Practical Machine Learning Project: Classification of Exercises Quality

MARIANO MOLINA GARCIA

14/06/2020

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.

In this project, the data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants will be used to quantify how well the exercise is done. The goal of your project is to predict the manner in which they did the exercise.

People were asked to perform barbell lifts correctly and incorrectly in 5 different ways, and the data previously commented have been stored in the file https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv. The data for this project come from this source: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har. More information about the data is available in that website, especifically in the section on the Weight Lifting Exercise Dataset.

Building the model. Data Analysis and selection of features.

The dataset contains 160 features, which can be used to predict if the exercise has been done properly or not. Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl. The way the exercise has been done is labelled using de "classe" variable in the dataset. This "classe" variable can have 5 different values: A,B,C,D and E, which correspond to five different fashions: 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). Therefore, class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

First of all, an initial analysis of the data havve been carried out. This analysis consists of:

1.- Load data. Checking the features and its range of values.

```r
library(caret)
library(AppliedPredictiveModeling)
library(dplyr)
trainingData <- read.csv("pml-training.csv")
testingData<- read.csv("pml-testing.csv")
str(trainingData)
str(testingData)
```

The number of features is 160. The number of observations for the training dataset is 19622 and the number of observations for the test dataset is 20.

```r
dim(trainingData)
```

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

```
dim(testingData)
```

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

2.- Checking if there is feature with wrong NA values. If the percentage is very high, remove the column. If it is lower, analyze the possibilities to impute values. After this process, the number of features have been reduced to 93.

```
#Check if there are many rows of any variable with values NA
na_count <-sapply(trainingData, function(y) sum(length(which(is.na(y)))))
na_count <- data.frame(na_count)
perc_na <- data.frame(100*(na_count/nrow(trainingData)))
#Remove that cols with many NA from the analysis
trainingData<-trainingData[,perc_na<90]
testingData<-testingData[,perc_na<90]
dim(trainingData)
```

```
## [1] 19622    93
```

```
dim(testingData)
```

```
## [1] 20 93
```

3.- Checking if there is feature with empty values. If the percentage is very high, remove the column. If it is lower, analyze the possibilities to impute values.After this process, the number of features have been reduced to 60.

```
pat <- "^[[:space:]]*$"
matches <-sapply(trainingData, function(x) grepl(pat, x))
matches<- data.frame(matches)
val_count_empty <-sapply(matches, function(y) sum(length(which(y==TRUE))))
val_count_empty<- data.frame(val_count_empty)
perc_empty <- data.frame(100*(val_count_empty/nrow(trainingData)))
#Remove that cols with many empty values from the analysis
trainingData<-trainingData[,perc_empty<90]
testingData<-testingData[,perc_empty<90]
dim(trainingData)
```

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

```
dim(testingData)
```

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

4.- Checking if there is features with a small variability of the values, which makes it useless for the prediction.There are not any feature with a single value, so the number of features for prediction is still 60.

```
val_count_unique <-sapply(trainingData, function(y) sum(length(unique(y))))
val_count_unique<- data.frame(val_count_unique)
trainingData<-trainingData[,val_count_unique>1]
testingData<-testingData[,val_count_unique>1]
dim(trainingData)
```

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

```
dim(testingData)
```

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

5.- Checking if there are features useless for the prediction due to its meaning (time, names, etc).After removing these features, the number have been reduced to 53.

```
remove_cols = c(1:7)
trainingData<-trainingData[,-remove_cols]
testingData<-testingData[,-remove_cols]
dim(trainingData)
```

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

```
dim(testingData)
```

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

6.- Remove near zero variables.

Near Zero variables: To identify these types of predictors, the following two metrics can be calculated: -the frequency of the most prevalent value over the second most frequent value (called the "frequency ratio"), which would be near one for well-behaved predictors and very large for highly-unbalanced data and -the "percent of unique values" is the number of unique values divided by the total number of samples (times 100) that approaches zero as the granularity of the data increases.

