Practical Machine Learning - Course Project

Predicting Movements from Data

Bruce Leistikow

March 20, 2016

Executive Summary

The purpose of this analysis is to make predictions about how people performed certain exercises. Within the training set, the variable named "classe" provides this data for each observation. This reports describes an approach to building a prediction model, illustrates cross validation, and offers an expectation of the sample error. Lastly, the model is applied to a test dataset to generate 20 predictions.

Load and Clean Data

The training and testing datasets are avaiable here:

- Training
- Testing

Explanatory documentation for the data can be found here:

- Data Description

Load required libraries used in this analysis.

```
library(caret)
library(ggplot2)
library(corrplot)
library(randomForest)
library(doParallel)
```

Load csv files into R for further refinement.

```
setwd("/Users/bleistikow/Documents/Coursera/Practical_Machine_Learning")
rawTrainingData <- read.csv("pml-training.csv", na.strings = c("NA", ""), strip.white = TRUE)
rawPredictData <- read.csv("pml-testing.csv", na.strings = c("NA", ""), strip.white = TRUE)
```

The first 7 columns do not add value to the prediction model, so they are removed from both datasets here

```
reducedTrainingData <- rawTrainingData[, -(1:7)]
reducedPredictData <- rawPredictData[, -(1:7)]
```

Many data elements contain NA's which can have an impact on the accuracy of the prediction model. Thus, removal of columns with numerous NA observations is imperative.

```
NAs <- apply(reducedTrainingData, 2, function(x) {sum(is.na(x))})

validTrainingData <- reducedTrainingData[, which(NAs == 0)]
validPredictData <- reducedPredictData[, which(NAs == 0)]
dim(validTrainingData)
```

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

```
dim(validPredictData)
```

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

Setup Data for Cross Validation

Partition the data into training and test sets for cross validation. The training data will receive 60% of the observations and the remaining 40% will go toward cross validation test data although this is unnecessary since we will likely choose a Random Forest model and cross validation is ‘baked in’ as the data is subsampled by the algorithm used to generate the model.

```
inTrain <- createDataPartition(y = validTrainingData$classe, p = 0.6, list = FALSE)
trainingData <- validTrainingData[inTrain, ]
testingData <- validTrainingData[-inTrain, ]
```

