PML - Weightlifting Quality Prediction Project

Tarzan11

November 22, 2015

Introduction

The goal of this project is to predict the manner in which subjects did the exercise (“classe” variable in the training set). The source of the data is an experiment in which 6 subjects performed weightlifting actions while sensors recorded their activity. What was added was a subjective ‘class’, indicating how well the exercise was done. This opened the door to an attempt to use data science to predict exercise quality. This project attempts to build a predictive model to predict the quality (class) based on other variable.

Source of the study and the data

The data and the author’s paper can be found at http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har). The paper shed some light on how the authors had derived features based on the raw sensor data. This prompted a close look at the data and ultimately resulted in the choice to eliminate a large number of the derived columns for the purposes of this project.

Process

The approach to build the model was as follows in order: - Data exploration and preparation - Model selection - Model training - Model evaluation and interpretation - Model application

Data exploration

Looking closely at the data it became apparent quickly that: 1) The training.csv data is a mixture of raw and derived fields; 2) The formulas for calculating derived fields are not available; 3) The data is time series but the authors had already calculated time windows and marked them; 4) Derived variables where available only on one of the rows in each window.

Model selection

I chose to build a Random Forest model early in the process, for a few reasons: 1) I thought it would be cool since I had not worked with Random Forest models before 2) It made life a lot easier by takin away the need for a lot of data / feature manipulation 3) It worked well with eliminating the derived variable from the data since it takes care of feature interactions

Data load and manipulation

Once loaded, I built a data set that only had the needed features and changed the data types to their appropriate types (i.e. classe had to become a factor variable for the classification problem).

```
library(caret)
```

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

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Load main training data and read it in
trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
trainfile <- "train.csv"
download.file(trainURL,trainfile,method="curl")
training <- read.csv("train.csv",header=TRUE, na.strings=NA,stringsAsFactors=FALSE)

## Data prep for modeling, getting rid of unused columns such as time stamps and
## correcting data types
training <- training[,-c(1,3,4,5,6,7)]
training$classe <- as.factor(training$classe)
training$user_name <- as.factor(training$user_name)

## Selecting columns to build model on
training <- select(training,
                   - c(starts_with("kurt"),starts_with("skew"),starts_with("max"),
                       starts_with("min"),
                       starts_with("ampli"),starts_with("var"),starts_with("avg"),
                       starts_with("stddev"), starts_with("new_w")))
```

Validation / cross validation

We know that the Random Forest algorithm in caret package uses boot strapping by default. This means cross validation is occuring during model train automatically. However, in order to measure and estimate the model accuracy, data was split into 2, training and test, a 70%, 30% split. Once the model was trained, it was

evaluated on the test set.

A key question here was whether to split the data using time windowing, this way, the splits would be more coherent in terms of time. Since the activities had time associated with them, this makes sense. In predicting how well an action is being performed, we want to understand not only what is being sensed in that moment but also what happened before and what is coming next. I decided this would complicate this a bit much for this project, but if we were doing this for real it would become a large part of the analysis.

```
## Split data for training and validation
inTrain <- createDataPartition(y=training$classe, p=0.7, list=FALSE)
trainx <-tbl_df(training[inTrain,])
testx <-tbl_df(training[-inTrain,])
```

Model training

Taking defaults for caret random forest, the model was trained. It took hours for each attempt at training to complete. I made a note to research how one can predict the computation time bofore even starting the train process.

Based on the variable importance after building a model with all the remaining variables, I saw that 8 of the variables had the highest level of importance and retrained the model just with them

```
## Train model
## rfmodel2 <- train(classe ~ .,method="rf", data=trainx, prox=TRUE)

rfmodel3 <- train(classe ~ roll_belt +pitch_forearm  +yaw_belt + pitch_belt + magnet_
dumbbell_y + magnet_dumbbell_z +
                        roll_forearm  + accel_dumbbell_y, method="rf", data=trainx,
prox=TRUE )
```