There are not any feature with near zero values, so the number of features for prediction is still 53.

```
nzv <- nearZeroVar(trainingData)
if (length(nzv) > 0) {
  trainingData <- trainingData[, -nzv]
  testingData <- testingData[, -nzv]
}
dim(trainingData)
```

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

```
dim(testingData)
```

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

7.- Checking the correlation between the features to remove features with extremely high correlation, which makes the information redundant for the prediction. After analyzing the correlations, and removing the features with a correlation above 0.75, the number of features have been reduced to 32.

```
#Keep only numerical columns
numericCols <-sapply(trainingData,is.numeric)
checkCorrTrainingData <- trainingData[,numericCols]
noCheckCorrTrainingData <- trainingData[,!numericCols]
#Calculate correlation
descrCor <-  cor(checkCorrTrainingData)
summary(descrCor[upper.tri(descrCor)])
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.992008 -0.110080  0.002092  0.001790  0.092552  0.980924
```

```
#Remove columns with correlation > cutoff
highlyCorDescr <- findCorrelation(descrCor, cutoff = .75)
checkCorrTrainingData <- checkCorrTrainingData[,-highlyCorDescr]
descrCor2 <- cor(checkCorrTrainingData)
summary(descrCor2[upper.tri(descrCor2)])
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.606983 -0.103773  0.006527  0.003332  0.087527  0.736546
```

```r
#Reconstruct dataframe with no numeric columns and numeric after correlation removal
trainingData <- data.frame(checkCorrTrainingData,"classe" = noCheckCorrTrainingData)
keepProblemId <- testingData$problem_id
testingData <- testingData[,intersect(colnames(testingData),colnames(trainingData))]
testingData <- data.frame(testingData,"problem_id" = keepProblemId)
dim(trainingData)
```

```
## [1] 19622    32
```

```r
dim(testingData)
```

```
## [1] 20 32
```

8.- Check if there is outliers plotting the data with the violin shape and remove the outliers. The features gyros_dumbell, gyros_forearm_x and gyros_forearm_z show outliers, and they have been removed. After analyzing the outliers, one observation of the training data have been removed, having 19621 to perform the analysis.

```r
library(dplyr)
trainingData<-trainingData%>%filter(gyros_dumbbell_y<10)
trainingData<-trainingData%>%filter(gyros_forearm_x>-10)
trainingData<-trainingData%>%filter(gyros_forearm_z<10)
dim(trainingData)
```

```
## [1] 19621    32
```

```r
dim(testingData)
```

```
## [1] 20 32
```

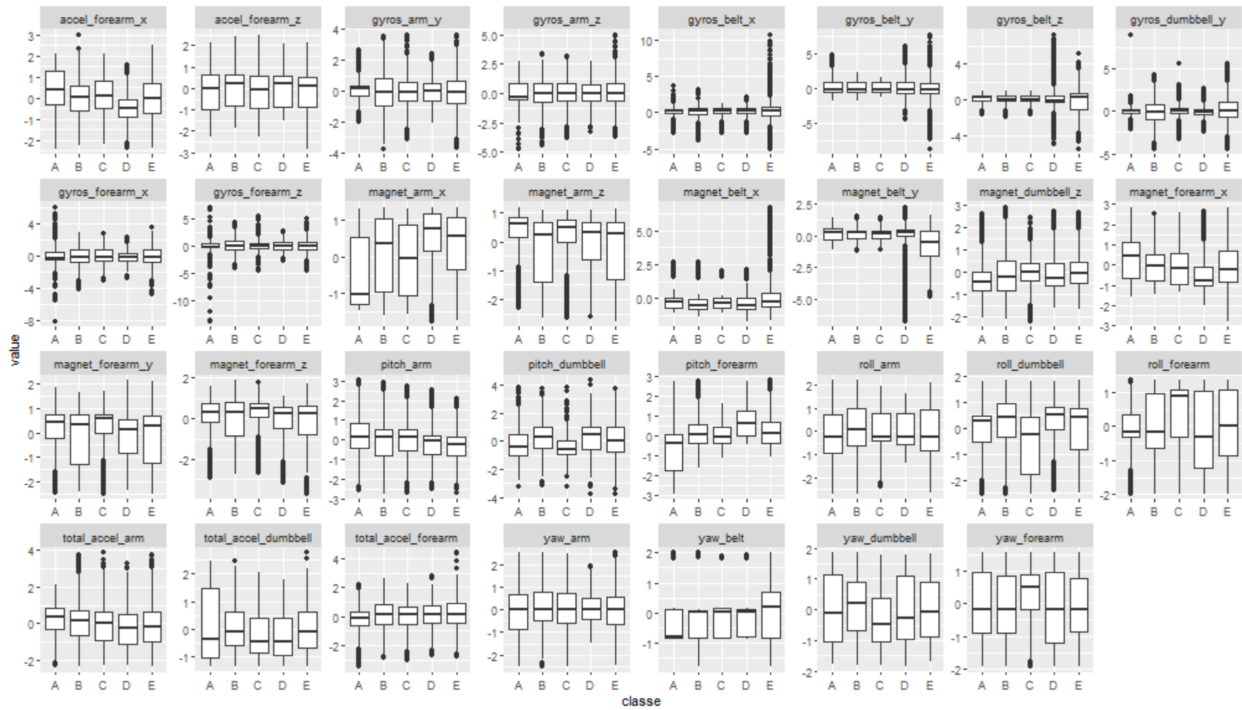
9.- Applying a preprocess to the data, to improve the performance or the prediction algorithms. From all the possible preprocess options (Possible values are “BoxCox”, “YeoJohnson”, “expoTrans”, “center”, “scale”, “range”, “knnImpute”, “bagImpute”, “medianImpute”, “pca”, “ica”, “spatialSign”, “corr”, “zv”, “nzv”, and “conditionalX”) centering and scaling have been selected. The correlation and nzv analysis have been carried out independently in previous steps.

```r
#Centering and Scaling
preProcValues <- preProcess(trainingData, method = c("center", "scale"))
trainTransformed <- predict(preProcValues, trainingData)
testTransformed <- predict(preProcValues, testingData)
```