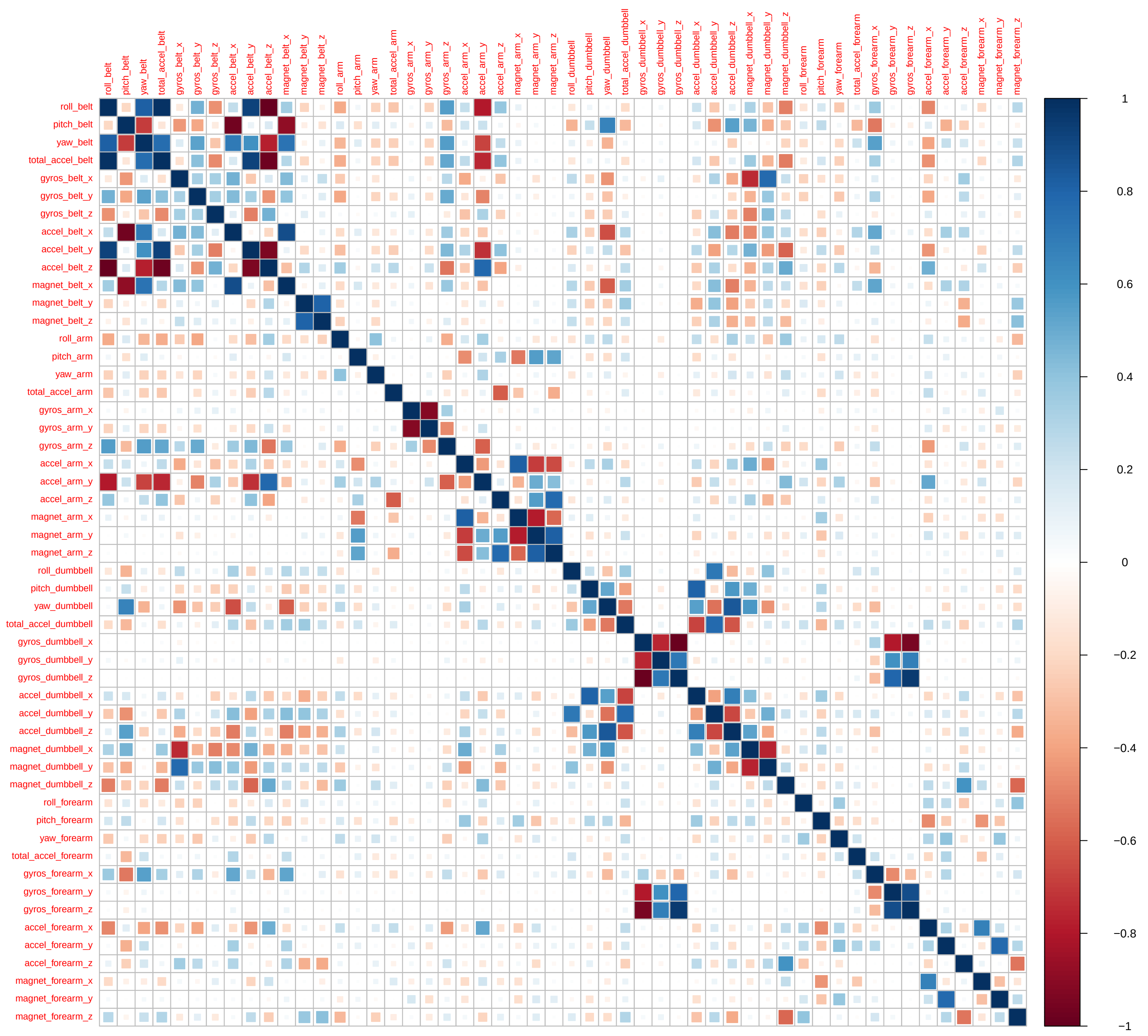
Review the cleaned and partitioned training data. The training data now has 11776 rows and 53 columns. One of which is the predictor/outcome variable named, “classe”.

```
head(trainingData, 2)
```

```
##   roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x gyros_belt_y
## 1      1.41       8.07    -94.4                3         0.00         0.00
## 5      1.48       8.07    -94.4                3         0.02         0.02
##   gyros_belt_z accel_belt_x accel_belt_y accel_belt_z magnet_belt_x
## 1        -0.02          -21            4           22            -3
## 5        -0.02          -21            2           24            -6
##   magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm total_accel_arm
## 1           599          -313     -128      22.5    -161              34
## 5           600          -302     -128      22.1    -161              34
##   gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x accel_arm_y accel_arm_z
## 1           0        0.00       -0.02        -288         109        -123
## 5           0       -0.03        0.00        -289         111        -123
##   magnet_arm_x magnet_arm_y magnet_arm_z roll_dumbbell pitch_dumbbell
## 1         -368          337          516      13.05217      -70.49400
## 5         -374          337          506      13.37872      -70.42856
##   yaw_dumbbell total_accel_dumbbell gyros_dumbbell_x gyros_dumbbell_y
## 1    -84.87394                   37                0            -0.02
## 5    -84.85306                   37                0            -0.02
##   gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y accel_dumbbell_z
## 1                0             -234               47             -271
## 5                0             -233               48             -270
##   magnet_dumbbell_x magnet_dumbbell_y magnet_dumbbell_z roll_forearm
## 1              -559               293               -65         28.4
## 5              -554               292               -68         28.0
##   pitch_forearm yaw_forearm total_accel_forearm gyros_forearm_x
## 1         -63.9        -153                  36            0.03
## 5         -63.9        -152                  36            0.02
##   gyros_forearm_y gyros_forearm_z accel_forearm_x accel_forearm_y
## 1               0           -0.02             192             203
## 5               0           -0.02             189             206
##   accel_forearm_z magnet_forearm_x magnet_forearm_y magnet_forearm_z
## 1            -215              -17              654              476
## 5            -214              -17              655              473
##   classe
## 1      A
## 5      A
```

