Prediction Assignment

Anand Rao

October 22, 2016

Executive Summary

The objective of this assignment is to analyze activity tracking data of six participants to determine how well they are performing certain activities. We build four different models using the training data and predict the values for the test data. Of the four methods used the Random Forest model has the best accuracy.

Background

Human Activity Recognition is a key area of study with the advent of activity tracking devices like *Jawbone*, *fitbit*, etc. Consumers wearing these devices often monitor their activity level, calories expended, number of miles walked etc. Most often these devices provide statistics on *how much* activity the person is doing; very rarely do they monitor *how well* they are doing these activities.

The goal of this project is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to monitor *how well* they do them. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways - exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

Once we read the data for this exercise we perform data cleansing and remove some of the variables that are close to zero and also variables that have a number of NAs. We then split the data into a training and test set. We fit four different types of models and then predict the values of the test data using these models to compare their accuracy. The four models we analyze are (a) Decision Trees; (b) Random Forest; (c) Gradient Boosting and (d) Support Vector Machine (SVM).

Data Loading and Exploratory Analysis

We first load all of the libraries that we will be using for this exercise. The data is read from the two URLs provided.We split the training data into two sets - 70% to build the model and the remaining 30% to test the model that we have built.

```
# Load all the requisite libraries
setwd("C:/Users/anand/Documents/Anand/PracticalMachineLearning(Coursera)")
library(knitr)
library(caret)
```

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

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

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

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(randomForest)
```

```
## 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
```

```
set.seed(12345)
```

```
# set the URL for the training and test data & read the data
TrainingDataURL <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
TestDataURL  <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

training <- read.csv(url(TrainingDataURL))
testing  <- read.csv(url(TestDataURL))

# Partition the Training dataset into two based on the classe variable
inTrain  <- createDataPartition(training$classe, p=0.7, list=FALSE)
Train1 <- training[inTrain, ]
Train2  <- training[-inTrain, ]
```

When we examine the data we see that there are a number of near zero values and NAs. We remove both these types of variables from the training data set to improve the model prediction. We also remove the first five columns which are identifiers and time stamps.

```
dim(Train1)
```

```
## [1] 13737   160
```

```
dim(Train2)
```

```
## [1] 5885  160
```

```
# Remove near zero variance variables from the training set
ZeroVars <- nearZeroVar(Train1)
Train1 <- Train1[, -ZeroVars]
Train2  <- Train2[, -ZeroVars]
dim(Train1)
```

```
## [1] 13737   106
```

```
dim(Train2)
```

```
## [1] 5885  106
```

```
# Remove NA variables from the training set
NAVars    <- sapply(Train1, function(x) mean(is.na(x))) > 0.95
Train1 <- Train1[, NAVars==FALSE]
Train2  <- Train2[, NAVars==FALSE]
dim(Train1)
```

```
## [1] 13737    59
```

```
dim(Train2)
```

```
## [1] 5885   59
```

```
# Remove identifier, user name and all time stamps from first 5 columns
Train1 <- Train1[, -(1:5)]
Train2  <- Train2[, -(1:5)]
dim(Train1)
```

