Practical Machine Learning Final Project

Shaopeng Lee

7/19/2020

Overview

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 will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

The data consists of a Training data and a Test data (to be used to validate the selected model).

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with.

Note: The dataset used in this project is a courtesy of "Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements"

Data Loading and Processing

```
testing<-read.csv("/Users/shorpen/Downloads/pml-testing.csv",na = c("", "NA"))
training<-read.csv("/Users/shorpen/Downloads/pml-training.csv",na = c("", "NA"))
dim(testing)
```

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

```
dim(training)
```

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

```
validData<- testing[, colSums(is.na(testing) )== 0]
#removeColumns <- which(colSums(is.na(training.raw) | training.raw=="") > maxNACount)
trainData<- training[, colSums(is.na(training)) == 0]
dim(trainData)
```

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

```
dim(validData)
```

```
## [1] 20 60
```

```
validData <- validData[, -c(1:7)]
trainData <- trainData[, -c(1:7)]
```

As shown in the above, there are 19622 observations and 160 variables in the Training dataset. We also observe variables that contains missing values, so we need to exclude those columns. Also, we further remove the first seven variables as they have little impact on the outcome. For now, we have two datasets, which are a training dataset used for the model training and the other is to valid the medel performance. The final input for the model traning contains r dim(trainData)[2]‘ variables after the feature selection.

Prepare the datasets for model training

In this part, we will first split the training data into 70% as the real train data and 30% as test data. This process is aimed to compute the out-of-sample errors in case of the biased model. The valid dataset will remain as it is and will be used to test the predication algorithm.

```
set.seed(1234)
library(caret)
```

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

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

```
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
training <- trainData[inTrain, ]
testing <- trainData[-inTrain, ]
dim(training)
```

