Using Activity Tracker Data to Study Exercise Techniques

Aditya Iyengar

15/04/2020

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. 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, we have with us data from accelerometers on the belt, forearm, arm, and dumb-bell of 6 participants. They were asked to perform bar-bell lifts correctly and incorrectly in 5 different ways. More information is available from the website here (see the section on the Weight Lifting Exercise Dataset).

Objective

The goal of the project is to predict the manner in which the people did the exercise. This is the "classe" variable in the training set. The prediction model will be used to predict 20 different test cases.

Preliminaries

Let us load the training and test datasets.

```
training <- read.csv("pml-training.csv")
test <- read.csv("pml-testing.csv")
```

Here are the dimensions of the training dataset.

```
dim(training)
```

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

We observe that there are several columns with majorly missing observations. We will eject these columns from the training and test sets. Also, the first seven rows contain the information about the people who participated in the tests, which is not relevant to the task. Hence, we eject these rows.

```
missingindex <- which(colSums(is.na(training) |training=="")>0.9*dim(training)[1])
training <- training[,-missingindex]
training <- training[,-c(1:7)]
missingindex2 <- which(colSums(is.na(test) |test=="")>0.9*dim(test)[1])
test <- test[, -missingindex2]
test <- test[, -c(1:7)]
```

Let us now view a brief summary of the training dataset.

```r
str(training)
```

```
## 'data.frame':    19622 obs. of  53 variables:
##  $ roll_belt           : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt    : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ gyros_belt_x        : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y        : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z        : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x        : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y        : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z        : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x       : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y       : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z       : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm           : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ gyros_arm_x         : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y         : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z         : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x         : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y         : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z         : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x        : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y        : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z        : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ roll_dumbbell       : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell      : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell        : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 0 0 -0.02 0 0 0 0 0 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48 ...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558 -551 -555 -549 -558 ...
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295 300 292 291 ...
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70 -74 -65 -69 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9 27.9 27.8 27.7 27.7 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -152 -152 -152 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.02 0.02 0.02 0.03 0.02 ...
##  $ gyros_forearm_y     : num  0 0 -0.02 -0.02 0 -0.02 0 -0.02 0 0 ...
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03 -0.02 0 -0.02 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195 193 193 190 ...
##  $ accel_forearm_y     : int  203 203 204 206 206 203 205 205 204 205 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215 ...
```

```
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473 ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

We now partition our training dataset into separate training and test datasets.

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
set.seed(11111)
inTrain <- createDataPartition(training$classe, p=0.7, list=FALSE)
traindata <- training[inTrain,]
testdata <- training[-inTrain,]
dim(traindata)
```

```
## [1] 13737    53
```

```r
dim(testdata)
```

```
## [1] 5885   53
```

Training

We use **Decision Trees**, **Random Forests**, **Gradient Boosting** and **Support Vector Machines** as the training algorithms. We compare the accuracies on the test component of the training data and choose the best method to be applied on the test data. Across the algorithms, we use **10-fold cross-validation** to limit the effects of overfitting and improve the efficiency of the models.