```
## [1] 13737    54
```

```
dim(Train2)
```

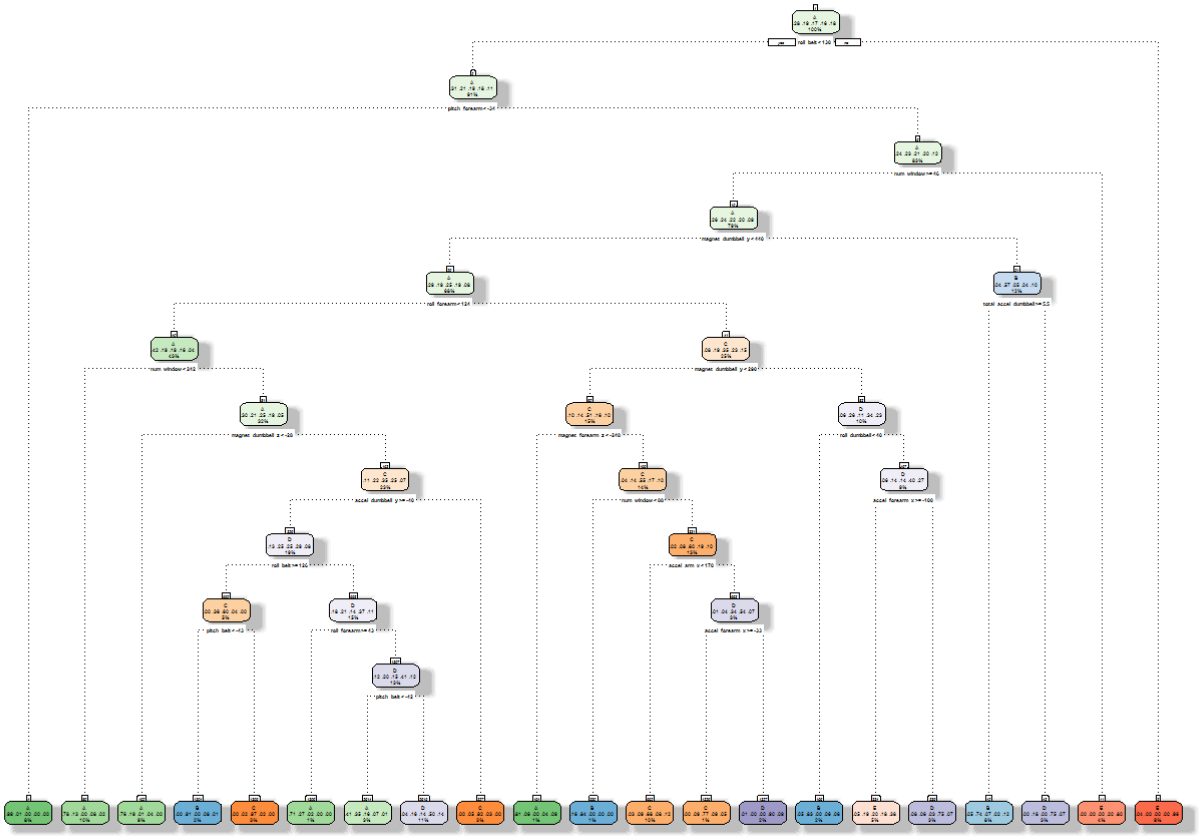
```
## [1] 5885   54
```

After the clean up process the number of variables were reduced from 106 variables to 54 variables.

Model: Decision Tree

We first build a decision tree model and plot it. The confusion matrix provides us with an analysis of the actual and predictived values.

```
set.seed(12345)
modDecisionTree <- rpart(classe ~ ., data=Train1, method="class")
fancyRpartPlot(modDecisionTree)
```



Rattle 2016-Oct-22 21:43:45 anand

```
# prediction on Test dataset
predictDecisionTree <- predict(modDecisionTree, newdata=Train2, type="class")
confusionMatrixDecisionTree <- confusionMatrix(predictDecisionTree, Train2$classe)
confusionMatrixDecisionTree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1530  269   51   79   16
##          B   35  575   31   25   68
##          C   17   73  743   68   84
##          D   39  146  130  702  128
##          E   53   76   71   90  786
##
## Overall Statistics
##
##                Accuracy : 0.7368
##                  95% CI : (0.7253, 0.748)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6656
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9140  0.50483   0.7242   0.7282   0.7264
## Specificity            0.9014  0.96650   0.9502   0.9100   0.9396
## Pos Pred Value         0.7866  0.78338   0.7543   0.6131   0.7305
## Neg Pred Value         0.9635  0.89051   0.9422   0.9447   0.9384
## Prevalence             0.2845  0.19354   0.1743   0.1638   0.1839
## Detection Rate         0.2600  0.09771   0.1263   0.1193   0.1336
## Detection Prevalence   0.3305  0.12472   0.1674   0.1946   0.1828
## Balanced Accuracy      0.9077  0.73566   0.8372   0.8191   0.8330
```

As shown by the details of the confusion matrix the accuracy of our decision tree model is around 73.68%.

Model: Random Forest

Next we build a random forest model. We use repeated cross-validation as the control.

```
set.seed(12345)
controlRF <- trainControl(method="repeatedcv", number=3)
modRandomForest <- suppressMessages(train(classe ~ ., data=Train1, method="rf",
                         trControl=controlRF))
modRandomForest$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: 27
##
##         OOB estimate of  error rate: 0.2%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    1    0    0    1 0.0005120328
## B    5 2652    1    0    0 0.0022573363
## C    0    5 2390    1    0 0.0025041736
## D    0    0    7 2245    0 0.0031083481
## E    0    1    0    5 2519 0.0023762376
```

```
# prediction on Test dataset
predictRandomForest <- predict(modRandomForest, newdata=Train2)
confusionMatrixRandomForest <- confusionMatrix(predictRandomForest, Train2$classe)
confusionMatrixRandomForest
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    5    0    0    0
##          B    0 1133    4    0    0
##          C    0    1 1022    8    0
##          D    0    0    0  956    3
##          E    0    0    0    0 1079
##
## Overall Statistics
##
##                Accuracy : 0.9964
##                  95% CI : (0.9946, 0.9978)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9955
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9947   0.9961   0.9917   0.9972
## Specificity            0.9988   0.9992   0.9981   0.9994   1.0000
## Pos Pred Value         0.9970   0.9965   0.9913   0.9969   1.0000
## Neg Pred Value         1.0000   0.9987   0.9992   0.9984   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1925   0.1737   0.1624   0.1833
## Detection Prevalence   0.2853   0.1932   0.1752   0.1630   0.1833
## Balanced Accuracy      0.9994   0.9969   0.9971   0.9955   0.9986
```

As shown by the details of the confusion matrix the accuracy of our random forest model is 99.64%.

Model: Gradient Boosting

Next we build a gradient boosting model using repeated cross-validation as the control. We then predict the values for our test data using the model.

