# Practical Machine Learning - Prediction Of Weight Lifting Excersise Activity

*Bhaskar Parvathaneni*

*May 15, 2015*

## Executive Summary

In this report we analyze the Humam Activity Recognition(HAR) data for weight lifting exercises. The goal is to predict how well an activity was performed by the person wearing the activity measuring device. Three prediction model algorithms were considered for prediction as outlined later in the model selection section. The best model selected for generating the prediction was *Random Forest* which had an accuracy of about *99.4%* and an out-of-sample error of about *0.60%*. A detailed comparison of results from the 3 models is presented in the results section.

## Data Preperation & Pre-Processing

The data loading and cleaning steps are described in the following sub-sections.

### Data Overview

The Weight Lifting Excercise(WLE) dataset was collected for a group of six participants aged 20-28 years wearing activity measuring devices. The activities measured were classified as follows

- Class A - done exactly accoring to specification
- Class B - throwing the elbows to the front
- Class C - lifting the dumbell only halfway
- Class D - lowering the dumbell only halfway
- Class E - throwing the hips to the front

### Data Loading

The data was downloaded into the current working directory set in R for the user.

```
cache=TRUE
#
# Download storm data file if it does not exist.
#
wleTrainDatasetURL="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
wleTestDatasetURL="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

if (!file.exists("pml-training.csv")) {
    downloadFile(wleTrainDatasetURL, "pml-training.csv")
}

if (!file.exists("pml-testing.csv")) {
```

```
    downloadFile(wleTestDatasetURL, "pml-training.csv")
}

rawTraining <- read.csv("pml-training.csv", na.strings=c("NA", ""))
rawTesting <- read.csv("pml-testing.csv", na.strings=c("NA", ""))
```

### Data Summary

The dimensions of the initial training and test dataset is as follows.

```
dim(rawTraining)
```

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

```
dim(rawTesting)
```

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

### Cleaning Data

In this step we first remove all variables with a high percentage of missing and *NA* values. We also remove variables that are not relevant to our analysis.

- Remove all variables with missing and *NA* values using the following R-script.

```
dropColumnsWithMissingAndNAValues <- (colSums(is.na(rawTraining))==0)
training <- rawTraining[,dropColumnsWithMissingAndNAValues]
dim(training)
```

```
## [1] 19622    60
```

- Remove the first 7 variables as they are not relevant to the prediction outcome.

```
training <- training[,8:length(training)]
dim(training)
```

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

```
trainingPredictorCols <- colnames(training[,1:length(training)-1])
testing <- rawTesting[,c(trainingPredictorCols, "problem_id")]
dim(testing)
```

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

```
set.seed(131719)
```

The above cleaning steps have reduced the number of predictor variables from *159* to *52*.

### Data Pre-Processing To Remove Highly Corelated Variables

In this step we analyze remaining predictor variables in the training dataset to determine the degree of co-relation. Based on this analysis we remove all variables with a high degree of correlation as they will contribute to higher variance and less accuracy. This step further reduces the predictor variables to *45*, the eleminated variables in this step are as follows.

```
suppressMessages(require(caret))
corTraining <- cor(training[,-53])
highlyCorelatedTrainingVarIndexes <- findCorrelation(corTraining, cutoff=.90)
names(training[,highlyCorelatedTrainingVarIndexes])
```

```
## [1] "accel_belt_z"     "roll_belt"        "accel_belt_y"
## [4] "accel_belt_x"     "gyros_arm_y"      "gyros_forearm_z"
## [7] "gyros_dumbbell_x"
```

```
training <- training[,-highlyCorelatedTrainingVarIndexes]
dim(training)
```

```
## [1] 19622    46
```

### Distribution Of Training Data

The distribution of training data based on the class of weight lifting excercise is as follows.

```
ggplot(data.frame(training),aes(x=classe)) + geom_histogram(fill="slateblue") + xlab("Class Of Weight Li
```

## Data Slicing

The training dataset is pretty big and is sliced further into training and validation subsets. This slicing will help in using the validation set for training and cross-validating the prediction model.

```
inTrain2 <- createDataPartition(y=training$classe, p=0.7, list=FALSE)
training2 <- training[inTrain2,]
validation2 <- training[-inTrain2,]
dim(training2)
```

```
## [1] 13737    46
```

```
dim(validation2)
```

```
## [1] 5885   46
```

## Prediction Model Selection

In the model selection step we consider three different model selection algorithms. A model is generated for ecach of the three cases by training and cross-validating with the datasets. The prediction model generated in each case will be used to perform the confusion matrix analysis to determine the accuracy and out-of-sample error.