Decision Tree

```r
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

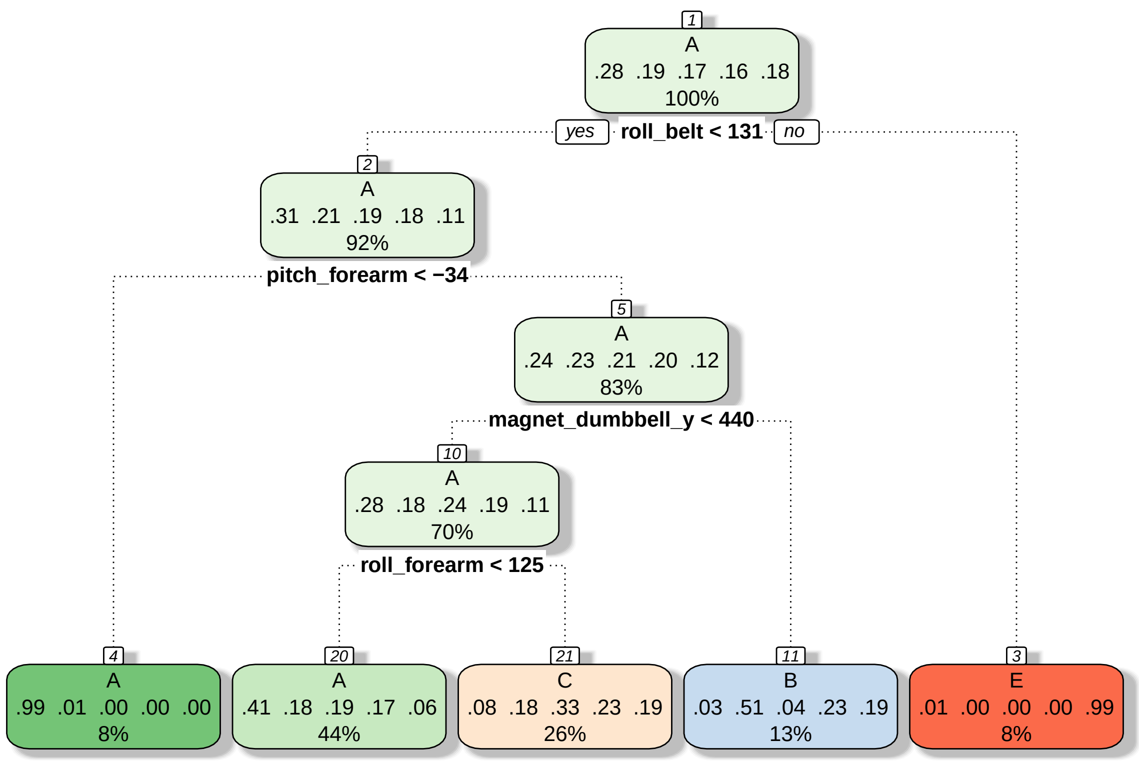
```r
crossvalidation <- trainControl(method="cv", number=5)
model_dt <- train(classe ~ ., data = traindata, method = "rpart", trControl = crossvalidation)
pred_dt <- predict(model_dt, newdata = testdata)
conf_dt <- confusionMatrix(testdata$classe, pred_dt)
conf_dt$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1523   22  123    0    6
##          B  497  375  267    0    0
##          C  453   33  540    0    0
##          D  454  157  353    0    0
##          E  155  155  296    0  476
```

```
conf_dt$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.4951572      0.3398960      0.4823034      0.5080158      0.5237043
## AccuracyPValue  McnemarPValue
##      0.9999945            NaN
```

```
fancyRpartPlot(model_dt$finalModel)
```



Rattle 2020−Jul−25 22:25:54 Aditya Iyengar

The accuracy obtained is **under 50%** and is extremely unsatisfactory. It would not be acceptable to use this model for further predictions.

Random Forest

