Practical Machine Learning: Prediction Assignment Writeup

michael.coursera@eipsoftware.com

November 20, 2017

Synopsis

From the rubric

Source

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, your goal will be to use data 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.

The purpose of the analysis is find a model that can reasonably predict the actions of the user of the wearable device.

Data

Data Set Source

The data was extracted from the research Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements

The dataset has 5 classes (sitting-down, standing-up, standing, walking, and sitting) collected over 8 hours of activities of 4 healthy subjects.

The datasets for the analysis are archived here:

Training Set

Testing Set

Load packages

```
# load standard packages in R
library(plyr);
library(dplyr);
#library(scales);library(reshape2);library(ggplot2);
library(recipes);library(survival);library(splines);library(parallel);library(gbm);library(randomForest
library(caret);
library(corrplot)
```

Load data

Load the two data sets. Convert empty values in the .csv files to NA; to be removed.

```
training_set <- read.csv("./pml-training.csv", header = TRUE, na.strings = c("NA","#DIV/0!",""))    #lo
testing_set  <- read.csv("./pml-testing.csv", header = TRUE, na.strings = c("NA","#DIV/0!","")) #load t

dim(training_set)   #19622 obs
```

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

```
dim(testing_set)    #20 obs
```

```
## [1]  20 160
```

```
summary(training_set$classe)    #summary of results for training set
```

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

Data Processing

1. Remove columns from the testing set that only have NA values.
2. Remove columns from training set that are not in the testing set; preserving the classe column
3. Remove the following columns from testing and training set; they are informational and not pertinent to the data analysis.
    - "X"
    - "user_name"
    - "raw_timestamp_part_1"
    - "raw_timestamp_part_2"
    - "cvtd_timestamp"
    - "new_window"
    - "num_window"
4. Remove any observations that NA values in them.

```
#1. remove N/A columns from the testing set
testing_set_scrub <- Filter(function(x)!all(is.na(x)), testing_set)

#2. remove columns from the training set
training_set_scrub <- training_set %>% select(c(names(testing_set_scrub[,1:59]), "classe"))

#3. remove first 7 columns from training and testing
training_set_scrub <- training_set_scrub %>% select(names(training_set_scrub[8:60]))
testing_set_scrub <- testing_set_scrub %>% select(names(testing_set_scrub[8:60]))


#verify the column names are same in both datasets; excluding the last column
print(c("Column Names are all equal: "
        ,all.equal(names(training_set_scrub[,1:52]), names(testing_set_scrub[,1:52]))
        ))
```

```
## [1] "Column Names are all equal: " "TRUE"
```

```
#4. remove any obs that have N/A values in them.
training_set_scrub <- Filter(function(x)!all(is.na(x)), training_set_scrub)
testing_set_scrub <- Filter(function(x)!all(is.na(x)), testing_set_scrub)

#current state of datasets
dim(training_set_scrub) #19622 obs
```

```
## [1] 19622    53
```

```
dim(testing_set_scrub)  #20 obs
```

```
## [1] 20 53
```

```
summary(training_set$classe)    #summary of results for training set
```

