Practical Machine Learning: Prediction Assignment

Raju Shah

May 30, 2016

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.

Data

The data sets originated from : Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012 Read more: http://groupware.les.inf.puc-rio.br/har#ixzz40iHm7JQz

Goal

For this project, we are given data from accelerometers on the belt, forearm, arm, and dumbell of 6 research study participants. Our training data consists of accelerometer data and a label identifying the quality of the activity the participant was doing. Our testing data consists of accelerometer data without the identifying label. Our goal is to predict the labels for the test set observations. We will also create a report describing how we built our model, how we used cross validation, what we think the expected out of sample error is, and why we made the choices we did. We will also use our prediction model to predict 20 different test cases.

```
library(ggplot2)

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

library(caret)

## Warning: package 'caret' was built under R version 3.2.5

## Loading required package: lattice

library(randomForest)

## Warning: package 'randomForest' was built under R version 3.2.5
```

```
## 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

library(gbm)

## Warning: package 'gbm' was built under R version 3.2.5

## 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

setwd("d:/coursera/datascience/08-practical-machine-learning/assignment")
```

Load data

- Loading data.

- Remove near zero covariates and those with more than 80% missing values since these variables will not provide much power for prediction.

- Calculate correlations between each remaining feature to the response, `classe`. Use `spearman` rank based correlation because `classe` is a factor.

- Plot the two features that have highest correlation with `classe` and color with `classe` to see if we can separate response based on these features.

```
# loading data
train_file <- "pml-training.csv"
test_file <- "pml-testing.csv"
train_url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
training.csv"
if (!file.exists(train_file)){
    download.file(train_url,train_file)
}
```

```
test_url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
testing.csv"
if (!file.exists(test_file)){
    download.file(test_url,test_file)
}

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

# remove near zero covariates
nsv <- nearZeroVar(training, saveMetrics = T)
training <- training[, !nsv$nzv]
# remove variables with more than 80% missing values
nav <- sapply(colnames(training), function(x) if(sum(is.na(training[, x])) >
0.8*nrow(training)){return(T)}else{return(F)})
training <- training[, !nav]
# calculate correlations
cor <- abs(sapply(colnames(training[, -ncol(training)]), function(x)
cor(as.numeric(training[, x]), as.numeric(training$classe), method =
"spearman")))
```

The training set has **19622** samples and **58** potential predictors after filtering.

There doesn't seem to be any strong predictors that correlates with `classe` well, so linear regression model is probably not suitable in this case. Boosting and random forests algorithms may generate more robust predictions for our data.

Boosting model

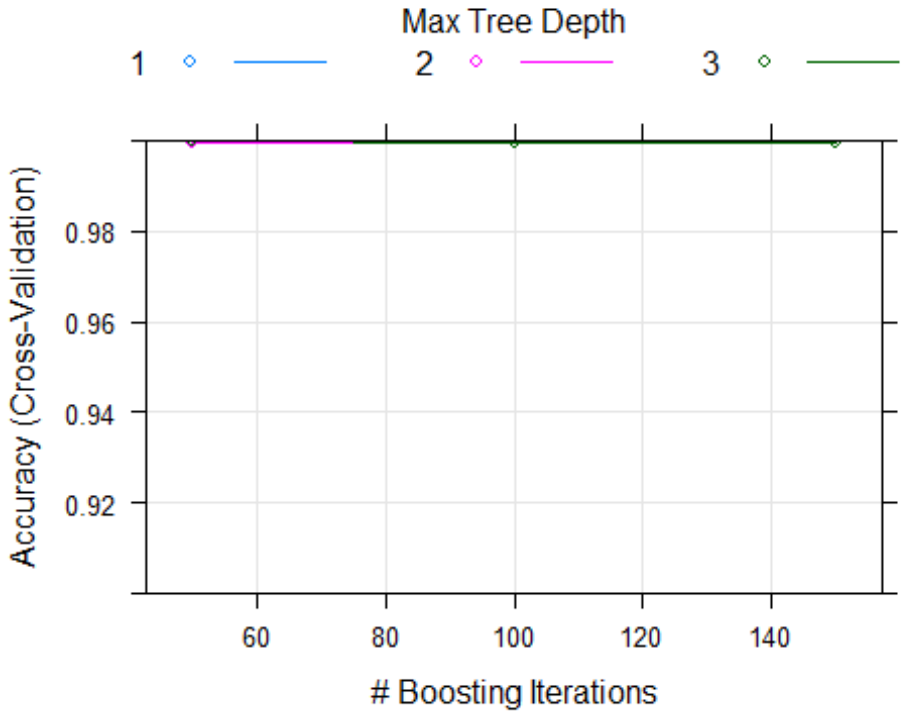
- Fit model with boosting algorithm and 10-fold cross validation to predict `classe` with all other predictors.