```r
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:ggplot2':
##
##     margin
```

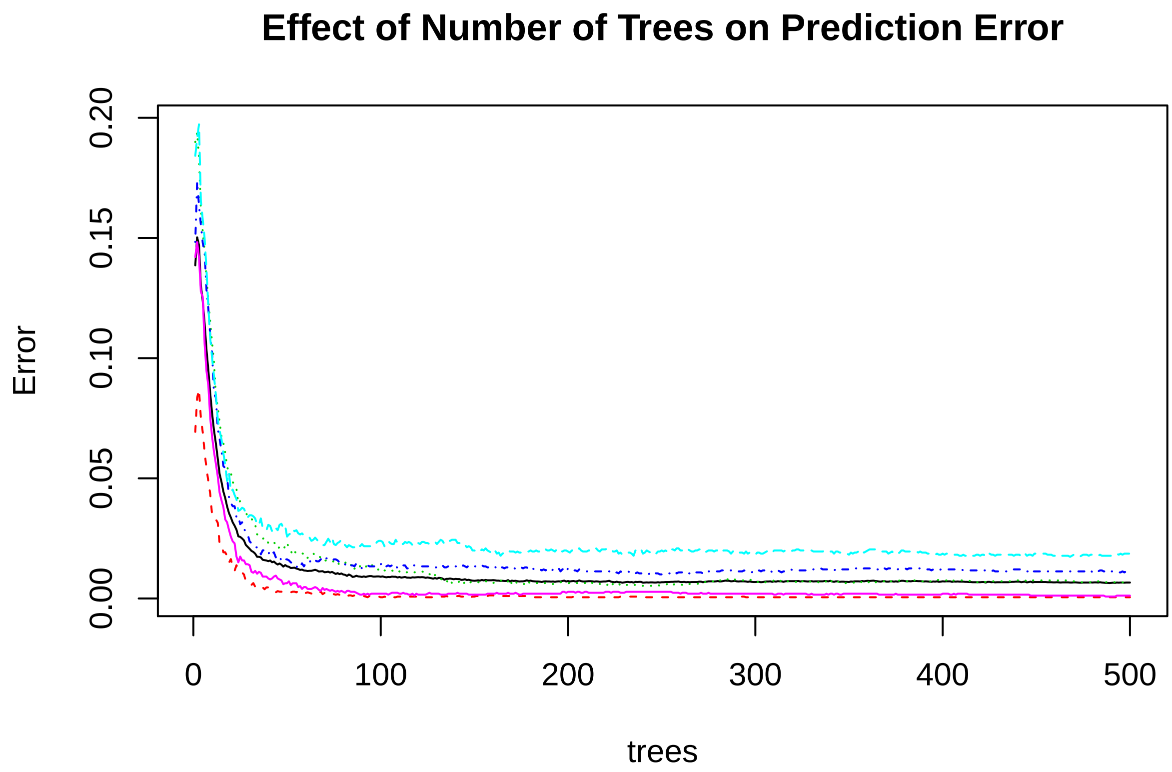
```r
model_rf <- train(classe ~ ., data = traindata, method = "rf", trControl = crossvalidation, verbose = F
pred_rf <- predict(model_rf, testdata)
conf_rf <- confusionMatrix(testdata$classe, pred_rf)
conf_rf$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    7 1132    0    0    0
##          C    0    9 1017    0    0
##          D    0    0   14  950    0
##          E    0    0    0    3 1079
```

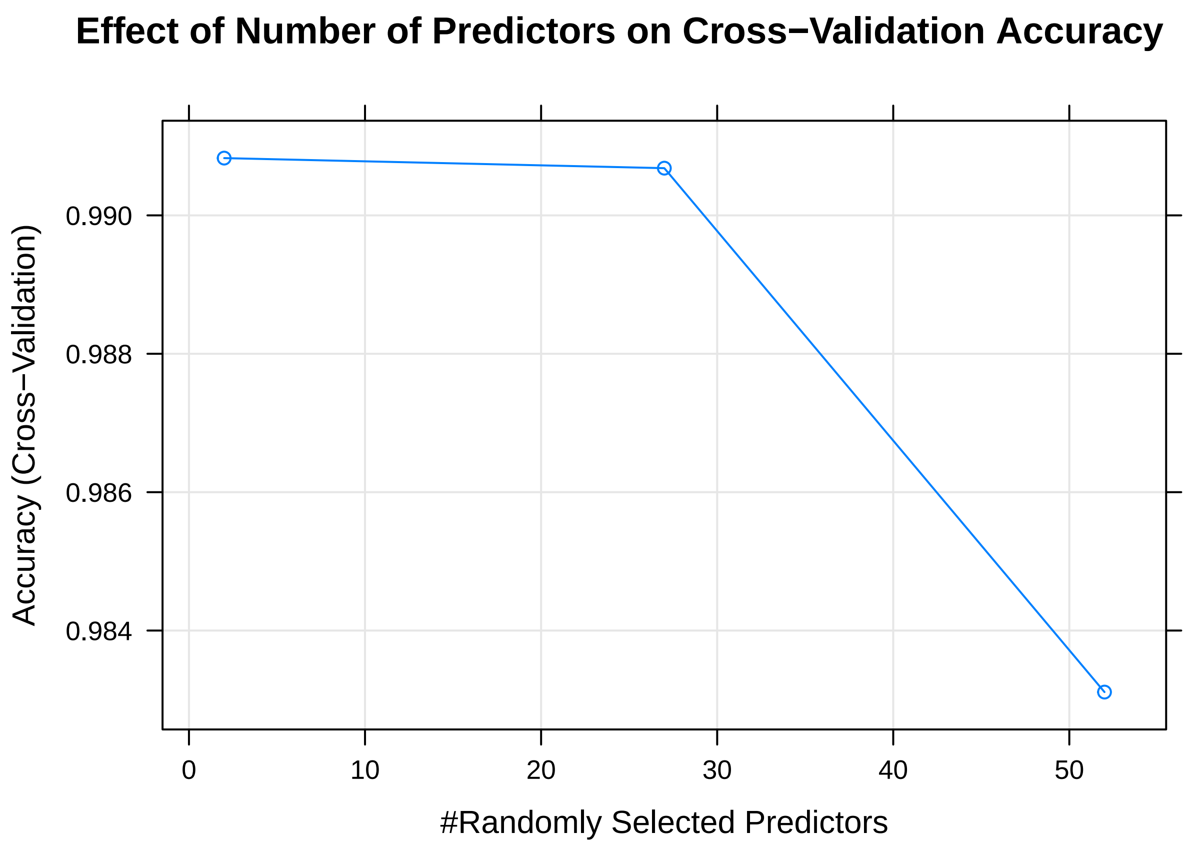
```r
conf_rf$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9943925      0.9929059      0.9921340      0.9961370      0.2856415
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```r
plot(model_rf$finalModel, main="Effect of Number of Trees on Prediction Error")
```




