Predicting the Manner the Excercise Is Being Performed

Sue, August 4, 2016

Introduction

The main goal of this project is to predict the manner in which 6 participants performed some exercise. This is the “classe” variable in the training set. By doing some cross validation the best model is chosen and uses to predict the 20 cases in the test dataset.

Data and Some Exploratory Analysis

In this project, the goal is to 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. More information is available here (http://groupware.les.inf.puc-rio.br/har). You can find the training and test data for this project here:

- Trainset (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)
- Testset (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

Loading and Cleaning Data

There are 160 variables in the original dataset many of which contain large number of NAs. Some other variables are highly correlated so we have to clean the data before further analysis.

Omitting Variables with High Amount of Missing Values

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(caret)
```

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

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

```r
library(tree)
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
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(rpart)
library(rpart.plot)
library(corrplot)

data= read.csv("./Prac_Mach_Ass.csv", header =T, stringsAsFactors = F )
data_test= read.csv("./Prac_Mach_Ass_test.csv", header =T, stringsAsFactors =
F )

#choosing usefull variables:

data2=select(data,user_name, raw_timestamp_part_1, num_window, roll_belt, pitch
_belt, yaw_belt, total_accel_belt, gyros_belt_x, gyros_belt_y, gyros_belt_z, ac
cel_belt_x, accel_belt_y, accel_belt_z, magnet_belt_x, magnet_belt_y, magnet_be
lt_z, roll_arm, pitch_arm, yaw_arm, total_accel_arm, gyros_arm_y, gyros_arm_x,
gyros_arm_z, accel_arm_x, accel_arm_y, accel_arm_z, magnet_arm_x, magnet_arm_
y, magnet_arm_z, roll_dumbbell, pitch_dumbbell, yaw_dumbbell, total_accel_dumbb
ell, gyros_dumbbell_y, gyros_dumbbell_x, gyros_dumbbell_z, accel_dumbbell_x, ac
cel_dumbbell_y, accel_dumbbell_z, magnet_dumbbell_x, magnet_dumbbell_y, magnet_
dumbbell_z,roll_forearm, pitch_forearm, yaw_forearm, total_accel_forearm, gyros
_forearm_y, gyros_forearm_x, gyros_forearm_z, accel_forearm_x, accel_forearm_
y, accel_forearm_z, magnet_forearm_x, magnet_forearm_y, magnet_forearm_z,class
e)

#Just numeric variables for further analysis

data2_c=select(data2, roll_belt, pitch_belt, yaw_belt, total_accel_belt, gyros_
belt_x, gyros_belt_y, gyros_belt_z, accel_belt_x, accel_belt_y, accel_belt_z, m
agnet_belt_x, magnet_belt_y, magnet_belt_z, roll_arm, pitch_arm, yaw_arm, total
_accel_arm, gyros_arm_y, gyros_arm_x, gyros_arm_z, accel_arm_x, accel_arm_y, ac
cel_arm_z, magnet_arm_x, magnet_arm_y, magnet_arm_z, roll_dumbbell, pitch_dumbb
ell, yaw_dumbbell, total_accel_dumbbell, gyros_dumbbell_y, gyros_dumbbell_x, gy
ros_dumbbell_z, accel_dumbbell_x, accel_dumbbell_y, accel_dumbbell_z, magnet_du
mbbell_x, magnet_dumbbell_y, magnet_dumbbell_z,roll_forearm, pitch_forearm, yaw
_forearm, total_accel_forearm, gyros_forearm_y, gyros_forearm_x, gyros_forearm_
z, accel_forearm_x, accel_forearm_y, accel_forearm_z, magnet_forearm_x, magnet_
forearm_y, magnet_forearm_z)
```

Finding Highly Correlated Variables

As we know high correlations between variables can ruin the model and predictions. Here I search for high correlations (more than 90%) and omit variables accordingly. The figure can show the correlations between variables much better. At the end just 46 out of 160 variable remain.