- Plot accuracy of this model on the scale `[0.9, 1]`.

```
#partition the dataset into train and test set
dataIndex <- createDataPartition(training$classe, p = 0.7, list = FALSE)
training_set <- training[dataIndex,]
testing_set <- training[-dataIndex,]
modelcontrol <- trainControl(method = "cv", number = 10, verboseIter = FALSE)

set.seed(123)
boostFit <- train(classe ~ ., method = "gbm", data = training_set, verbose =
FALSE, trControl = modelcontrol)

## Loading required package: plyr

## Warning: package 'plyr' was built under R version 3.2.3

boostFit
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    58 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12362, 12363, 12363, 12362, 12363, 12364, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9998544  0.9998159
##   1                  100      0.9997816  0.9997237
##   1                  150      0.9997816  0.9997237
##   2                   50      0.9997088  0.9996317
##   2                  100      0.9997088  0.9996317
##   2                  150      0.9997816  0.9997237
##   3                   50      0.9998544  0.9998159
##   3                  100      0.9997816  0.9997237
##   3                  150      0.9997816  0.9997237
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 50, interaction.depth
##  = 1, shrinkage = 0.1 and n.minobsinnode = 10.
```

```r
plot(boostFit, ylim = c(0.9, 1))
```



The boosting algorithm generated a good model with **accuracy = 0.997**.

Random forests model

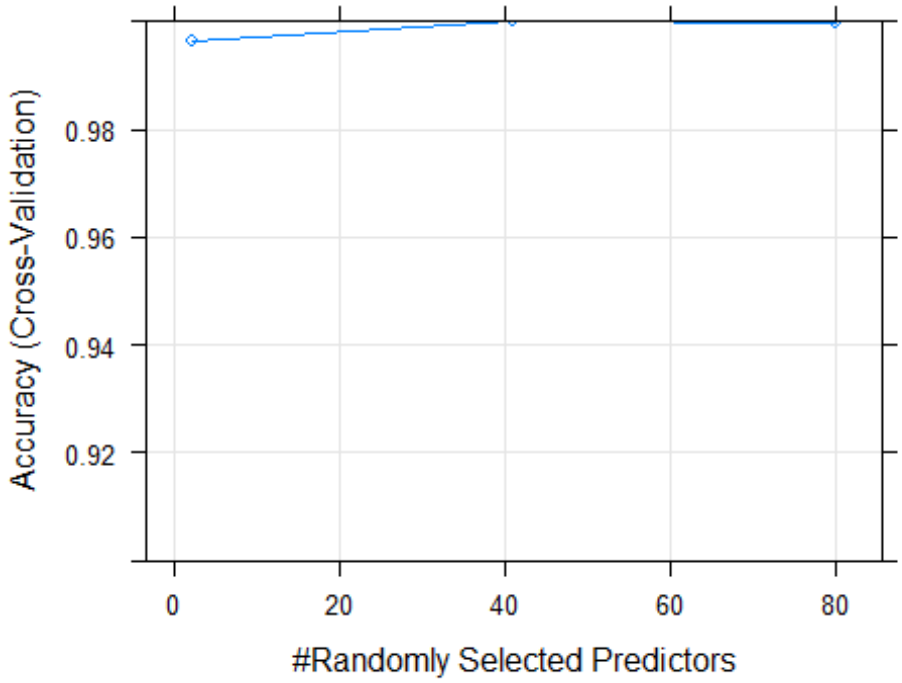
- Fit model with random forests algorithm and 10-fold cross validation to predict `classe` with all other predictors.