10.- Finally, some plots are performed to analyze the reamining features before proceeding to the use of multiclass classification models for prediction.

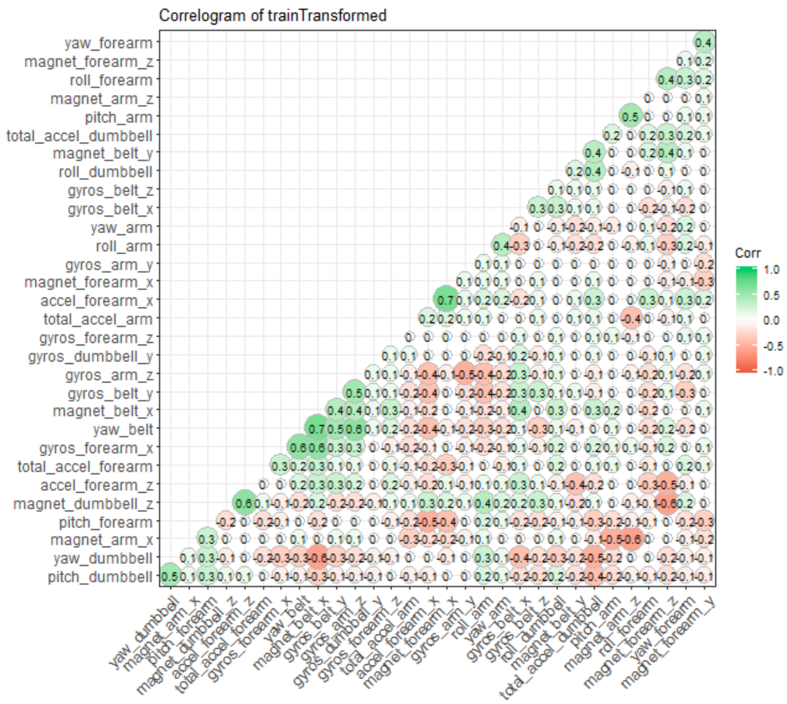
A boxplot analysis have been carried to, to visualize if there are some features which can be detected at first sight as good feature to distinguised between “classe” groups.

```r
library(ggplot2)
library(tidyr)
trainTransformedlong <- gather(trainTransformed, key="measure", value="value"
                               ,c(names(trainTransformed[,1:31])))
p <- ggplot(data = trainTransformedlong, aes(x=classe,y=value)) +
    geom_boxplot()
p + facet_wrap( ~measure, scales="free",ncol=8)
```



