Prediction Assignment Writeup

Yugandhar Patil

10th Nov 2023

### Objective

The purpose of this project was to quantify how well the participants performed a barbell lifting exercise and to classify the measurement read from an accelerometer into 5 different classes (Class A:Class E).

Please reference the links below for the data sources:

<http://groupware.les.inf.puc-rio.br/har>

<https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv>

<https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv>

#### Install/load the required packages needed for the creation of the model

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(rpart)  
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.4.4

## 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

#### Load the training and testing datasets

train<-read.csv("C:/Users/aao1009/Desktop/pml-training.csv",na.strings=c("NA","#DIV/0!",""))  
test<-read.csv("C:/Users/aao1009/Desktop/pml-testing.csv",na.strings=c("NA","#DIV/0!",""))

#### Remove null columns and the first 7 columns that will not be used

test\_clean <- names(test[,colSums(is.na(test)) == 0]) [8:59]  
clean\_train<-train[,c(test\_clean,"classe")]  
clean\_test<-test[,c(test\_clean,"problem\_id")]

#### Check the dimensions of the clean test and train sets

dim(clean\_test)

## [1] 20 53

dim(clean\_train)

## [1] 19622 53

#### Split the data into the training and testing datasets

set.seed(100)  
inTrain<-createDataPartition(clean\_train$classe, p=0.7, list=FALSE)  
training<-clean\_train[inTrain,]  
testing<-clean\_train[-inTrain,]  
dim(training)

## [1] 13737 53

dim(testing)

## [1] 5885 53

### Predicting the outcome using 3 different models

#### LDA Model

lda\_model<-train(classe~ ., data=training, method="lda")  
set.seed(200)  
predict<-predict(lda\_model,testing)  
confusionMatrix(predict,testing$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 1379 186 107 53 37  
## B 32 703 95 40 189  
## C 137 160 686 121 106  
## D 122 40 115 705 113  
## E 4 50 23 45 637  
##   
## Overall Statistics  
##   
## Accuracy : 0.6984   
## 95% CI : (0.6865, 0.7101)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6182   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.8238 0.6172 0.6686 0.7313 0.5887  
## Specificity 0.9090 0.9250 0.8922 0.9207 0.9746  
## Pos Pred Value 0.7826 0.6638 0.5669 0.6438 0.8393  
## Neg Pred Value 0.9285 0.9097 0.9273 0.9459 0.9132  
## Prevalence 0.2845 0.1935 0.1743 0.1638 0.1839  
## Detection Rate 0.2343 0.1195 0.1166 0.1198 0.1082  
## Detection Prevalence 0.2994 0.1799 0.2056 0.1861 0.1290  
## Balanced Accuracy 0.8664 0.7711 0.7804 0.8260 0.7817

The LDA model gave a 70% accuracy on the testing set, with the expected out of sample error around 30%.

#### Decision Tree Model

decision\_tree\_model<-rpart(classe~ ., data=training,method="class")  
set.seed(300)  
predict<-predict(decision\_tree\_model,testing,type="class")  
confusionMatrix(predict,testing$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 1472 275 34 115 42  
## B 50 624 60 19 62  
## C 44 104 847 146 121  
## D 59 69 60 590 53  
## E 49 67 25 94 804  
##   
## Overall Statistics  
##   
## Accuracy : 0.737   
## 95% CI : (0.7255, 0.7482)  
## 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 0.8793 0.5478 0.8255 0.6120 0.7431  
## Specificity 0.8893 0.9598 0.9146 0.9510 0.9511  
## Pos Pred Value 0.7595 0.7656 0.6712 0.7100 0.7738  
## Neg Pred Value 0.9488 0.8984 0.9613 0.9260 0.9426  
## Prevalence 0.2845 0.1935 0.1743 0.1638 0.1839  
## Detection Rate 0.2501 0.1060 0.1439 0.1003 0.1366  
## Detection Prevalence 0.3293 0.1385 0.2144 0.1412 0.1766  
## Balanced Accuracy 0.8843 0.7538 0.8701 0.7815 0.8471

The Decision Tree Model gave a 74% accuracy on the testing set, with the expected out of sample error around 26%.

#### Random Forest Model

random\_forest\_mod<-randomForest(classe~ ., data=training, ntree=500)  
set.seed(300)  
predict<-predict(random\_forest\_mod, testing, type ="class")  
confusionMatrix(predict,testing$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 1673 7 0 0 0  
## B 1 1131 3 0 0  
## C 0 1 1021 9 1  
## D 0 0 2 955 1  
## E 0 0 0 0 1080  
##   
## Overall Statistics  
##   
## Accuracy : 0.9958   
## 95% CI : (0.9937, 0.9972)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9946   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9994 0.9930 0.9951 0.9907 0.9982  
## Specificity 0.9983 0.9992 0.9977 0.9994 1.0000  
## Pos Pred Value 0.9958 0.9965 0.9893 0.9969 1.0000  
## Neg Pred Value 0.9998 0.9983 0.9990 0.9982 0.9996  
## Prevalence 0.2845 0.1935 0.1743 0.1638 0.1839  
## Detection Rate 0.2843 0.1922 0.1735 0.1623 0.1835  
## Detection Prevalence 0.2855 0.1929 0.1754 0.1628 0.1835  
## Balanced Accuracy 0.9989 0.9961 0.9964 0.9950 0.9991

The Random Forest Model gave a 99.6% accuracy on the testing set, with the expected out of sample error around 0.4%.

### Conclusion

The greatest accuracy was achieved using the Random Forest Model, which gave an accuracy of 99.6%. Hence, this model was further used to make predictions on the exercise performance for 20 participants.

predict<-predict(random\_forest\_mod, clean\_test, type ="class")  
predict

## 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