- Regression Tree (rpart)
- Random Forest (rf)
- Gradient Boosting (gbm)

### Regression Tree

In this case the data is analyzed by doing a recursive classification that builds a binary decision tree to pick the features in the model.

#### Prediction Model Fit

In this case the prediction model is generated using the *rpart* machine learning algorithm that is part of the *caret* package. The *rpart* processing will take a few minutes(2 to 6 mins).

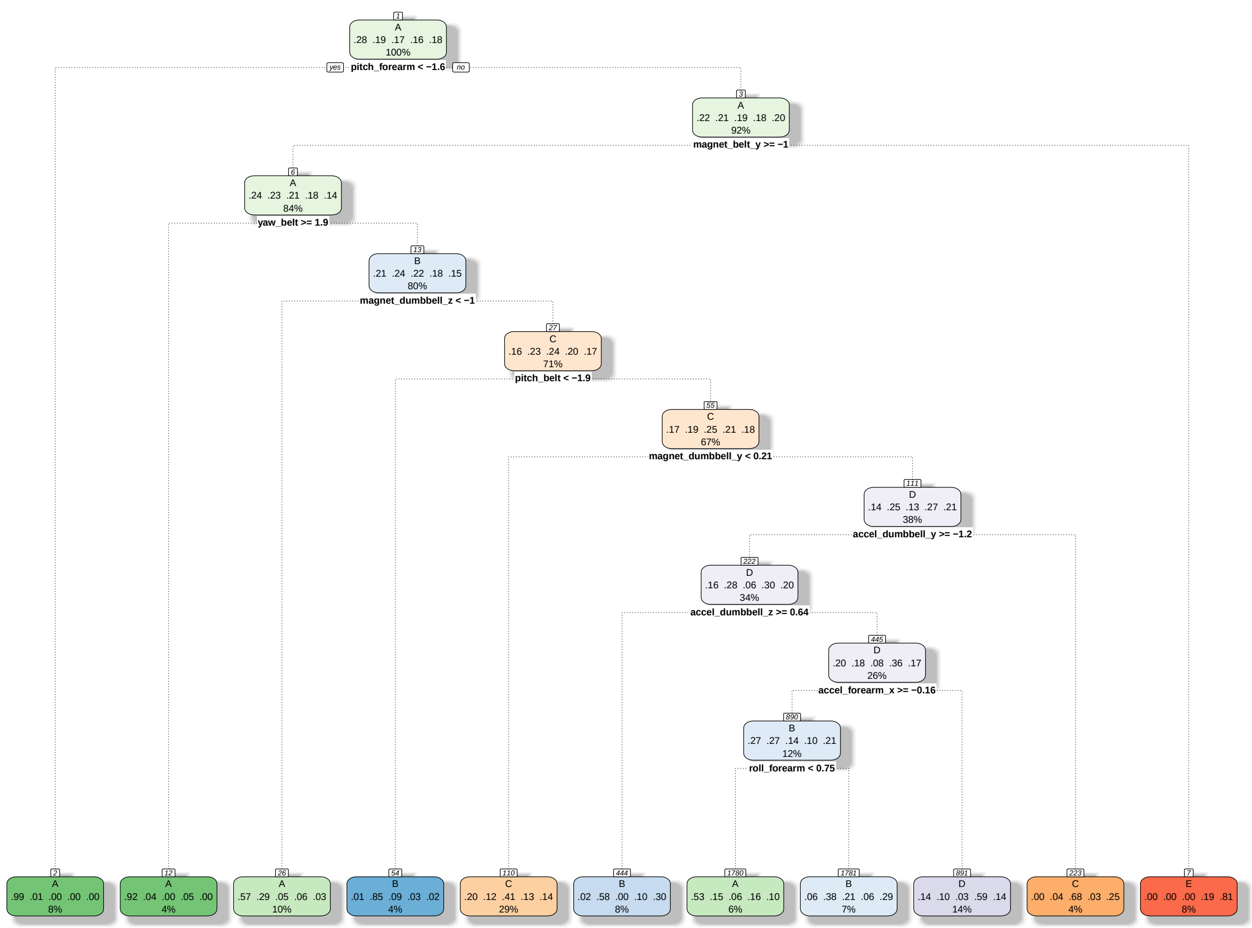
```
suppressMessages(require(rpart))
suppressMessages(require(rpart.plot))
rpartModelFit <- train(classe~.,data=training2,preProcess=c("knnImpute","center","scale"),method="rpart
rpartModelFit
```

```
## CART
##
## 13737 samples
##    45 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
```

```
## Pre-processing:  nearest neighbor imputation, centered, scaled
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 12362, 12365, 12363, 12363, 12362, 12365, ...
##
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.02044553  0.5960457  0.4891410  0.03551772   0.04293449
##   0.02944767  0.5525940  0.4371618  0.03035805   0.03548932
##   0.04353575  0.4088399  0.2143625  0.11012072   0.18809556
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.02044553.
```