```
## 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:dplyr':
##
##     combine
```

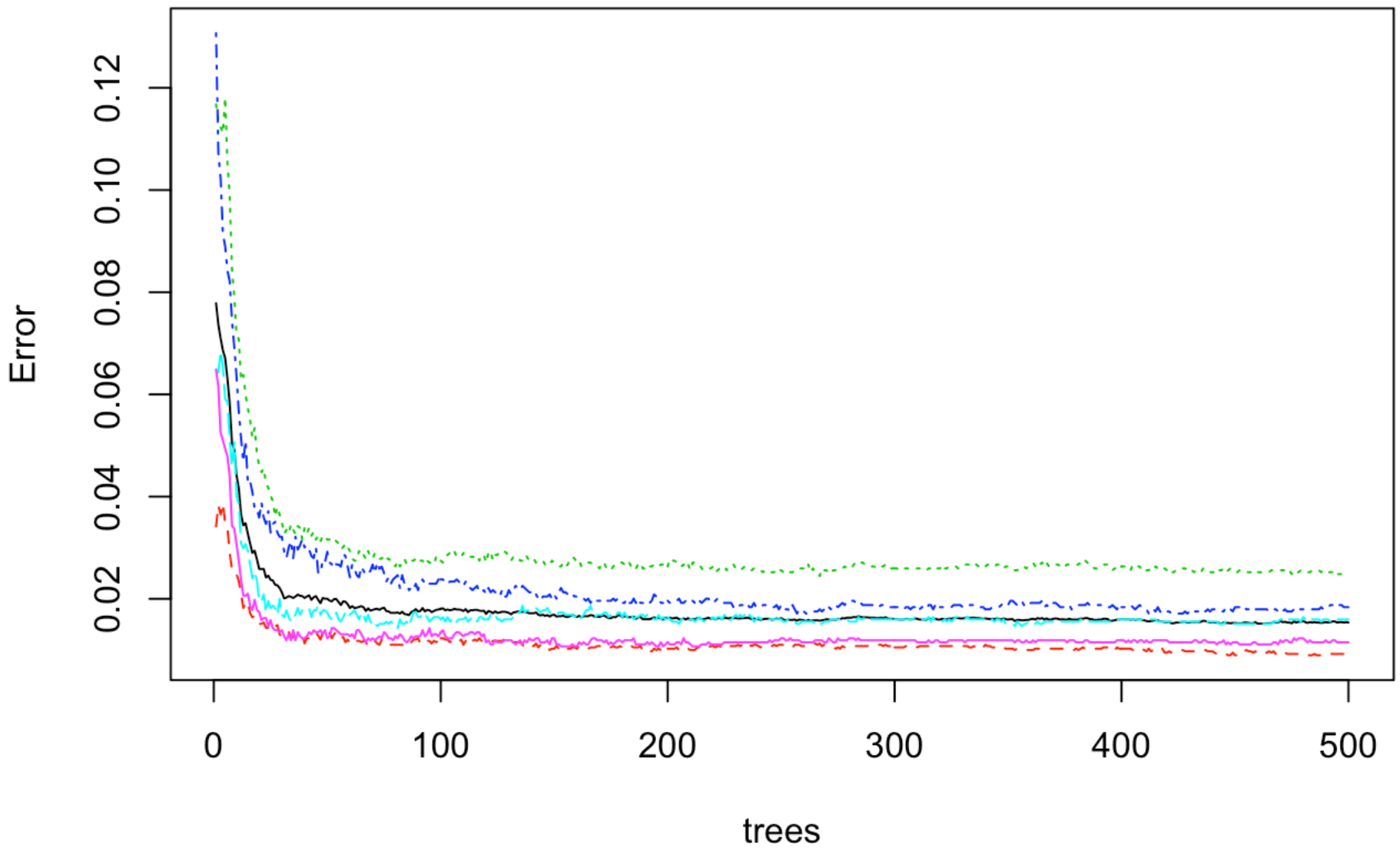
```
rfmodel3
```

```
## Random Forest
##
## 13737 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##   2     0.9781733  0.9724135  0.002052407  0.002591219
##   5     0.9765724  0.9703901  0.002389132  0.003010405
##   8     0.9698738  0.9619257  0.005001187  0.006308867
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
## Print the variables in order of imporatnce
imp <- importance(rfmodel3$finalModel)
round(imp[order(imp,decreasing=TRUE),],0)
```

```
##          roll_belt           yaw_belt        pitch_belt     pitch_forearm
##               2000               1589               1327               1266
## magnet_dumbbell_z magnet_dumbbell_y      roll_forearm  accel_dumbbell_y
##               1259               1215               1070                995
```

```
plot(rfmodel3$finalModel)
```



Model evaluation - estimating out of sample error rate

The prediction on the out of sample population is very good, making me more comfortable about the overfitting.

```
## Applying the model on the the test split
evalset <-   predict(rfmodel3,testx)
## Ealuating the results (comparing to actuals)
evalset.correct <- evalset == testx$classe
print(paste("Percent Correct Predictions on Evaluation Set =",round(mean(evalset.corr
ect)*100,2),"%"))
```

```
## [1] "Percent Correct Predictions on Evaluation Set = 98.59 %"
```

```
confusionMatrix(evalset,testx$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1658   11    0    0    0
##          B    6 1107    5    0    4
##          C    7   12 1019   11    7
##          D    3    8    2  953    6
##          E    0    1    0    0 1065
##
## Overall Statistics
##
##                Accuracy : 0.9859
##                  95% CI : (0.9825, 0.9888)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9822
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9904   0.9719   0.9932   0.9886   0.9843
## Specificity            0.9974   0.9968   0.9924   0.9961   0.9998
## Pos Pred Value         0.9934   0.9866   0.9650   0.9805   0.9991
## Neg Pred Value         0.9962   0.9933   0.9986   0.9978   0.9965
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2817   0.1881   0.1732   0.1619   0.1810
## Detection Prevalence   0.2836   0.1907   0.1794   0.1652   0.1811
## Balanced Accuracy      0.9939   0.9844   0.9928   0.9924   0.9920
```

Applying the model to the test cases

I was able to get 100% of the answers right based on the model.