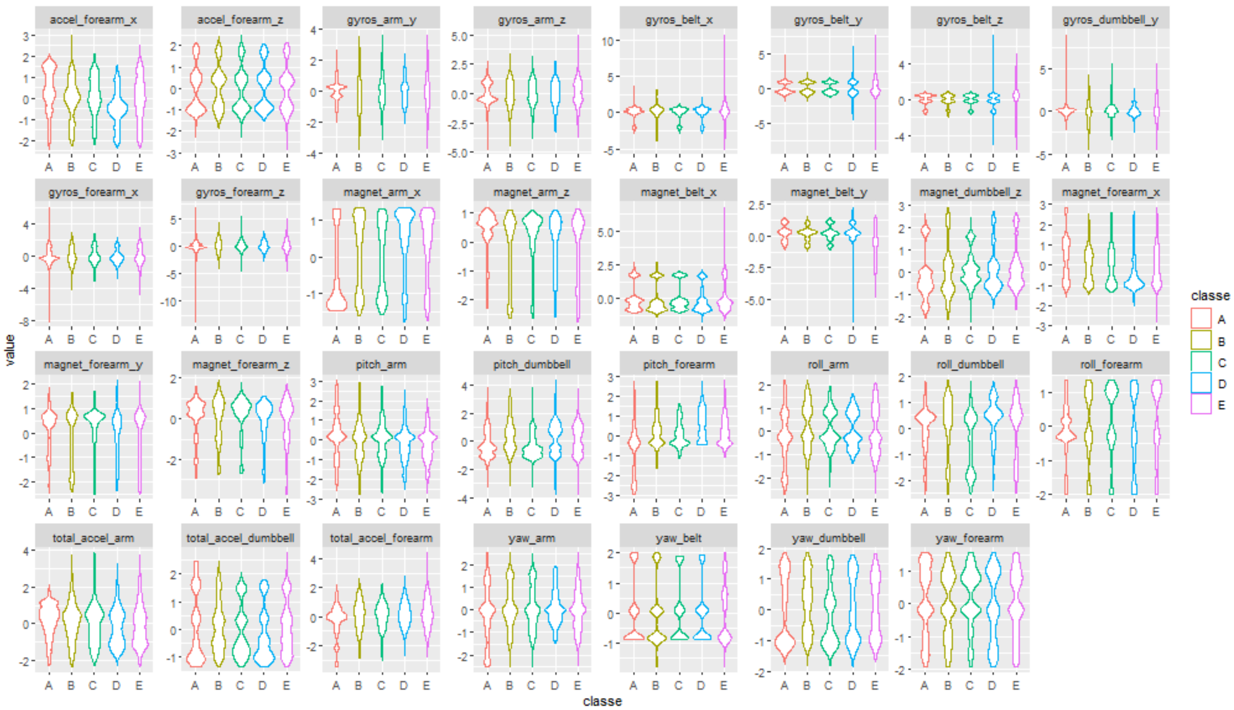
After that, a correlogram to show the correlation between the remaining features is presented as well.

```
#CorreloGram
library(ggplot2)
library(ggcorrplot)
corr <- round(cor(trainTransformed[,1:31]), 1)
ggcorrplot(corr, hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3,
           method="circle",
           colors = c("tomato2", "white", "springgreen3"),
           title="Correlogram of trainTransformed",
           ggtheme=theme_bw)
```



Finally, a violin plot to see the distribution of values per "classe" for each feature is shown as well.

```r
library(ggplot2)
library(tidyr)
library(dplyr)
trainTransformedlong3 <- gather(trainTransformed, key="measure", value="value",
                                c(names(trainTransformed[,1:31])))
val=colnames(trainTransformedlong3)[1]
# plot
g <- ggplot(trainTransformedlong3, aes(x=classe, y=value,color=classe))
g <- g + geom_violin()
g + facet_wrap( ~measure,scales="free",ncol=8)
```



Building the model. Analysis of Multiclass classification algorithms.