```r
plot(model_rf, main="Effect of Number of Predictors on Cross-Validation Accuracy")
```



The accuracy obtained is **over 99%**, hence this appears to be an excellent predictor.

We also see that the error in the model reduces rapidly until the number of trees are around 30. Increasing the number of trees further doesn't have a major increment in the accuracy.

The optimal number of predictors can be anywhere from 2 to around 25. Increasing the number of predictors further reduces the accuracy, suggesting that there may be interdependent predictors that may be strongly correlated.

Gradient Boosting

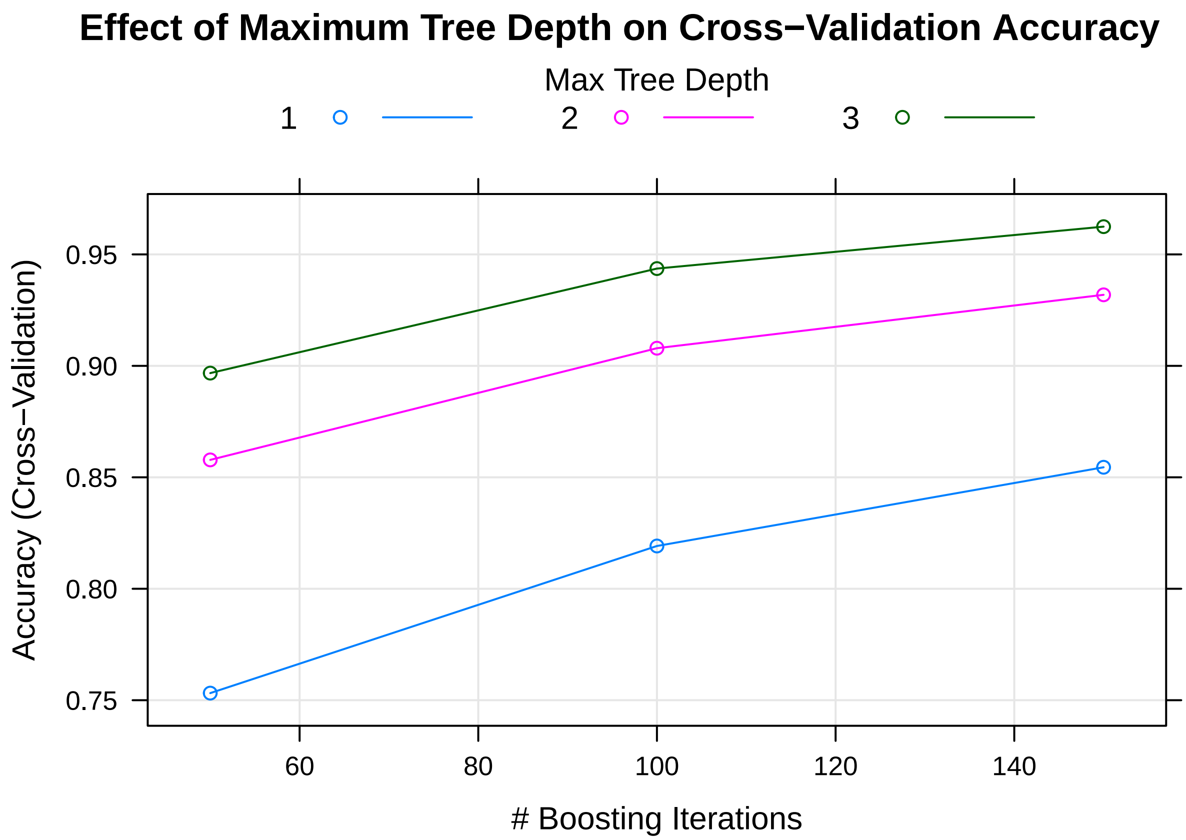
```
model_gb <- train(classe ~ ., data = training, method = "gbm", trControl = crossvalidation, verbose=FALS
pred_gb <- predict(model_gb, testdata)
conf_gb <- confusionMatrix(testdata$classe, pred_gb)
conf_gb$table
```