### Regression Decision Tree Plot

```r
suppressMessages(require(rattle))
fancyRpartPlot(rpartModelFit$finalModel)
```




Rattle 2015−May−23 11:27:14 bparvat

### Prediction & Accuracy Using Regression Tree

The *Regression Tree* model generated in the previous step is used to predict the outcome *classe* on the validation set. The accuracy of the prediction based on the confusion matrix analysis is as follows.

```
rpartPredict <- predict(rpartModelFit,newdata=validation2)
confusionMatrix(rpartPredict,validation2$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1213  226   50   96   67
##          B   41  616  102   79  270
##          C  330  217  845  241  293
##          D   88   79   28  481  119
##          E    2    1    1   67  333
##
## Overall Statistics
##
##                Accuracy : 0.5927
##                  95% CI : (0.58, 0.6053)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4857
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7246   0.5408   0.8236  0.49896  0.30776
## Specificity            0.8957   0.8963   0.7775  0.93619  0.98522
## Pos Pred Value         0.7343   0.5560   0.4387  0.60503  0.82426
## Neg Pred Value         0.8911   0.8905   0.9543  0.90511  0.86335
## Prevalence             0.2845   0.1935   0.1743  0.16381  0.18386
## Detection Rate         0.2061   0.1047   0.1436  0.08173  0.05658
## Detection Prevalence   0.2807   0.1883   0.3273  0.13509  0.06865
## Balanced Accuracy      0.8102   0.7186   0.8006  0.71758  0.64649
```

## Random Forest

In this method an ensemble learning method is used for classification and regression tasks. This is accomplished by construction multiple decision trees at training time and outputing the mean prediction of individual trees.