The following 15 multiclass classification methos have been tested, and its accuracy compared: knn, rf, pda, parRF, sda, hdrda, LMT, slda, hdda, RRFglobal, C5.0, LogitBoost, pam, PART, rpart.

For all of them, a cv method has been selected for cross validation.

Other methods could have been selected. The available resampling methods are: The "boot", "boot632", "optimism_boot", "boot_all", "cv", "repeatedcv", "LOOCV", "LGOCV" (for repeated training/test splits), "none" (only fits one model to the entire training set), "oob" (only for random forest, bagged trees, bagged earth, bagged flexible discriminant analysis, or conditional tree forest models), timeslice, "adaptive_cv", "adaptive_boot" or "adaptive_LGOCV"

```
library(doParallel)
registerDoParallel(4)
getDoParWorkers()
```

- KNN

```
fitKNN = train(
    classe ~.,
    data = trainTransformed,
    method = "knn",
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE),
    tuneGrid = expand.grid(k = seq(1, 21, by = 2))
)
head(predict(fitKNN, type = "prob"))
predictedKNN<-predict(fitKNN,newdata=trainTransformed)
CM_fitKNN <- confusionMatrix(predictedKNN, trainTransformed$classe)
```

```r
predictedKNN2<-predict(fitKNN,newdata=testTransformed)
varImp_fitKNN <- varImp(fitKNN)
```

- Random Forest

```r
fitRF = train(
    classe ~.,
    data = trainTransformed,
    method = 'rf',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitRF, type = "prob"))
predictedRF<-predict(fitRF,newdata=trainTransformed)
CM_fitRF <- confusionMatrix(predictedRF, trainTransformed$classe)
predictedRF2<-predict(fitRF,newdata=testTransformed)
varImp_fitRF <- varImp(fitRF)
```

- pda

```r
library(mda)
fitpda= train(
    classe ~.,
    data = trainTransformed,
    method = 'pda',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitpda, type = "prob"))
predictedpda<-predict(fitpda,newdata=trainTransformed)
CM_fitpda <- confusionMatrix(predictedpda, trainTransformed$classe)
predictedpda2<-predict(fitpda,newdata=testTransformed)
varImp_fitpda <- varImp(fitpda)
```

-Parallel Random Forest

```r
library(e1071)
library(randomForest)
fitPRF = train(
    classe ~.,
    data = trainTransformed,
    method = 'parRF',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitPRF, type = "prob"))
predictedPRF<-predict(fitPRF,newdata=trainTransformed)
CM_fitPRF <- confusionMatrix(predictedPRF, trainTransformed$classe)
predictedPRF2<-predict(fitPRF,newdata=testTransformed)
varImp_fitPRF <- varImp(fitPRF)
```

-sda

```r
library(sda)
fitsda= train(
    classe ~.,
```

```r
    data = trainTransformed,
    method = 'sda',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitsda, type = "prob"))
predictedsda<-predict(fitsda,newdata=trainTransformed)
CM_fitsda <- confusionMatrix(predictedsda, trainTransformed$classe)
predictedsda2<-predict(fitsda,newdata=testTransformed)
varImp_fitsda <- varImp(fitsda)
```

- hdrda

```r
library(sparsediscrim)
fithdrda= train(
    classe ~.,
    data = trainTransformed,
    method = 'hdrda',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fithdrda, type = "prob"))
predictedhdrda<-predict(fithdrda,newdata=trainTransformed)
CM_fithdrda <- confusionMatrix(predictedhdrda, trainTransformed$classe)
predictedhdrda2<-predict(fithdrda,newdata=testTransformed)
varImp_fithdrda <- varImp(fithdrda)
```

- LMT (Logistic Model Trees)

```r
##Logistic Model Trees
library(RWeka)
fitLMT = train(
    classe ~.,
    data = trainTransformed,
    method = "LMT",
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitLMT, type = "prob"))
predictedLMT<-predict(fitLMT,newdata=trainTransformed)
CM_fitLMT <- confusionMatrix(predictedLMT, trainTransformed$classe)
predictedLMT2<-predict(fitLMT,newdata=testTransformed)
varImp_fitLMT <- varImp(fitLMT)
```