```
##             Reference
## Prediction    A    B    C    D    E
##           A 1658   14    0    2    0
##           B   25 1093   21    0    0
##           C    0   20  991   13    2
##           D    1    2   21  939    1
##           E    2   10    9   14 1047
```

```
conf_rf$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9943925      0.9929059      0.9921340      0.9961370      0.2856415
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
plot(model_gb, main = "Effect of Maximum Tree Depth on Cross-Validation Accuracy")
```



The accuracy obtained with a maximum tree depth of 3 and 10-fold cross validation is **around 96%**, which is pretty good, but inferior to that obtained by the Random Forest.

Support Vector Machine

```
library(e1071)
model_svm <- svm(classe~., data = traindata, type = "C-classification", kernel = "radial", cost = 5, cr
pred_svm <- predict(model_svm, testdata)
conf_svm <- confusionMatrix(testdata$classe, pred_svm)
conf_svm$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1669    4    1    0    0
```

```
##           B   22 1112    3    0    2
##           C    1   12 1011    2    0
##           D    0    0   58  906    0
##           E    0    0    5   14 1063
```

```r
conf_svm$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9789295      0.9733368      0.9749289      0.9824449      0.2875106
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

The accuracy obtained is **around 80%**, which, while not dismal by itself, stands no chance against the accuracy of the Random Forest model.
Thus we choose the **Random Forest** algorithm for the final testing.