```r
#Finding highly correlated variables
cor_columns = findCorrelation(cor(data2_c), cutoff = .89, verbose = TRUE)
```

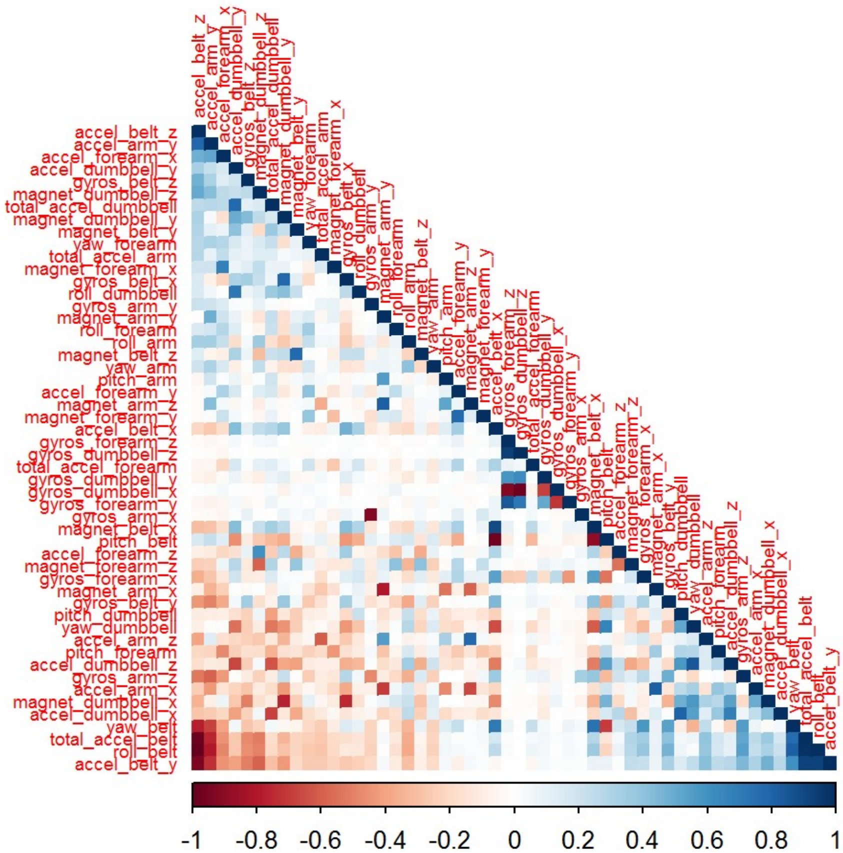
```
## Compare row 10  and column  1 with corr  0.992
##   Means:  0.27 vs 0.168 so flagging column 10
## Compare row 1  and column  9 with corr  0.925
##   Means:  0.25 vs 0.164 so flagging column 1
## Compare row 9  and column  4 with corr  0.928
##   Means:  0.233 vs 0.161 so flagging column 9
## Compare row 8  and column  2 with corr  0.966
##   Means:  0.245 vs 0.157 so flagging column 8
## Compare row 18  and column  19 with corr  0.918
##   Means:  0.091 vs 0.158 so flagging column 19
## Compare row 46  and column  32 with corr  0.914
##   Means:  0.101 vs 0.161 so flagging column 32
## Compare row 46  and column  33 with corr  0.933
##   Means:  0.083 vs 0.164 so flagging column 33
## All correlations <= 0.89
```

```
#Removing correlated columns
data2_no_c = select(data2_c, -cor_columns)

#Checking near zero variable
zero_var=nearZeroVar(data2_no_c)
#No nearzero variance variable
#Check if there is any NA
sum(is.na(data2_no_c))
```

```
## [1] 0
```

```
#No NA
corrplot(cor(data2_c), order = "FPC", method = "color", type = "lower", tl.cex
= 0.6)
```



Partitioning the Training Set for Some Cross Validation

```
#adding classe variable to the rest of them
data_final=mutate(data2_no_c, classe=data2$classe)

#Split the data to training and testing data
inTrain=createDataPartition(data_final$classe, p=0.7, list = F)
Train=data_final[inTrain, ]
Test=data_final[-inTrain, ]
```

Analysis

Here I use different methods for analysis. I mostly will use caret package.

Decition Tree from Tree Package