A review of the correlation graph may prove interesting as it will show which variables are inter-related and which are not. This could also be useful if we wanted to further reduce the factors used in building the model.

```r
correlationData = cor(trainingData[,-c(grep("classe", names(trainingData)), length(trainingData))],
                      use = "pairwise.complete.obs")
corrplot(correlationData, method = "square", tl.cex=0.7)
```



Build Model

A Random Forest model typically yields the highest accuracy, but can be computationally intensive. The `randomForest()` method is used here since it is speedier than the `train()` function with `method="rf"` parameter found in the `caret` package.

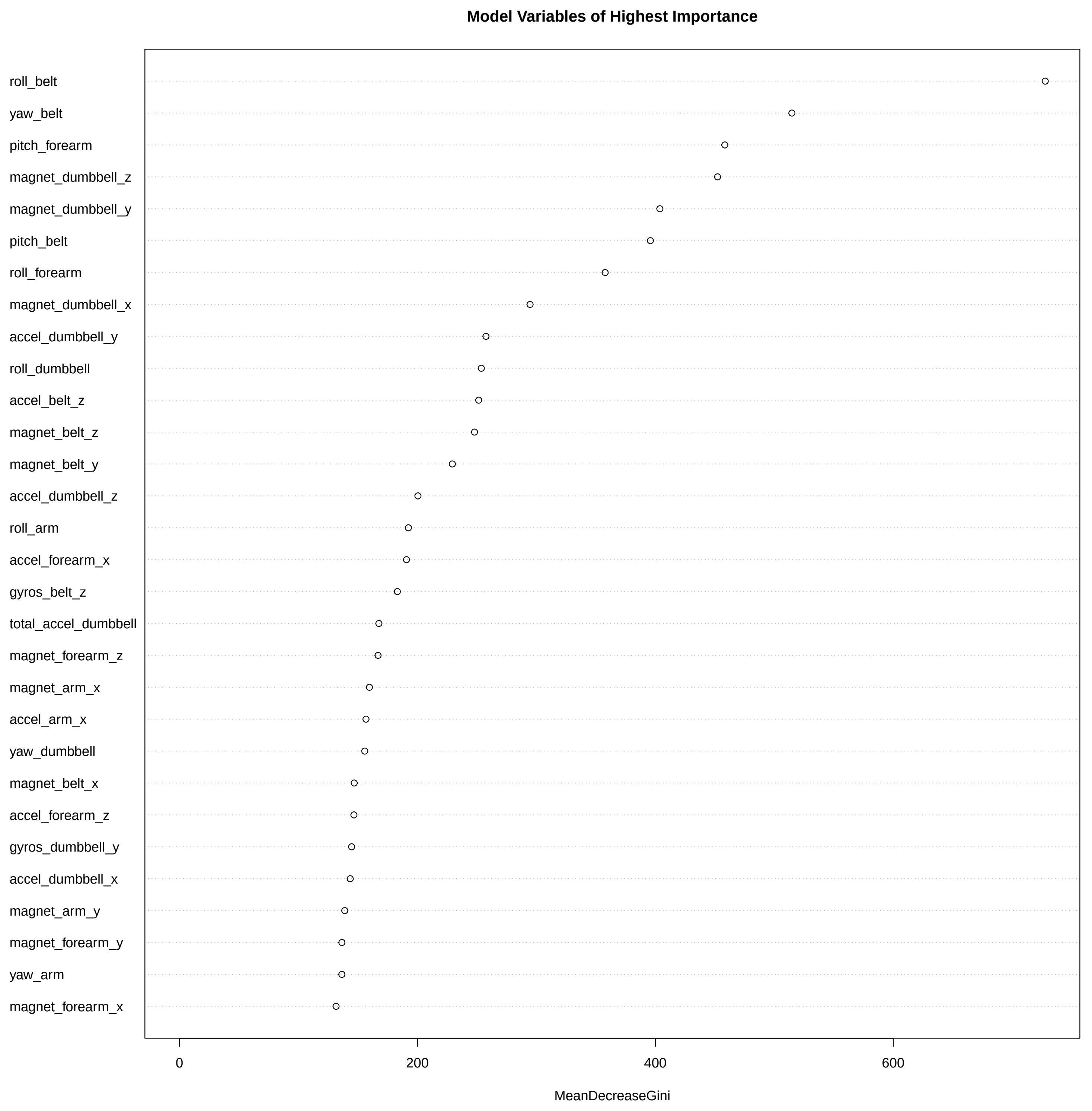
```r
set.seed(320)
rfModel <- randomForest(classe ~ ., data=trainingData)
rfModel
```

