"){
        charcolumnlist <- c(charcolumnlist,i)
        }
}
```

re-downloading weightlift and using stringsAsFactors = TRUE, I'll select only these rows and see if they are usable.

```
possunusable <- read.csv("./trainWLED.csv", stringsAsFactors = TRUE, na.strings = " ")
poss2 <- possunusable[,charcolumnlist]
```

As expected, besides the username, cvtd_timestamp, and new_window columns, the other 100 columns are NA over 99% of the time. I'll remove those columns.

```
rmcolumnlist <- charcolumnlist[-c(1:3)] #the first, second, and third values

weightsdata <- weightlift[,-rmcolumnlist]
weightsdata[,60] <- as.factor(weightsdata[,60])

rm(poss2, possunusable) #reduce clutter
```

Building a Model

To start, I want to make an anonymous model (remove names) that doesn't use time data as a benchmark for other models to beat. I will ensemble the models as well, depending on their accuracies.

```
weightstrain <- weightsdata[8:60] #no windows, names, or time stamps

set.seed(4)

inTrain <- createDataPartition(weightstrain$classe, p = .7, list = FALSE)

training <- weightstrain[inTrain,]
testing <- weightstrain[-inTrain,]

LDAmodel <- train(classe ~., method = "lda", data = training)
```

```
## Loading required package: MASS
```

```
LDApred <- predict(LDAmodel, newdata = training)

confusionMatrix(LDApred,training$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3199  397  229  132   97
##          B   79 1713  226   89  432
##          C  300  325 1605  278  231
##          D  312  108  279 1668  249
##          E   16  115   57   85 1516
##
## Overall Statistics
##
##                Accuracy : 0.7062
##                  95% CI : (0.6985, 0.7138)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6283
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8190   0.6445   0.6699   0.7407   0.6004
## Specificity            0.9130   0.9254   0.9000   0.9175   0.9757
## Pos Pred Value         0.7891   0.6747   0.5860   0.6376   0.8474
## Neg Pred Value         0.9270   0.9156   0.9281   0.9475   0.9156
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2329   0.1247   0.1168   0.1214   0.1104
## Detection Prevalence   0.2951   0.1848   0.1994   0.1904   0.1302
## Balanced Accuracy      0.8660   0.7850   0.7849   0.8291   0.7880
```

70% accuracy is low so I'll test an RF model and compare. Using the parallel and doParallel packages as suggested by **https://github.com/lgreski/datasciencectacontent/blob/master/markdown/pml-randomForestPerformance.md** (https://github.com/lgreski/datasciencectacontent/blob/master/markdown/pml-randomForestPerformance.md**)

Note: *Cross Validation* is included in the trainControls for the RF Model.

```
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
cluster <- makeCluster(detectCores() - 1)  # using code from link
registerDoParallel(cluster)                # using code from link

tcControls <- trainControl(method = "cv", number = 10, allowParallel = TRUE)

RFmodel <- train(classe ~ ., method = "rf", data = training, trControl = tcControls)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
stopCluster(cluster)  # using code from link
registerDoSEQ()       # using code from link

RFpred <- predict(RFmodel, newdata = training)
confusionMatrix(RFpred, training$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3906    0    0    0    0
##          B    0 2658    0    0    0
##          C    0    0 2396    0    0
##          D    0    0    0 2252    0
##          E    0    0    0    0 2525
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Using parallel processing to reduce my runtime to about 15 minutes, the confusionMatrix shows 100% accuracy - possibly overfitting or the worst case scenario, a classe proxy is still inside the dataset.

I'll use the test set to check the results.

```
RFtestpred <- predict(RFmodel, newdata = testing)

confusionMatrix(RFtestpred, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    6    0    0    0
##          B    2 1133    7    0    0
##          C    1    0 1019   20    1
##          D    0    0    0  944    1
##          E    0    0    0    0 1080
##
## Overall Statistics
##
##                Accuracy : 0.9935
##                  95% CI : (0.9911, 0.9954)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9918
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9947   0.9932   0.9793   0.9982
## Specificity            0.9986   0.9981   0.9955   0.9998   1.0000
## Pos Pred Value         0.9964   0.9921   0.9789   0.9989   1.0000
## Neg Pred Value         0.9993   0.9987   0.9986   0.9960   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1925   0.1732   0.1604   0.1835
## Detection Prevalence   0.2850   0.1941   0.1769   0.1606   0.1835
## Balanced Accuracy      0.9984   0.9964   0.9943   0.9895   0.9991
```

Out of Sample Error and Cross Validation

99.34% Accuracy on the testing set~ Thus an out of sample error of 1-Accuracy is estimated to be .66%. Although the 95% confidence interveral for the accuracy is (.991,.9953) so the out of sample error is between .66 with a 95% confidence interval of (.47%,.9%).

```
RFmodel
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12362, 12362, 12364, 12363, 12363, 12365, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9921384  0.9900545
##   27    0.9919198  0.9897786
##   52    0.9836203  0.9792798
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

10 - fold cross validation was used in the random forest model.