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

```
dim(testing)
```

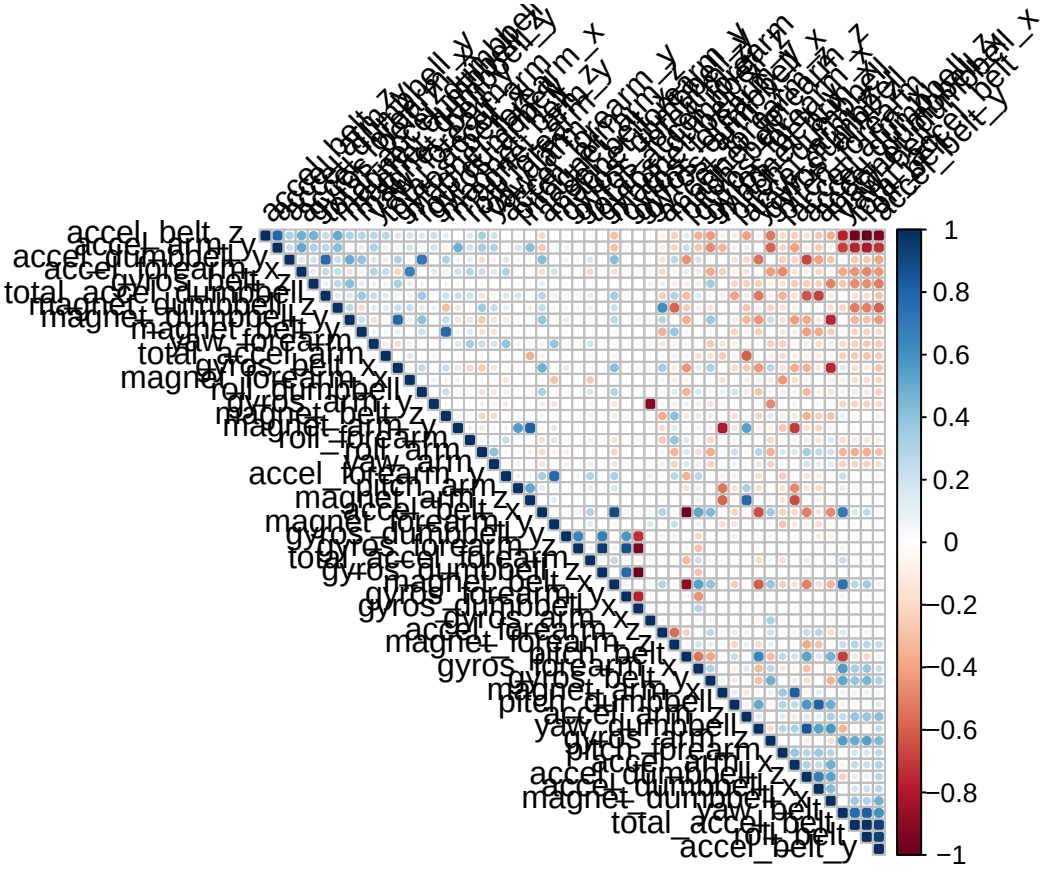
```
## [1] 5885   53
```

We pick out 53 variables and then compute the correlation paramaters. Then, we use the correlation plot to display the relation between the variables.

```
res <- cor(training[,-53])
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
corrplot(res, type = "upper", order = "FPC", tl.col = "black", tl.srt = 45)
```



The correlated predictors are those with a dark color. The blue means the postive and the red is negative. We use the findCorrelation function to search for highly correlated attributes with a cuttoff 0.75. The below is the names of highly correlated factors:

```
highlyCorrelated = findCorrelation(res, cutoff=0.75)
names(training)[highlyCorrelated]
```

```
##  [1] "accel_belt_z"      "roll_belt"         "accel_belt_y"
##  [4] "total_accel_belt"  "accel_dumbbell_z"  "accel_belt_x"
##  [7] "pitch_belt"        "magnet_dumbbell_x" "accel_dumbbell_y"
## [10] "magnet_dumbbell_y" "accel_dumbbell_x"  "accel_arm_x"
## [13] "accel_arm_z"       "magnet_arm_y"      "magnet_belt_z"
## [16] "accel_forearm_y"   "gyros_forearm_y"   "gyros_dumbbell_x"
## [19] "gyros_dumbbell_z"  "gyros_arm_x"
```

Model Building

For this project, we are aimed to predict a class factor, so we will choose three different algorithms, which are classification trees, random forests and a generalized boosted model, to predict the outcome. 1.Classification trees 2.Random forests 3.Generalized Boosted Model

Selection 1: Classification Trees

```
set.seed(12345)
library(rpart)
library(rattle)
```