- slda

```r
library(ipred)
fitslda= train(
    classe ~.,
    data = trainTransformed,
    method = 'slda',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitslda, type = "prob"))
predictedslda<-predict(fitslda,newdata=trainTransformed)
```

```
CM_fitslda <- confusionMatrix(predictedslda, trainTransformed$classe)
predictedslda2<-predict(fitslda,newdata=testTransformed)
varImp_fitslda <- varImp(fitslda)
```

- hdda

```
library(HDclassif)
fithdda= train(
    classe ~.,
    data = trainTransformed,
    method = 'hdda',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fithdda, type = "prob"))
predictedhdda<-predict(fithdda,newdata=trainTransformed)
CM_fithdda <- confusionMatrix(predictedhdda, trainTransformed$classe)
predictedhdda2<-predict(fithdda,newdata=testTransformed)
varImp_fithdda <- varImp(fithdda)
```

- RRFglobal

```
library(RRF)
fitRRFglobal= train(
    classe ~.,
    data = trainTransformed,
    method = 'RRFglobal',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitRRFglobal, type = "prob"))
predictedRRFglobal<-predict(fitRRFglobal,newdata=trainTransformed)
CM_fitRRFglobal <- confusionMatrix(predictedRRFglobal, trainTransformed$classe)
predictedRRFglobal2<-predict(fitRRFglobal,newdata=testTransformed)
varImp_fitRRFglobal <- varImp(fitRRFglobal)
```

- C5.0

```
fitC50 = train(
    classe ~.,
    data = trainTransformed,
    method = 'C5.0',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitC50, type = "prob"))
predictedC50<-predict(fitC50,newdata=trainTransformed)
CM_fitC50 <- confusionMatrix(predictedC50, trainTransformed$classe)
predictedC502<-predict(fitC50,newdata=testTransformed)
varImp_fitC50 <- varImp(fitC50)
```

- LogitBoost

```
library(caTools)
fitLogitBoost= train(
    classe ~.,
```

```
    data = trainTransformed,
    method = 'LogitBoost',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitLogitBoost, type = "prob"))
predictedLogitBoost<-predict(fitLogitBoost,newdata=trainTransformed)
CM_fitLogitBoost <- confusionMatrix(predictedLogitBoost, trainTransformed$classe)
predictedLogitBoost2<-predict(fitLogitBoost,newdata=testTransformed)
varImp_fitLogitBoost <- varImp(fitLogitBoost)
```

- pam

```
library(pamr)
fitpam= train(
    classe ~.,
    data = trainTransformed,
    method = 'pam',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitpam, type = "prob"))
predictedpam<-predict(fitpam,newdata=trainTransformed)
CM_fitpam <- confusionMatrix(predictedpam, trainTransformed$classe)
predictedpam2<-predict(fitpam,newdata=testTransformed)
varImp_fitpam <- varImp(fitpam)
```

- PART (Rule Based Classifier)

```
fitRBC = train(
    classe ~.,
    data = trainTransformed,
    method = 'PART',
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitRBC, type = "prob"))
predictedRBC<-predict(fitRBC,newdata=trainTransformed)
CM_fitRBC <- confusionMatrix(predictedRBC, trainTransformed$classe)
predictedRBC2<-predict(fitRBC,newdata=testTransformed)
varImp_fitRBC <- varImp(fitRBC)
```

- rpart (Tree with RPART)

```
fitRPART = train(
    classe ~.,
    data = trainTransformed,
    method = "rpart",
    trControl = trainControl(method = "cv", number = 10,savePredictions = "final"
                             ,allowParallel = TRUE)
)
head(predict(fitRPART, type = "prob"))
predictedRPART<-predict(fitRPART,newdata=trainTransformed)
CM_fitRPART <- confusionMatrix(predictedRPART, trainTransformed$classe)
predictedRPART2<-predict(fitRPART,newdata=testTransformed)
varImp_fitRPART <- varImp(fitRPART)
```

Results of the multiclass classification models. Importance of features.