- Plot accuracy of the model on the **same scale** as boosting model.

```
set.seed(123)
rfFit <- train(classe ~ ., method = "rf", data = training_set, trControl =
modelcontrol)

rfFit

## Random Forest
##
## 13737 samples
##    58 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12362, 12363, 12363, 12362, 12363, 12364, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9961416  0.9951196
```

```
##   41    1.0000000  1.0000000
##   80    0.9998545  0.9998160
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 41.
```

```
plot(rfFit, ylim = c(0.9, 1))
```




```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    0 1139    0    0    0
##          C    0    1 1025    0    0
##          D    0    0    0  964    0
##          E    0    0    0    0 1082
##
## Overall Statistics
##
##                Accuracy : 0.9998
##                  95% CI : (0.9991, 1)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9998
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9991   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   0.9998   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   0.9990   1.0000   1.0000
## Neg Pred Value         1.0000   0.9998   1.0000   1.0000   1.0000
## Prevalence             0.2845   0.1937   0.1742   0.1638   0.1839
## Detection Rate         0.2845   0.1935   0.1742   0.1638   0.1839
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      1.0000   0.9996   0.9999   1.0000   1.0000
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    1    0    0    0
##          B    0 1139    0    0    0
##          C    0    0 1026    0    0
##          D    0    0    0  963    1
##          E    0    0    0    0 1082
##
## Overall Statistics
##
##                Accuracy : 0.9997
##                  95% CI : (0.9988, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9996
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9991   1.0000   1.0000   0.9991
## Specificity            0.9998   1.0000   1.0000   0.9998   1.0000
## Pos Pred Value         0.9994   1.0000   1.0000   0.9990   1.0000
## Neg Pred Value         1.0000   0.9998   1.0000   1.0000   0.9998
## Prevalence             0.2843   0.1937   0.1743   0.1636   0.1840
## Detection Rate         0.2843   0.1935   0.1743   0.1636   0.1839
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9999   0.9996   1.0000   0.9999   0.9995
```

The random forests algorithm generated a very accurate model with **accuracy close to 1**. Compared to boosting model, this model generally has better performance in terms of accuracy as we see from the plots.

Out of Sample error

```
## Calculate OOS Error
missClass = function(values, predicted) {
        sum(predicted != values) / length(values)
}
OOS_errRateRF = missClass(testing_set$classe, rfFit_predicted)
OOS_errRateRF
```

```
## [1] 0.0001699235
```

Final model and prediction

- Comparing model accuracy of the two models generated, random forests and boosting, random forests model has overall better accuracy. So, I'll use this model for prediction.

- The final random forests model contains 500 trees with 40 variables tried at each split. The five most important predictors in this model are X, roll_belt, pitch_forearm, raw_timestamp_part_1, accel_belt_z.

- Estimated **out of sample error rate** for the random forests model is **1.699235310^{-4}%** as reported by the final model.

- Predict the test set and output results for automatic grader.

```
# final model
rfFit$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: 41
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3906    0    0    0    0           0
## B    0 2658    0    0    0           0
## C    0    0 2396    0    0           0
## D    0    0    0 2252    0           0
## E    0    0    0    0 2525           0
```

```
# prediction
prediction <- as.character(predict(rfFit, testing))
```

```
# write prediction files
pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("prediction/problem_id_", i, ".txt")
    write.table(x[i], file = filename, quote = FALSE, row.names = FALSE,
```

```
col.names = FALSE)
  }
}
pml_write_files(prediction)
```