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

Checking for covariates

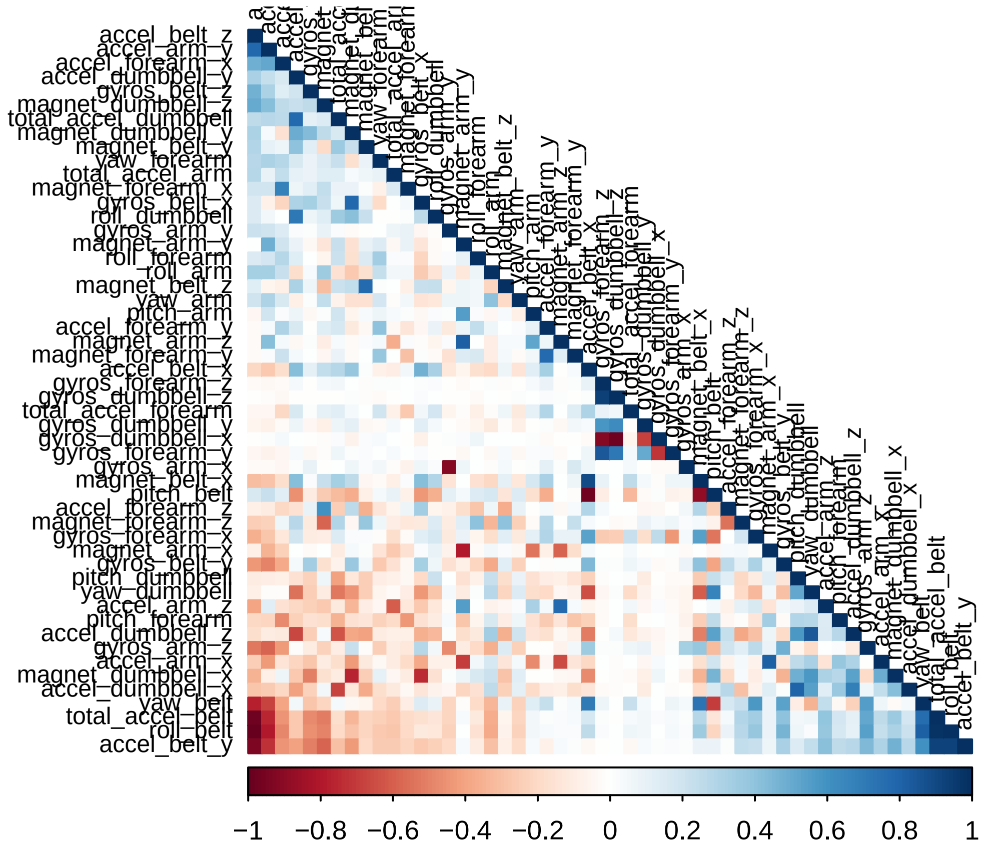
Want to minimize any possible covariates in the datasets.

1. Check for covariates that have low variability
   - show results for any rows that have zeroVar == TRUE or nzv == TRUE
   - remove any zeroVar and nzv variables
   - show results for top 5 with highest unique
2. Plot the correlation results

```
#1. run zero variance check
covariate_check <- nearZeroVar(training_set_scrub[,-53], saveMetrics = TRUE
                               ,names=TRUE ,foreach = TRUE)

#1a. show results
covariate_true_zeroVar <- count(covariate_check %>% filter(zeroVar == FALSE))
covariate_true_nzv <- count(covariate_check %>% filter(nzv == FALSE))



#1b. Remove any zeroVar and nzv from the sets
# no values found skipping step

#1c. show top 5 uniques
covariate_check_u5 <- head(covariate_check[order(covariate_check$percentUnique,decreasing = TRUE),],5)
knitr::kable(covariate_check_u5)
```

	freqRatio	percentUnique	zeroVar	nzv
roll_dumbbell	1.022388	84.20650	FALSE	FALSE
yaw_dumbbell	1.132231	83.48283	FALSE	FALSE
pitch_dumbbell	2.277372	81.74498	FALSE	FALSE
pitch_arm	87.256410	15.73234	FALSE	FALSE
pitch_forearm	65.983051	14.85577	FALSE	FALSE

```
#2: Correlation Results
covariate_matrix_check <- cor(training_set_scrub[,-53])
#plot the results
corrplot(covariate_matrix_check, order = "FPC", method = "color" ,type = "lower"
         ,tl.cex = 0.75, tl.col = "black")
```



Count zeroVar == FALSE: 52

Count nzv == FALSE: 52

There were no zero variance variables; therefore skipped removing any variables from the dataset.

The correlation plot shows a minimum amount of correlated values, at this point I will leave all variables in the dataset.

Model

Group data

There are a large number of observations in the training set provided. In order to determine which training method is better I have elected to break the training data into two groups; and then subdivide those groups into training and test set.

The method will then allow for creating the training models and doing a preliminary test; before testing against the provided testing set of 20 observations.

1. Create two sets of group ids; 70% for the training set and 30% for the testing set
2. Split the two groups into testing and training sets.
3. How many observations in each group?

```r
set.seed(252502525)
#1. two groups 70/30 split
group_ids       <- createDataPartition(training_set_scrub$classe, times = 2, p=0.7, list = FALSE)

#2. training group & testing group
train_group_1   <- training_set_scrub[group_ids[,1],]
test_group_1    <- training_set_scrub[-group_ids[,1],]

train_group_2   <- training_set_scrub[group_ids[,2],]
test_group_2    <- training_set_scrub[-group_ids[,2],]

#13737 obs in training group & 5885 obs in testing group
cbind(nrow(train_group_1),nrow(train_group_2), nrow(test_group_1), nrow(test_group_2))
```

```
##       [,1]  [,2] [,3] [,4]
## [1,] 13737 13737 5885 5885
```

Model Generation

Create two models to predict the results for the testing set. I choose a generalized boosted model and random forest model.