```
set.seed(12345)
controlGBM <- trainControl(method = "repeatedcv", number = 3)
modGradientBoosting  <- suppressMessages(train(classe ~ ., data=Train1, method = "gbm",
                    trControl = controlGBM,
                    verbose = FALSE))
modGradientBoosting$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 53 predictors of which 42 had non-zero influence.
```

```
# prediction on Test dataset
predictGradientBoosting <- predict(modGradientBoosting, newdata=Train2)
confusionMatrixGradientBoosting <- confusionMatrix(predictGradientBoosting, Train2$classe)
confusionMatrixGradientBoosting
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672   13    0    2    0
##          B    2 1114   16    4    1
##          C    0   12 1006   11    3
##          D    0    0    2  947   11
##          E    0    0    2    0 1067
##
## Overall Statistics
##
##                Accuracy : 0.9866
##                  95% CI : (0.9833, 0.9894)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.983
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9781   0.9805   0.9824   0.9861
## Specificity            0.9964   0.9952   0.9946   0.9974   0.9996
## Pos Pred Value         0.9911   0.9798   0.9748   0.9865   0.9981
## Neg Pred Value         0.9995   0.9947   0.9959   0.9965   0.9969
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1893   0.1709   0.1609   0.1813
## Detection Prevalence   0.2867   0.1932   0.1754   0.1631   0.1816
## Balanced Accuracy      0.9976   0.9866   0.9876   0.9899   0.9929
```

As shown by the details of the confusion matrix the accuracy of our random forest model is 98.66%.

Model: Support Vector Machine (SVM)

Next we build a SVM model using repeated cross-validation as the control and use the SVMRadial method. We then predict the values for our test data using the model.

```
set.seed(12345)
controlSVM <- trainControl(method = "repeatedcv", number = 5,
                                                        classProbs=TRUE)
modSVM  <- suppressMessages(train(classe ~ ., data=Train1,
                 method = "svmRadial",
                 tuneLength = 9,
                 preProc = c("center","scale"),
                 metric="ROC",
                 trControl = controlSVM))
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "ROC" was not
## in the result set. Accuracy will be used instead.
```

```
modSVM$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 64
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0135443145643756
##
## Number of Support Vectors : 3008
##
## Objective Function Value : -7000.732 -1610.012 -1408.205 -608.4104 -4519.612 -998.6264 -1387.
869 -12722.44 -2089.175 -2756.327
## Training error : 0.002693
## Probability model included.
```

```
# prediction on Test dataset
predictSVM <- predict(modSVM, newdata=Train2)
confusionMatrixSVM <- confusionMatrix(predictSVM, Train2$classe)
confusionMatrixSVM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    6    5   74   53
##          B    2 1101   47   76   84
##          C    0   31  911  140  141
##          D    0    0   63  439   46
##          E    0    1    0  235  758
##
## Overall Statistics
##
##                Accuracy : 0.8294
##                  95% CI : (0.8195, 0.8389)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7832
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9666   0.8879  0.45539   0.7006
## Specificity            0.9672   0.9560   0.9358  0.97785   0.9509
## Pos Pred Value         0.9238   0.8405   0.7449  0.80109   0.7626
## Neg Pred Value         0.9995   0.9917   0.9753  0.90163   0.9338
## Prevalence             0.2845   0.1935   0.1743  0.16381   0.1839
## Detection Rate         0.2841   0.1871   0.1548  0.07460   0.1288
## Detection Prevalence   0.3076   0.2226   0.2078  0.09312   0.1689
## Balanced Accuracy      0.9830   0.9613   0.9119  0.71662   0.8257
```

As shown by the details of the confusion matrix the accuracy of our SVM model is 82.94%.

Conclusion

Of the four models we built the accuracy is highest for the Random Forest model. We will use this model to predict the values of any future data including the 20 test cases to be used as part of the assignment. The least accurate model is the Decision Tree model.

```
predictTestingData <- predict(modRandomForest, newdata=testing)
predictTestingData
```

```
##  [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
```

The high accuracy of the Random Forest model leads one to believe that the data might have been specially built for the assignment. The accuracy is likely to be much lower for most real world problems.