Prediction Assignment Writeup

Channabasavaraj

January 28, 2016

Background and Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks.

One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, we will use data recorded from accelerometers on the belt, forearm, arm, and dumbbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

More information is available from the website http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

The goal of this project is to predict the manner in which the participants did the exercise. This is the classe variable of the training set, which classifies the correct and incorrect outcomes into A, B, C, D, and E categories. This report describes how the model for the project was built, its cross validation, expected out of sample error calculation, and the choices made. It was used successfully to accurately predict all 20 different test cases on the Coursera website.

This document is the write-up submission for the course Practical Machine Learning by Jeff Leek, PhD, Professor at Johns Hopkins University, Bloomberg School of Public Health. This 4-week course was offered on Coursera in June 2015, and is part of Johns Hopkins Data Science Specialization.

Data Description The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har

We first download the data from the links referenced above to our computer and upload the files into R (using RStudio), interpreting the miscellaneous NA, #DIV/0! and empty fields as NA:

```
setwd("C:/Coursera/Practical Machine Learning/")

training <- read.csv("pml-training.csv", na.strings = c("NA", "#DIV/0!", ""))

testing  <- read.csv("pml-testing.csv",  na.strings = c("NA", "#DIV/0!", ""))
```

We take a quick look at the data and particularly at classe which is the variable we need to predict:

```
str(training, list.len=15)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484
```

```
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_picth_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ kurtosis_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_belt      : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

```
table(training$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

```
prop.table(table(training$user_name, training$classe), 1)
```

```
##
##                   A         B         C         D         E
##   adelmo   0.2993320 0.1993834 0.1927030 0.1323227 0.1762590
##   carlitos 0.2679949 0.2217224 0.1584190 0.1561697 0.1956941
##   charles  0.2542421 0.2106900 0.1524321 0.1815611 0.2010747
##   eurico   0.2817590 0.1928339 0.1592834 0.1895765 0.1765472
##   jeremy   0.3459730 0.1437390 0.1916520 0.1534392 0.1651969
##   pedro    0.2452107 0.1934866 0.1911877 0.1796935 0.1904215
```

```
prop.table(table(training$classe))
```

```
##
##         A         B         C         D         E
## 0.2843747 0.1935073 0.1743961 0.1638977 0.1838243
```

Based on the above information, let's first do some basic data clean-up by removing columns 1 to 6, which are there just for information and reference purposes:

```
training <- training[, 7:160]
testing  <- testing[, 7:160]
```

and removing all columns that are mostly NA:

```
is_data  <- apply(!is.na(training), 2, sum) > 19621  #which is the number of observations
training <- training[, is_data]
testing  <- testing[, is_data]
```

Before we can move forward with data analysis, we split the training set into two for cross validation purposes. We randomly subsample 60% of the set for training purposes (actual model building), while the 40% remainder will be used only for testing, evaluation and accuracy measurement.

```
library(caret)
```

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

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

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

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

```
## Warning: package 'ggplot2' was built under R version 3.2.3
```

```
##library(caret)
## Loading required package: lattice
## Loading required package: ggplot2
```

```
set.seed(3141592)
inTrain <- createDataPartition(y=training$classe, p=0.60, list=FALSE)
train1  <- training[inTrain,]
train2  <- training[-inTrain,]
dim(train1)
```

```
## [1] 11776    54
```

```
dim(train2)
```

```
## [1] 7846   54
```

The caret library that we are using can be either loaded directly from CRAN using the command install.packages("caret") in R, or downloaded from the caret website. The website also includes a somewhat comprehensive documentation.

At this stage, train1 is the training data set (it contains 11776 observations, or about 60% of the entire training data set), and train2 is the testing data set (it contains 7846 observations, or about 40% of the entire training data set). The dataset train2 will never be looked at, and will be used only for accuracy measurements.