1. Generalized Boosted Model
   - create a control set for the GBM model
   - create the model with a control set
2. Review the results of the model
3. Run the prediction against the test set
4. Create a confusion matrix and review results

GBM Model

```r
#1. create the model
set.seed(252502525)
gbm_control <- trainControl(method = "repeatedcv", number=5, repeats= 1)
gbm_model   <- train(classe ~ ., data = train_group_1, method = "gbm"
                    ,trControl = gbm_control, verbose = FALSE)

#2. results of the model
gbm_model$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 52 predictors of which 44 had non-zero influence.
```

```r
#3. prediction against test_group_1
gbm_predict <- predict(gbm_model, newdata = test_group_1)

#4. create a confusion matrix
gbm_confusion_matrix    <- confusionMatrix(gbm_predict , test_group_1$classe)
gbm_confusion_matrix$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1652   44    0    0    4
##           B   18 1060   36    3   10
##           C    3   33  980   43    6
##           D    0    1    8  911   12
##           E    1    1    2    7 1050
```

```
gbm_confusion_matrix$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9605777      0.9501064      0.9552874      0.9654049      0.2844520
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Random Forest Model

1. Randomized Forest Model
    - create a control set for the rf model
    - create the model with the control set
2. Review the results of the model
3. Run a prediction against the test set
4. Create a confusion matrix and review the results

```
set.seed(252502525)
#1. create the model
rf_control  <- trainControl(method = "cv", number = 5, verboseIter = FALSE)
rf_model    <- train(classe ~ ., data = train_group_2, method="rf" ,trControl=rf_control)

#2. results of the model
rf_model$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.78%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3901    3    1    1    0 0.001280082
## B   19 2632    7    0    0 0.009781791
## C    0   24 2368    4    0 0.011686144
## D    0    0   39 2211    2 0.018206039
## E    0    0    2    5 2518 0.002772277
```

```
#3. prediction against test_group_1
rf_predict  <- predict(rf_model, newdata = test_group_2)

#4. create a confusion matrix
rf_confusion_matrix <- confusionMatrix(rf_predict ,test_group_2$classe)
rf_confusion_matrix$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    1    0    0    0
##          B    0 1137    7    0    0
##          C    1    1 1019   19    1
##          D    0    0    0  945    0
##          E    0    0    0    0 1081
```

```
rf_confusion_matrix$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9949023      0.9935517      0.9927306      0.9965580      0.2844520
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Model Results

The accuracy rates from both models was very high. In non-academic world it would be highly unlikely to have accuracy rates this high. Normally I would find the accuracy rates suspiciously high. And would conclude there is an issue with the model.

However for the purposes of the exercise, it will be assumed the model is not in error.

The random forest model was ~2.8% better than the Generalized Boosted Model.

Out of Sample Error

The sample error for the gbm model was: (1-0.9629) = 0.0371

The sample error for the rf model was: (1-0.9949) = 0.0051

Because the accuracy rates are so high; the corresponding out of sample error rates are very low.

Run Models against the Test Data

Normally I would choose the model with the higher accuracy rate to run against the test data set. However since both models accuracy rate > 90% I decided to run both models against the test data set to verify both produced the same results.

```
#use the scrubbed data

model_1_results <- predict(gbm_model , newdata = testing_set_scrub)
model_2_results <- predict(rf_model , newdata = testing_set_scrub)

model_1_results
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

```
model_2_results
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

Summary:

The results from both models were identical. The accuracy rates from both models are suspiciously higher than anticipated. The random forest model did produce slightly more accurate results. Both models were able to take advantage of a large sample dataset to create the model.

In production code, I would favor the Generalized Boosted Model because the accuracy was high enough, $> 90\%$, and the GBM model ran significantly faster, 8 minutes to process vs 22 minutes, compared to the random forest model.