In the following table, the results for accuracy for the algorithms evaluated are presented:

```
result_Analysis = data.frame(methods = methods<-c("C50","hdda","hdrda","KNN","LMT"
                                                  ,"LogitBoost","pam","pda","PRF"
                                                  ,"RBC","RF","RPART","RRFglobal"
                                                  ,"sda","slda"),
accuracy = c(CM_fitC50$overall["Accuracy"],CM_fithdda$overall["Accuracy"]
             ,CM_fithdrda$overall["Accuracy"],CM_fitC50$overall["Accuracy"],
             CM_fitKNN$overall["Accuracy"],CM_fitLogitBoost$overall["Accuracy"]
             ,CM_fitpam$overall["Accuracy"],CM_fitpda$overall["Accuracy"],
             CM_fitPRF$overall["Accuracy"],CM_fitRBC$overall["Accuracy"]
             ,CM_fitRF$overall["Accuracy"],CM_fitRPART$overall["Accuracy"],
             CM_fitRRFglobal$overall["Accuracy"],CM_fitsda$overall["Accuracy"]
             ,CM_fitslda$overall["Accuracy"]))
result_Analysis
```

```
##       methods  accuracy
## 1         C50 1.0000000
## 2        hdda 0.7601549
## 3       hdrda 0.8249834
## 4         KNN 1.0000000
## 5         LMT 1.0000000
## 6  LogitBoost 0.8700265
## 7         pam 0.4182254
## 8         pda 0.5834565
## 9         PRF 1.0000000
## 10        RBC 0.9992355
## 11         RF 1.0000000
## 12      RPART 0.5246929
## 13  RRFglobal 1.0000000
## 14        sda 0.5832017
## 15       slda 0.4235768
```

The results show that 6 methods have an in sample accuracy of 1, meaning that they are able to classify properly all the observations in the training data. With this is sample accuracy, it is expected to have a very high classification out of sample accuracy.To finalize the analysis,the importance of each predictor have been analyzed for those 6 methods with a 100% accuracy.

- Importance for C50

```
## C5.0 variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                     Overall
## gyros_belt_z         100.00
## magnet_forearm_x     100.00
## gyros_dumbbell_y     100.00
## magnet_forearm_y     100.00
## pitch_forearm        100.00
```

```
## magnet_belt_y      100.00
## yaw_belt            99.98
## yaw_arm             99.88
## magnet_arm_z        99.76
## roll_arm            98.78
## magnet_dumbbell_z   98.64
## gyros_belt_y        98.56
## roll_forearm        97.97
## gyros_arm_y         97.69
## magnet_belt_x       96.47
## magnet_forearm_z    94.15
## roll_dumbbell       93.48
## pitch_dumbbell      88.82
## gyros_belt_x        88.59
## yaw_dumbbell        86.33
```

- Importance for KNN

```
## ROC curve variable importance
##
##   variables are sorted by maximum importance across the classes
##   only 20 most important variables shown (out of 31)
##
##                          A       B      C     D       E
## pitch_forearm        62.964 100.000 67.237 62.96 100.000
## accel_forearm_x      40.675  81.171 40.675 40.68  81.171
## magnet_arm_x         53.104  78.110 65.004 53.10  78.110
## pitch_dumbbell       52.842  52.842 52.842 71.31  44.049
## magnet_forearm_x     38.891  71.126 33.586 33.59  71.126
## magnet_belt_y        15.289  10.317 67.372 10.32  15.289
## roll_dumbbell        41.216  51.434 30.666 62.50  51.434
## magnet_dumbbell_z    56.342  36.753 53.959 23.75  56.342
## magnet_arm_z         52.443  52.443 52.443 52.44  40.079
## pitch_arm            26.743  42.163 49.028 26.74  42.163
## yaw_dumbbell         19.186  19.186 19.186 41.29  13.608
## total_accel_arm      30.686  40.517 32.030 15.97  40.517
## magnet_forearm_y     19.326  37.449 28.196 24.54  37.449
## roll_arm             34.545  34.545 34.545 34.54  24.087
## roll_forearm         34.196   6.325 13.790 23.73  34.196
## total_accel_forearm  23.437  27.044 31.981 23.44  27.044
## magnet_belt_x        25.177  25.177 25.177 25.26  18.092
## yaw_forearm          10.754  16.280 11.636 22.29  16.280
## total_accel_dumbbell  9.533  17.622  9.533 19.89  17.622
## yaw_belt              9.833   9.833 18.426 17.17   8.828
```