### Prediction Model Fit

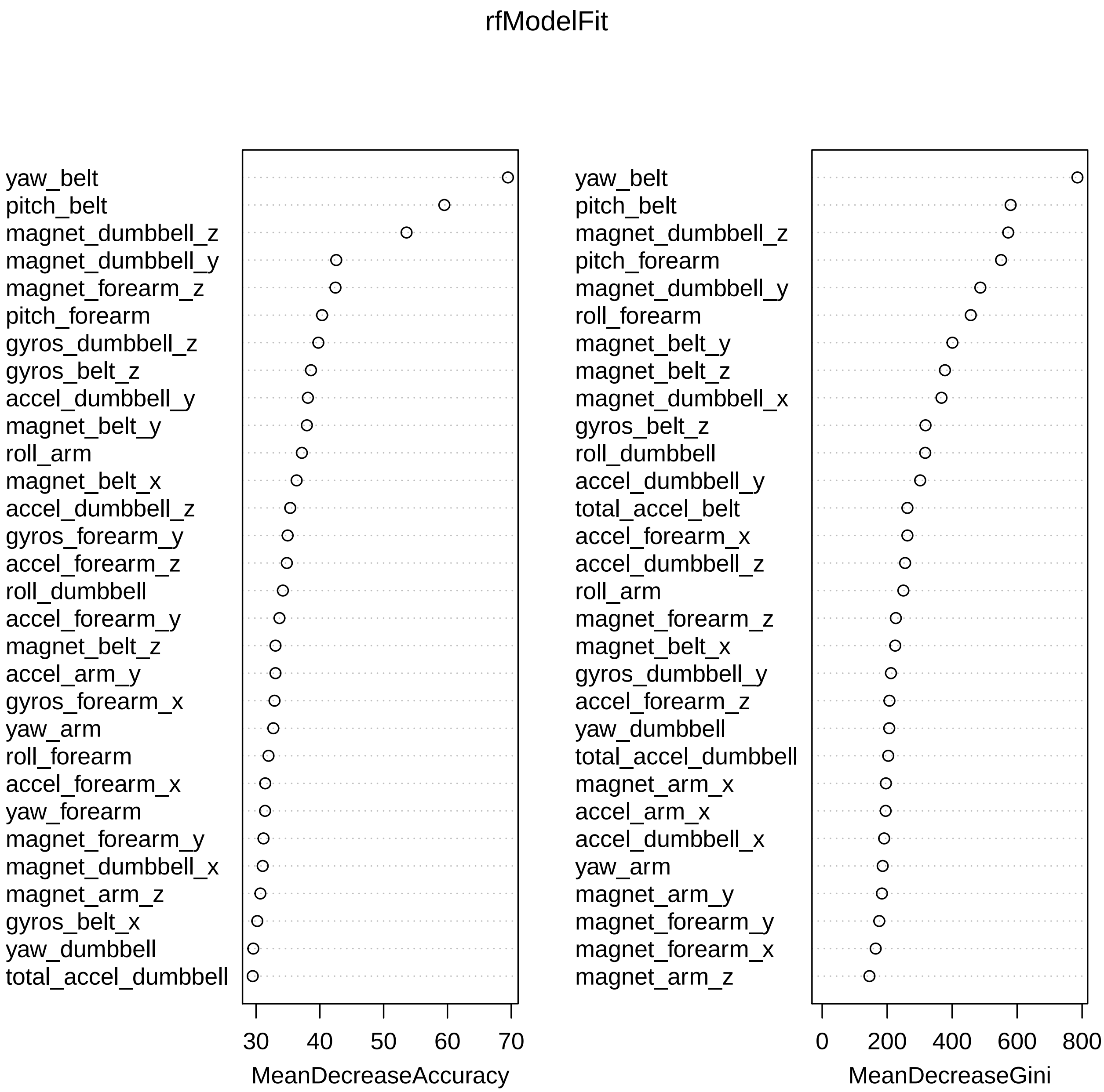
The prediction model is generated using the *randomForest* machine learning algorithm in the *randomForest* package. This processing can take upto a few mins (Usually between 5 & 10 mins).

```
##
## Call:
```

```
##  randomForest(formula = classe ~ ., data = training2, nTree = 200,      importance = TRUE, proximity
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 0.6%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3903    1    0    1    1 0.0007680492
## B   17 2633    7    0    1 0.0094055681
## C    0   19 2373    4    0 0.0095993322
## D    1    0   23 2227    1 0.0111012433
## E    0    0    2    4 2519 0.0023762376
```

### Plot Of Variable Importance In Random Forest

```
varImpPlot(rfModelFit)
```




#### Prediction & Accuracy Using Random Forest

The *Random Forest* model fit generated in the previous step is used to predict the outcome on the validation data. The accuracy results based on confusion matrix analysis is as follows.

```
rfPredict <- predict(rfModelFit, newdata=validation2)
confusionMatrix(rfPredict,validation2$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    9    0    0    0
```

```
##          B    2 1129   12    0    0
##          C    0    1 1014   12    1
##          D    0    0    0  949    0
##          E    0    0    0    3 1081
##
## Overall Statistics
##
##                Accuracy : 0.9932
##                  95% CI : (0.9908, 0.9951)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9914
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9912   0.9883   0.9844   0.9991
## Specificity            0.9979   0.9971   0.9971   1.0000   0.9994
## Pos Pred Value         0.9946   0.9878   0.9864   1.0000   0.9972
## Neg Pred Value         0.9995   0.9979   0.9975   0.9970   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1918   0.1723   0.1613   0.1837
## Detection Prevalence   0.2856   0.1942   0.1747   0.1613   0.1842
## Balanced Accuracy      0.9983   0.9941   0.9927   0.9922   0.9992
```