We can now [i] identify the "zero covariates"" from train1 and [ii] remove these "zero covariates"" from both train1 and train2:

```
nzv_cols <- nearZeroVar(train1)
if(length(nzv_cols) > 0) {
  train1 <- train1[, -nzv_cols]
  train2 <- train2[, -nzv_cols]
}
dim(train1)
```

```
## [1] 11776    54
```

```
dim(train2)
```

```
## [1] 7846   54
```

This step didn't do anything as the earlier removal of NA was sufficient to clean the data. We are satisfied that we now have 53 clean covariates to build a model for classe (which is the 54th column of the data set).

Data Manipulation

53 covariates is a lot of variables. Let's look at their relative importance using the output of a quick Random Forest algorithm (which we call directly using randomForest() rather than the caret package purely for speed purposes as we cannot specify the number of trees to use in caret), and plotting data importance using varImpPlot():

```
library(randomForest)
```

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

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

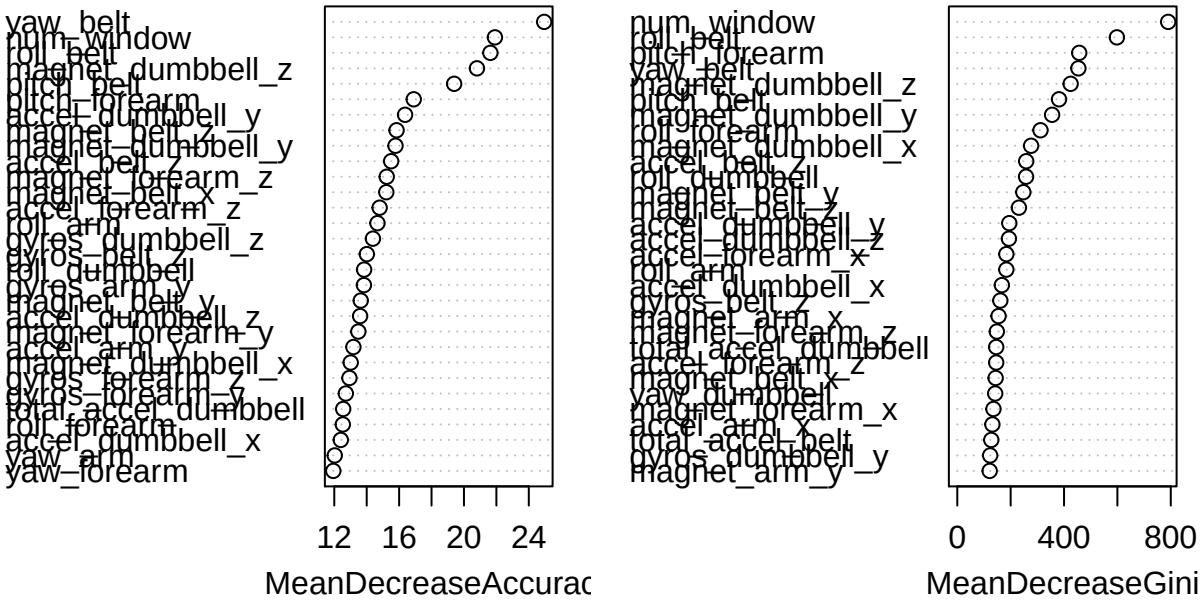
```
## Type rfNews() to see new features/changes/bug fixes.
```

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

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

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(3141592)
fitModel <- randomForest(classe~., data=train1, importance=TRUE, ntree=100)
varImpPlot(fitModel)
```



Using the Accuracy and Gini graphs above, we select the top 10 variables that we'll use for model building. If the accuracy of the resulting model is acceptable, limiting the number of variables is a good idea to ensure readability and interpretability of the model. A model with 10 parameters is certainly much more user friendly than a model with 53 parameters.

Our 10 covariates are: yaw_belt, roll_belt, num_window, pitch_belt, magnet_dumbbell_y, magnet_dumbbell_z, pitch_forearm, accel_dumbbell_y, roll_arm, and roll_forearm.

Let's analyze the correlations between these 10 variables. The following code calculates the correlation matrix, replaces the 1s in the diagonal with 0s, and outputs which variables have an absolute value correlation above 75%:

```
correl = cor(train1[,c("yaw_belt","roll_belt","num_window","pitch_belt","magnet_dumbbell_z","magnet_dumb
diag(correl) <- 0
which(abs(correl)>0.75, arr.ind=TRUE)
```

```
##           row col
## roll_belt   2   1
## yaw_belt    1   2
```

So we may have a problem with roll_belt and yaw_belt which have a high correlation (above 75%) with each other:

```
cor(train1$roll_belt, train1$yaw_belt)
```

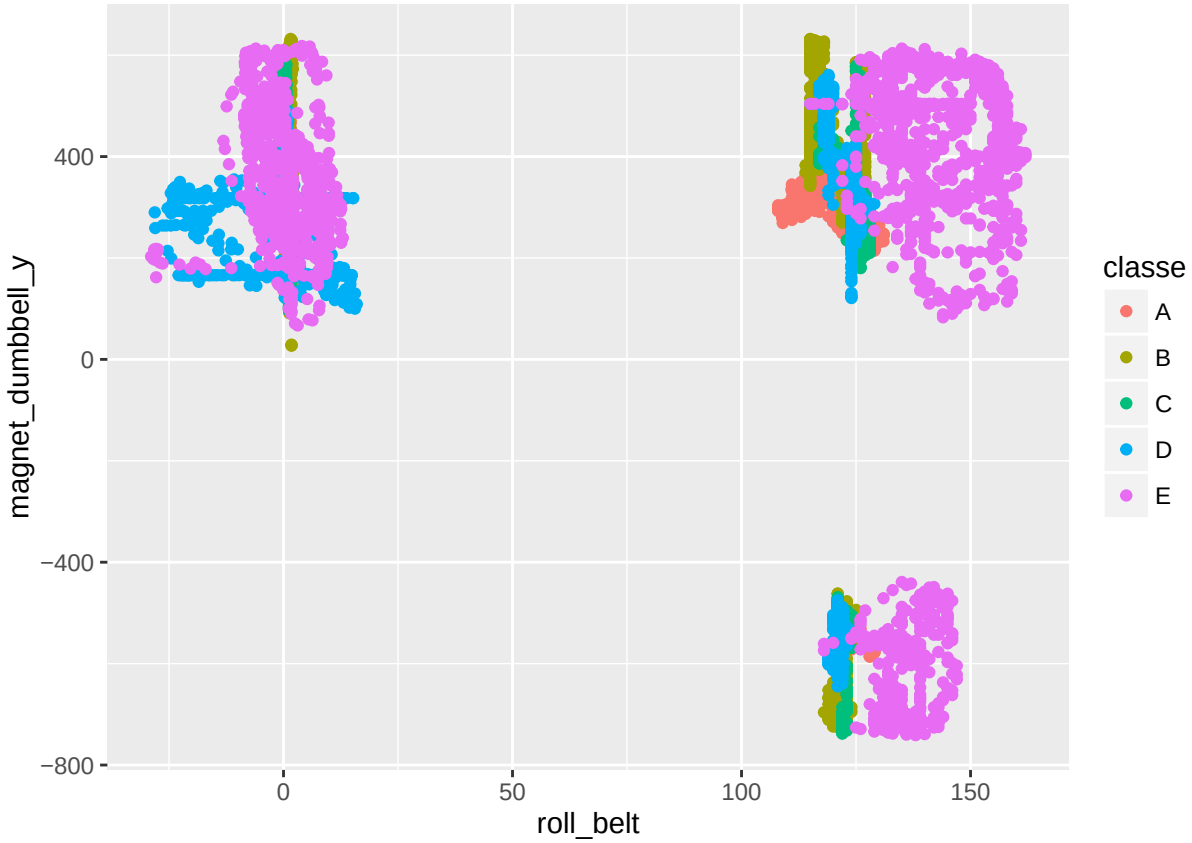
```
## [1] 0.8152349
```

These two variables are on top of the Accuracy and Gini graphs, and it may seem scary to eliminate one of them. Let's be bold and without doing any PCA analysis, we eliminate yaw_belt from the list of 10 variables and concentrate only on the remaining 9 variables.

By re-running the correlation script above (eliminating yaw_belt) and outputting max(correl), we find that the maximum correlation among these 9 variables is 50.57% so we are satisfied with this choice of relatively independent set of covariates.

We can identify an interesting relationship between roll_belt and magnet_dumbbell_y:

```
qplot(roll_belt, magnet_dumbbell_y, colour=classe, data=train1)
```



This graph suggests that we could probably categorize the data into groups based on roll_belt values.

Incidentally, a quick tree classifier selects roll_belt as the first discriminant among all 53 covariates (which explains why we have eliminated yaw_belt instead of roll_belt, and not the opposite: it is a "more important" covariate):

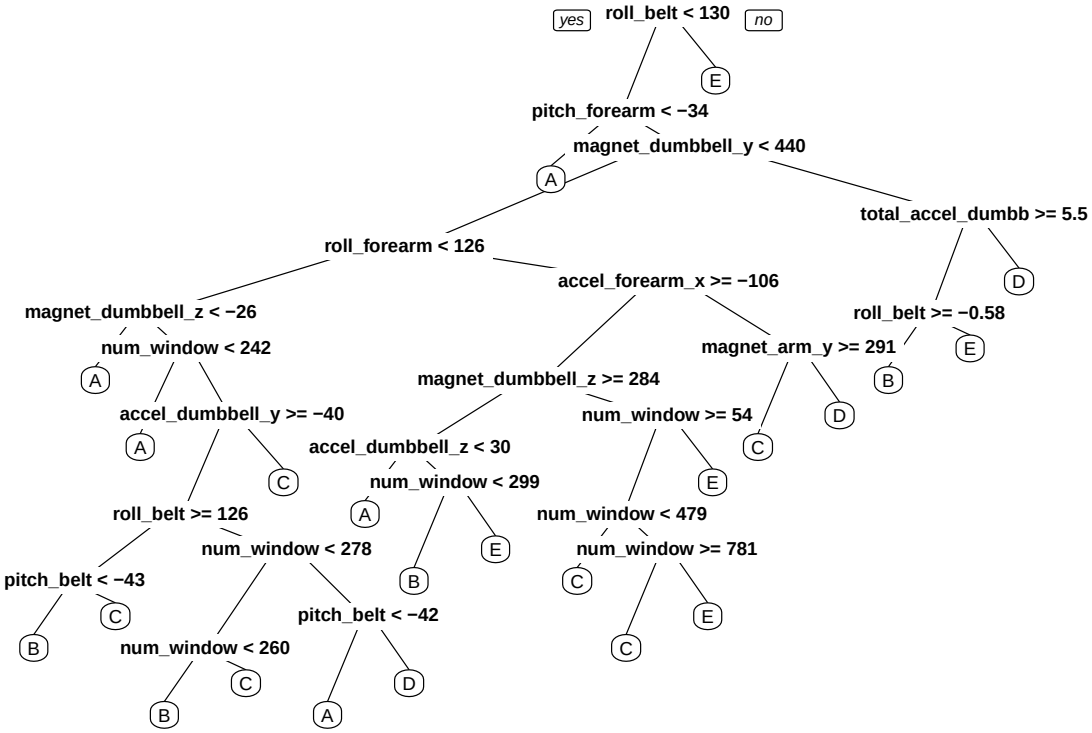
```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.2.3
```

```
## Loading required package: rpart
```

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

```
fitModel <- rpart(classe~., data=train1, method="class")
prp(fitModel)
```



However, we will not investigate tree classifiers further as the Random Forest algorithm will prove very satisfactory

Modeling

We are now ready to create our model. We are using a Random Forest algorithm, using the train() function from the caret package. We are using 9 variables out of the 53 as model parameters. These variables were among the most significant variables generated by an initial Random Forest algorithm, and are roll_belt, num_window, pitch_belt, magnet_dumbbell_y, magnet_dumbbell_z, pitch_forearm, accel_dumbbell_y, roll_arm, and roll_forearm. These variable are relatively independent as the maximum correlation among them is 50.57%. We are using a 2-fold cross-validation control. This is the simplest k-fold cross-validation possible and it will give a reduced computation time. Because the data set is large, using a small number of folds is justified.