- Importance for LMT

```
## ROC curve variable importance
##
##   variables are sorted by maximum importance across the classes
##   only 20 most important variables shown (out of 31)
##
##                          A       B      C     D       E
## pitch_forearm        62.964 100.000 67.237 62.96 100.000
## accel_forearm_x      40.675  81.171 40.675 40.68  81.171
## magnet_arm_x         53.104  78.110 65.004 53.10  78.110
```

```
## pitch_dumbbell       52.842  52.842 52.842 71.31  44.049
## magnet_forearm_x     38.891  71.126 33.586 33.59  71.126
## magnet_belt_y        15.289  10.317 67.372 10.32  15.289
## roll_dumbbell        41.216  51.434 30.666 62.50  51.434
## magnet_dumbbell_z    56.342  36.753 53.959 23.75  56.342
## magnet_arm_z         52.443  52.443 52.443 52.44  40.079
## pitch_arm            26.743  42.163 49.028 26.74  42.163
## yaw_dumbbell         19.186  19.186 19.186 41.29  13.608
## total_accel_arm      30.686  40.517 32.030 15.97  40.517
## magnet_forearm_y     19.326  37.449 28.196 24.54  37.449
## roll_arm             34.545  34.545 34.545 34.54  24.087
## roll_forearm         34.196   6.325 13.790 23.73  34.196
## total_accel_forearm  23.437  27.044 31.981 23.44  27.044
## magnet_belt_x        25.177  25.177 25.177 25.26  18.092
## yaw_forearm          10.754  16.280 11.636 22.29  16.280
## total_accel_dumbbell  9.533  17.622  9.533 19.89  17.622
## yaw_belt              9.833   9.833 18.426 17.17   8.828
```

- Importance for PRF

```
## parRF variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                      Overall
## yaw_belt              100.00
## magnet_dumbbell_z      80.55
## magnet_belt_y          67.07
## pitch_forearm          66.86
## roll_dumbbell          56.55
## roll_forearm           51.98
## gyros_belt_z           44.00
## roll_arm               38.62
## gyros_dumbbell_y       37.79
## total_accel_dumbbell   37.35
## yaw_dumbbell           37.15
## accel_forearm_x        32.61
## accel_forearm_z        31.97
## pitch_dumbbell         31.19
## magnet_forearm_z       30.62
## magnet_belt_x          30.46
## yaw_arm                29.72
## magnet_forearm_y       28.68
## magnet_forearm_x       27.56
## magnet_arm_x           27.42
```

- Importance for RF

```
## rf variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                      Overall
## yaw_belt              100.00
## magnet_dumbbell_z      74.64
## magnet_belt_y          67.66
```

```
## pitch_forearm          65.28
## roll_forearm           58.02
## roll_dumbbell          53.96
## gyros_belt_z           42.88
## roll_arm               40.64
## total_accel_dumbbell   39.35
## yaw_dumbbell           37.96
## gyros_dumbbell_y       37.50
## accel_forearm_x        34.37
## magnet_arm_x           32.23
## pitch_dumbbell         31.52
## magnet_forearm_z       30.72
## accel_forearm_z        29.36
## magnet_belt_x          29.25
## magnet_forearm_y       28.01
## magnet_forearm_x       27.51
## yaw_arm                27.46
```

- Importance for RRF Global

```
## RRFglobal variable importance
##
##   only 20 most important variables shown (out of 31)
##
##                       Overall
## yaw_belt              100.000
## pitch_forearm          78.181
## magnet_belt_y          74.533
## magnet_dumbbell_z      55.788
## roll_dumbbell          47.250
## total_accel_dumbbell   40.856
## roll_forearm           38.187
## accel_forearm_z        30.353
## gyros_belt_z           26.907
## magnet_forearm_z       18.316
## accel_forearm_x        18.316
## yaw_dumbbell           17.891
## magnet_belt_x          16.838
## roll_arm               15.080
## yaw_arm                14.240
## pitch_arm              12.333
## magnet_arm_x           11.732
## gyros_arm_y             9.711
## pitch_dumbbell          9.587
## gyros_dumbbell_y        9.366
```