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

```
##  randomForest(formula = classe ~ ., data = trainingData)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.65%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3345    1    0    0    2 0.0008960573
## B    9 2265    5    0    0 0.0061430452
## C    0   17 2033    4    0 0.0102239533
## D    0    0   25 1901    4 0.0150259067
## E    0    0    1    8 2156 0.0041570439
```

Another interesting graphic is the Variable Importance Plot. We generate this plot to quickly examine the most important factors in the Random Forest model.

```
varImpPlot(rfModel, main = "Model Variables of Highest Importance")
```



Assessing Accuracy of the Model

We use several methods to assess the accuracy of the model, and now Cross Validate the model on the data reserved for testing which we set-aside earlier. We do this before applying the model to the prediction data.

```
pred <- predict(rfModel, newdata=testingData)
sum(pred == testingData$classe) / length(pred)
```

```
## [1] 0.9934999
```

```
crossValidationData <- postResample(pred, testingData$classe)
crossValidationData
```

```
##  Accuracy     Kappa
## 0.9934999 0.9917771
```

Our model seems quite good with Accuracy of: 99.35%.

Confusion Matrix

Apply a Confusion Matrix and examine the error rate of the model

```
set.seed(320)
confuseMat <- confusionMatrix(pred, testingData$classe)
confuseMat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    8    0    0    0
##          B    1 1508    8    0    0
##          C    0    2 1359   21    5
##          D    0    0    1 1264    4
##          E    0    0    0    1 1433
##
## Overall Statistics
##
##                Accuracy : 0.9935
##                  95% CI : (0.9915, 0.9952)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9918
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9934   0.9934   0.9829   0.9938
## Specificity            0.9986   0.9986   0.9957   0.9992   0.9998
## Pos Pred Value         0.9964   0.9941   0.9798   0.9961   0.9993
## Neg Pred Value         0.9998   0.9984   0.9986   0.9967   0.9986
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1922   0.1732   0.1611   0.1826
## Detection Prevalence   0.2854   0.1933   0.1768   0.1617   0.1828
## Balanced Accuracy      0.9991   0.9960   0.9945   0.9911   0.9968
```

```
pr <- postResample(pred, testingData$classe)[[1]]
1 - pr[[1]]
```

```
## [1] 0.006500127
```

Both methods of calculating the error are close, showing the Out of Sample Error to be: 0.65%. It seems like we are ready to use this model!

Predict on 20 cases

We now use our Random Forest model to generate the outcomes using the prediction data.

```
predictedValues <- predict(rfModel, validPredictData)
predictedValues
```

```
##  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
```

Conclusion

We refined the movement dataset to eliminate more than 100 low utility columns, created a Random Forest prediction model which demonstrated an extremely high accuracy rate, and used cross validation methods to verify the model using the `postResample()` and `confusionMatrix()` methods which confirmed the low error rate of this approach. Finally, we applied the Random Forest model to the prediction / validation data to generate the desired answers.