```
set.seed(3141592)
fitModel <- train(classe~roll_belt+num_window+pitch_belt+magnet_dumbbell_y+magnet_dumbbell_z+pitch_forea
                  data=train1,
                  method="rf",
                  trControl=trainControl(method="cv",number=2),
                  prox=TRUE,
                  verbose=TRUE,
                  allowParallel=TRUE)
```

The above line of code required 5.2 minutes to execute on a nicely configured Windows 7 64-Bit Sager NP2650 laptop with one i7-4700MQ processor, 6MB L3 cache, 2.40GHz, and 16GB dual channel DDR3 SDRAM at 1600MHz (selecting all 53 variables would increase this time to 7.1 minutes, an increase of 37%, without increasing the accuracy of the model), so we may want to save the model generated for later use:

```
saveRDS(fitModel, "modelRF.Rds")
```

We can later use this tree, by allocating it directly to a variable using the command:

```
fitModel <- readRDS("modelRF.Rds")
```

Note that the modelRF.Rds file uses more than 50M of space on our disk. We are talking about a seriously large file, containing lots of data, about 5 times the size of the training set on the disk)

How accurate is this model?

We can use caret's confusionMatrix() function applied on train2 (the test set) to get an idea of the accuracy

```
predictions <- predict(fitModel, newdata=train2)
confusionMat <- confusionMatrix(predictions, train2$classe)
confusionMat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    0    0    0    0
##          B    1 1517    1    0    2
##          C    0    0 1367    6    1
##          D    0    1    0 1280    4
##          E    0    0    0    0 1435
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.9967, 0.9988)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9993   0.9993   0.9953   0.9951
## Specificity            1.0000   0.9994   0.9989   0.9992   1.0000
## Pos Pred Value         1.0000   0.9974   0.9949   0.9961   1.0000
## Neg Pred Value         0.9998   0.9998   0.9998   0.9991   0.9989
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1933   0.1742   0.1631   0.1829
## Detection Prevalence   0.2843   0.1939   0.1751   0.1638   0.1829
## Balanced Accuracy      0.9998   0.9994   0.9991   0.9973   0.9976
```

Woah! 99.77% is a very impressive number for accuracy which totally validates the idea / hypothesis we made to eliminate most variables and use only 9 relatively independent covariates.

Estimation of the out-of-sample error rate The train2 test set was removed and left untouched during variable selection, training and optimizing of the Random Forest algorithm. Therefore this testing subset gives an unbiased estimate of the Random Forest algorithm's prediction accuracy (99.77% as calculated above). The Random Forest's out-of-sample error rate is derived by the formula 100% - Accuracy = 0.23%, or can be calculated directly by the following lines of code:

```
missClass = function(values, predicted) {
  sum(predicted != values) / length(values)
}
OOS_errRate = missClass(train2$classe, predictions)
OOS_errRate
```

```
## [1] 0.002039256
```

The out-of-sample error rate is 0.23%.

We are now ready to answer Coursera's challenge and predict the 20 observations in testing (recall that testing corresponds to the data set pml-testing.csv)

Coursera Submission

We predict the classification of the 20 observations of the testing data set for Coursera's "Course Project: Submission" challenge page:

```
predictions <- predict(fitModel, newdata=testing)
testing$classe <- predictions
```

We create one .CSV file with all the results, presented in two columns (named problem_id and classe) and 20 rows of data:

```
submit <- data.frame(problem_id = testing$problem_id, classe = predictions)
write.csv(submit, file = "coursera-submission.csv", row.names = FALSE)
```

And we create twenty .TXT file that we will upload one by one in the Coursera website (the 20 files created are called problem_1.txt to problem_20.txt):

```
answers = testing$classe
write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_",i,".txt")
    write.table(x[i], file=filename, quote=FALSE, row.names=FALSE, col.names=FALSE)
  }
}
write_files(answers)
```