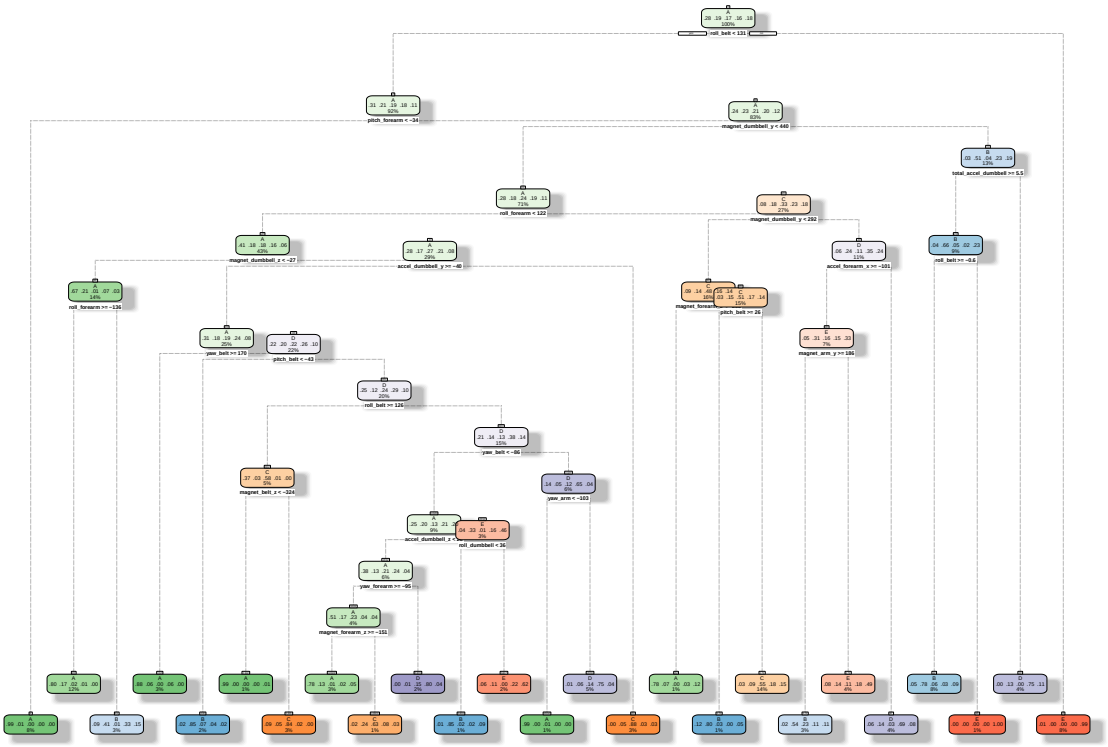
```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(e1071)
decisionTreeModel <- rpart(classe ~., data=training, method="class")
fancyRpartPlot(decisionTreeModel)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



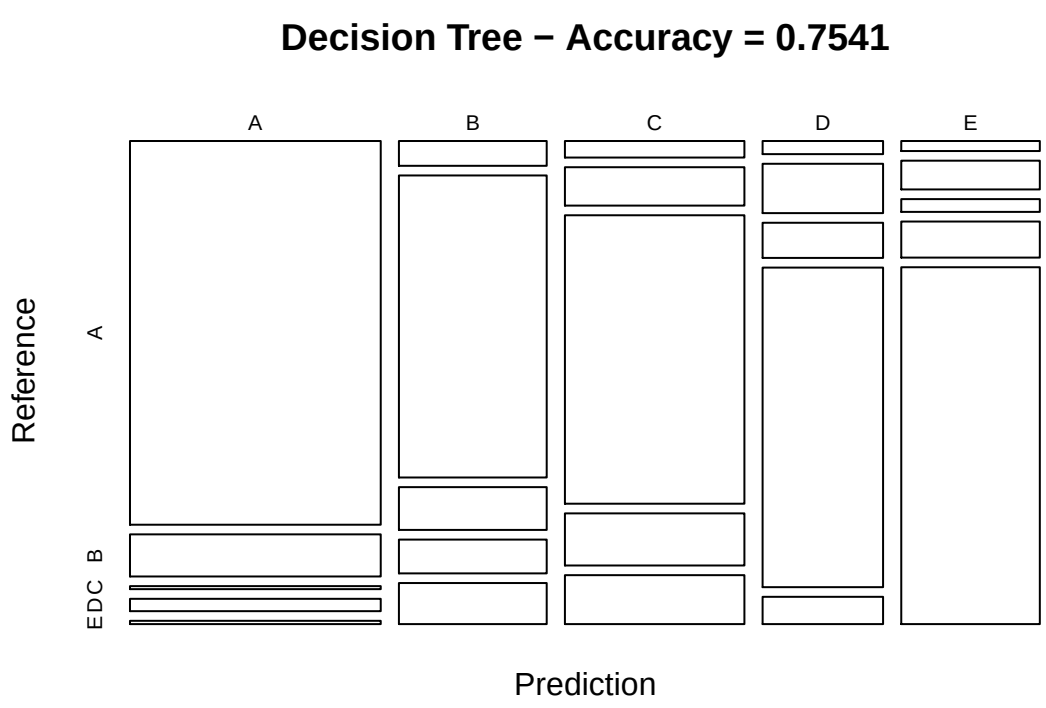

```
predictTressModel <- predict(decisionTreeModel,testing,type="class")
cmtree <- confusionMatrix(predictTressModel, testing$classe)
```

In the first part, we first fit the model and use the fancyRpartPlot() function to plot the classification tree. Then we validate the model on the test dataset to find out how well it performs by computing the accuracy.

```
cmtree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1522  167   12   49   13
##          B   58  706  100   79   96
##          C   47  109  819  148  139
##          D   25   94   67  609   52
##          E   22   63   28   79  782
##
## Overall Statistics
##
##                Accuracy : 0.7541
##                  95% CI : (0.7429, 0.7651)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6885
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9092   0.6198   0.7982   0.6317   0.7227
## Specificity            0.9428   0.9298   0.9088   0.9516   0.9600
## Pos Pred Value         0.8633   0.6795   0.6490   0.7190   0.8029
## Neg Pred Value         0.9631   0.9106   0.9552   0.9295   0.9389
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2586   0.1200   0.1392   0.1035   0.1329
## Detection Prevalence   0.2996   0.1766   0.2144   0.1439   0.1655
## Balanced Accuracy      0.9260   0.7748   0.8535   0.7917   0.8414
```