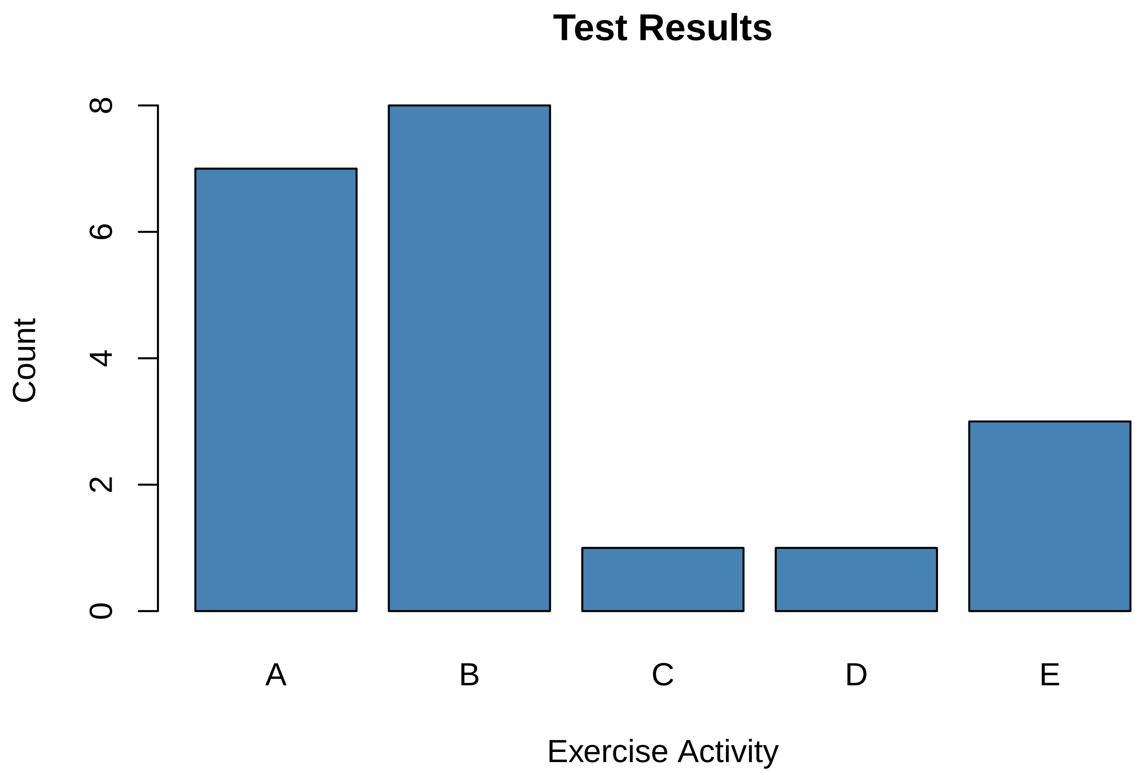
Testing

We test our Random Forest model against the unseen test data that consists of 20 rows of observations.

```r
predict(model_rf, test)
```

```
##  [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
```

```r
plot(predict(model_rf, test), main = "Test Results", ylab = "Count", xlab = "Exercise Activity", col =
```



Interestingly, our SVM and GB models also give identical results as can be seen below. However the results of the Decision tree are quite some way off.

```r
predict(model_svm, test)
```

```
##  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
```

```r
predict(model_gb, test)
```

```
##  [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
```

```r
predict(model_dt, test)
```

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

Conclusion

Random forests are extremely accurate predictors and can be used to model large volumes of data. The given test samples appear to be conservative as the results from the lesser accurate Gradient Boosting and

SVM algorithms mimic those from the highly accurate Random Forest.
10-fold cross-validation was used to minimize the effect of overfitting without compromising on the length of the datasets. Often, it may be advisable to omit covariates that are highly correlated in favour of obtaining better accuracies.

References

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.