## Gradient Boosting

This is a machine learning technique that produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

### Prediction Model Fit

The prediction model is generated using the *gbm* machine learing algorithm in the *caret* package.

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    45 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing:  nearest neighbor imputation, centered, scaled
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 12364, 12364, 12362, 12363, 12363, 12365, ...
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa      Accuracy SD
##   1                   50      0.7396765  0.6700783  0.014788924
##   1                  100      0.8125461  0.7627012  0.015311359
```

```
##   1                   150      0.8449411  0.8037372  0.010464066
##   2                    50      0.8521471  0.8127287  0.011599431
##   2                   100      0.9063104  0.8814294  0.006343022
##   2                   150      0.9303333  0.9118460  0.004247134
##   3                    50      0.8956077  0.8678228  0.007653449
##   3                   100      0.9409606  0.9252897  0.005517564
##   3                   150      0.9598879  0.9492495  0.004418185
##   Kappa SD
##   0.018628991
##   0.019378692
##   0.013222650
##   0.014680482
##   0.008046662
##   0.005376777
##   0.009734261
##   0.006992231
##   0.005591667
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3 and shrinkage = 0.1.
```

### Prediction & Accuracy Using Gradient Boosting

The *gbm* model fit from the previous step is used to predict the outcome against the validation dataset. The accuracy results from the confusion matrix analysis is as follows.

```
gbmPredict <- predict(gbmModelFit,newdata=validation2)
confusionMatrix(gbmPredict,validation2$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1639   37    0    2    3
##          B   23 1060   43    3    9
##          C   11   35  966   34   15
##          D    0    1   15  911   12
##          E    1    6    2   14 1043
##
## Overall Statistics
##
##                Accuracy : 0.9548
##                  95% CI : (0.9492, 0.96)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9428
##  Mcnemar's Test P-Value : 5.332e-05
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9791   0.9306   0.9415   0.9450   0.9640
## Specificity            0.9900   0.9836   0.9804   0.9943   0.9952
## Pos Pred Value         0.9750   0.9315   0.9105   0.9702   0.9784
## Neg Pred Value         0.9917   0.9834   0.9876   0.9893   0.9919
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2785   0.1801   0.1641   0.1548   0.1772
## Detection Prevalence   0.2856   0.1934   0.1803   0.1596   0.1811
## Balanced Accuracy      0.9846   0.9571   0.9610   0.9697   0.9796
```

# Results & Prediction With Test Data

Comparing the accuracy results from the three machine learning algorithms it is clear that the *Random Forest* performs best with *99.4%* accuracy with an out-of-sample error between *0.58%* & *0.60%*. The accuracy results and the out-of-sample error for each of the model selection algorithms were computed using the confusion matrix analysis using the validation dataset. The accuracy and the excpected out-of-sample error are as follows.

- Regression Tree Model has accuracy of about *59%* with an out-of-sample error of about *41%*.

```
##  Accuracy
## 0.5926933
```

- Random Forest Model has an accuracy of *99.4%* with an out-of-sample error of about *0.60%*

```
##  Accuracy
## 0.9932031
```

- Gradient Boosting Model has an accuracy of about *95%*, with an out-of-sample error of about *5%*.

```
##  Accuracy
## 0.9548003
```

Applying the best fitting model *Random Forest* to the test dataset we get the following prediction results.

```
testPredictWithRandomForestModel <- predict(rfModelFit,newdata=testing)
testPredictWithRandomForestModel
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

The code fore generating the project submission results is as follows.

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

pml_write_files(testPredictWithRandomForestModel)
```

# References


1. 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.
2. HAR Dataset
3. Random Forests