```
plot(cmtree$table, col = cmtree$byClass,
     main = paste("Decision Tree - Accuracy =", round(cmtree$overall['Accuracy'], 4)))
```



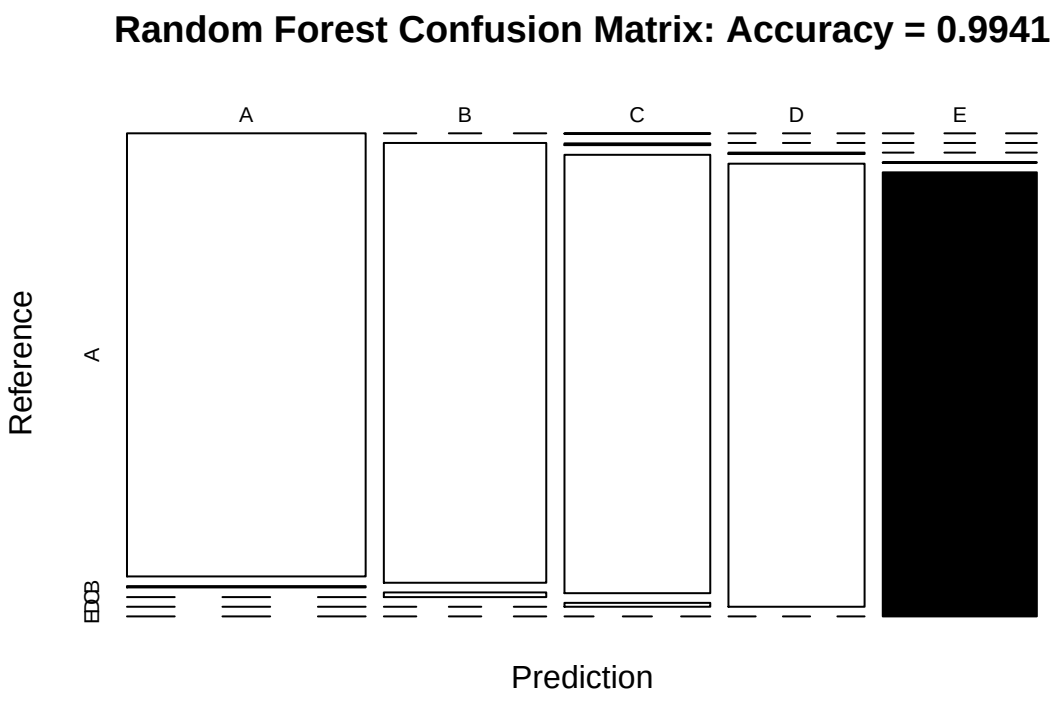
We can see the accuracy rate of the Classification Trees is 0.7541206, which is a bit low. ### Selection 2: Random Forest

```
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
library(caret)
modFit <- train(classe ~., data=training, method="rf",trControl=controlRF)
#modFit$finalModel
predictRF1 <- predict(modFit, newdata=testing)
cmrf <- confusionMatrix(predictRF1, testing$classe)
cmrf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    5    0    0    0
##          B    0 1130   12    0    0
##          C    1    4 1011    9    0
##          D    0    0    3  954    0
##          E    0    0    0    1 1082
##
## Overall Statistics
##
##                Accuracy : 0.9941
##                  95% CI : (0.9917, 0.9959)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9925
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9921   0.9854   0.9896   1.0000
## Specificity            0.9988   0.9975   0.9971   0.9994   0.9998
## Pos Pred Value         0.9970   0.9895   0.9863   0.9969   0.9991
## Neg Pred Value         0.9998   0.9981   0.9969   0.9980   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1920   0.1718   0.1621   0.1839
## Detection Prevalence   0.2851   0.1941   0.1742   0.1626   0.1840
## Balanced Accuracy      0.9991   0.9948   0.9912   0.9945   0.9999
```

```
plot(cmrf$table, col = cmrf$byClass, main = paste("Random Forest Confusion Matrix: Accuracy =", round(cm
```



As same in the above model, we fit a random forest model this time. Also, we set up 3-fold cross validation for training. Then, we validate the model on the testing with an accuray,0.9940527 under 3-fold cross validation. It is fairly high, but it might be due to overfitting.

Selection 3: Generalized Boosted Regression Models

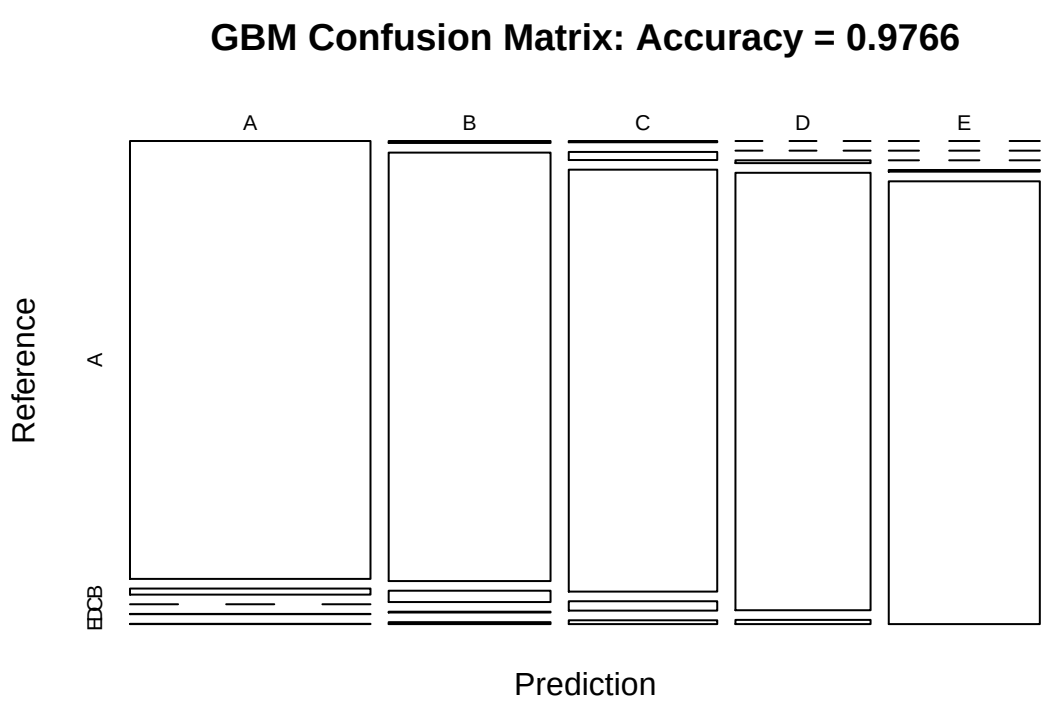
```
set.seed(12345)
controlGBM <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
modGBM  <- train(classe ~ ., data=trainData, method = "gbm", trControl = controlGBM, verbose = FALSE)
modGBM$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 52 predictors of which 51 had non-zero influence.
```

```
predictGBM <- predict(modGBM, newdata=testing)
cmGBM <- confusionMatrix(predictGBM, testing$classe)
cmGBM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1666   23    0    1    1
##          B    5 1097   29    2    5
##          C    3   19  991   22    9
##          D    0    0    6  935    9
##          E    0    0    0    4 1058
##
## Overall Statistics
##
##                Accuracy : 0.9766
##                  95% CI : (0.9724, 0.9803)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9703
##
##  Mcnemar's Test P-Value : 1.611e-06
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9952   0.9631   0.9659   0.9699   0.9778
## Specificity            0.9941   0.9914   0.9891   0.9970   0.9992
## Pos Pred Value         0.9852   0.9640   0.9492   0.9842   0.9962
## Neg Pred Value         0.9981   0.9912   0.9928   0.9941   0.9950
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2831   0.1864   0.1684   0.1589   0.1798
## Detection Prevalence   0.2873   0.1934   0.1774   0.1614   0.1805
## Balanced Accuracy      0.9946   0.9772   0.9775   0.9834   0.9885
```

```
plot(cmGBM$table, col = cmGBM$byClass, main = paste("GBM Confusion Matrix: Accuracy =", round(cmGBM$ove
```



This time, we fit a gradient boosted model. Applying the same process, we got an accuracy of 0.9765506 under 5-fold cross validation. The out-of-sample error is .

Model Validation

After the comparison, we pick out the best fitted model to apply it on the validation data. The below result is our predication based on the Random Forest model.

```
Results <- predict(modFit, newdata=validData)
Results
```

```
##  [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 gbm model can also be used for prediction:

```
predGbmTest <- predict(modGBM, newdata = validData)
table(Results, predGbmTest)
```

```
##         predGbmTest
## Results A B C D E
##       A 7 0 0 0 0
##       B 0 8 0 0 0
##       C 0 0 1 0 0
##       D 0 0 0 1 0
##       E 0 0 0 0 3
```

The two models produce the same results, as shown in the confusion matrix above.