```
#Analysis
##############################################rpart function###############
#################
set.seed(12345)
treeT<- rpart(classe ~ .,data=Train, method="class")
print(treeT)
```

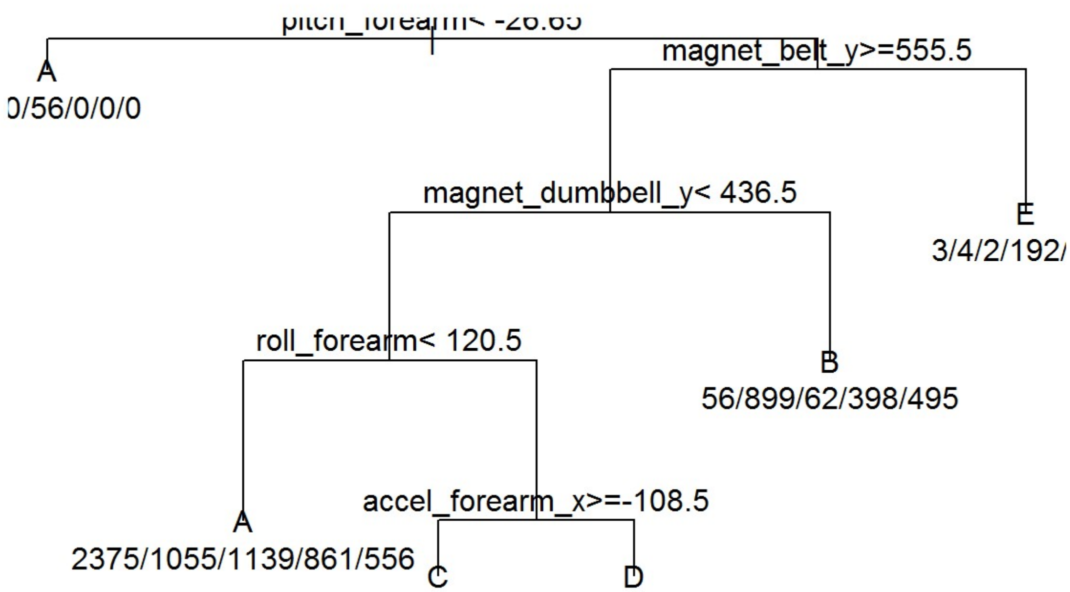
```
#Cross Validation
tree.P=predict(treeT,  newdata=Test, type="class")
confMatTR <- confusionMatrix(tree.P,Test$classe)
confMatTR
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1526  267   38   96   87
##          B   53  593  118   42  168
##          C   29  150  765  104  164
##          D   54  103   98  710   63
##          E   12   26    7   12  600
##
## Overall Statistics
##
##                Accuracy : 0.7127
##                  95% CI : (0.7009, 0.7242)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6342
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9116   0.5206   0.7456   0.7365   0.5545
## Specificity            0.8841   0.9197   0.9080   0.9354   0.9881
## Pos Pred Value         0.7577   0.6088   0.6312   0.6907   0.9132
## Neg Pred Value         0.9618   0.8888   0.9441   0.9477   0.9078
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2593   0.1008   0.1300   0.1206   0.1020
## Detection Prevalence   0.3422   0.1655   0.2059   0.1747   0.1116
## Balanced Accuracy      0.8979   0.7202   0.8268   0.8359   0.7713
```

Not a promising prediction! Just 70% of accuracy!

Tree Method from Caret Package

```
##################################################rpart as method(more_clear)###
###################
set.seed(12345)
tree.Train2<- train(classe ~ .,method = "rpart", data=Train)
print(tree.Train2$finalModel)
plot(tree.Train2$finalModel)
text(tree.Train2$finalModel, use.n = T)
```




```
#Cross Validation
tree.predict2=predict(tree.Train2, Test)
confMatTR2 <- confusionMatrix(tree.predict2, Test$classe)
confMatTR2
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1522  475  441  363  239
##          B   35  383   47  182  189
##          C   87  244  433  104  231
##          D   30   37  105  241   48
##          E    0    0    0   74  375
## 
## Overall Statistics
## 
##                Accuracy : 0.502
##                  95% CI : (0.4891, 0.5148)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.3499
##  Mcnemar's Test P-Value : < 2.2e-16
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9092  0.33626  0.42203  0.25000  0.34658
## Specificity            0.6395  0.90455  0.86293  0.95529  0.98459
## Pos Pred Value         0.5007  0.45813  0.39399  0.52278  0.83519
## Neg Pred Value         0.9466  0.85027  0.87610  0.86670  0.86994
## Prevalence             0.2845  0.19354  0.17434  0.16381  0.18386
## Detection Rate         0.2586  0.06508  0.07358  0.04095  0.06372
## Detection Prevalence   0.5166  0.14206  0.18675  0.07833  0.07630
## Balanced Accuracy      0.7744  0.62041  0.64248  0.60265  0.66559
```

Prediction gets worse using caret package (only 49% of accuracy)!

Random Forest from Caret Package

```
#################################################Random Forest################
###################
set.seed(12345)
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
rf.Train=train(classe~., method = "rf", data=Train, trControl=controlRF)
rf.Train$finalModel
```

```r
#Cross Validation
predictRF <- predict(rf.Train, newdata=Test)
confMatRF <- confusionMatrix(predictRF, Test$classe)
confMatRF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1670    3    0    0    0
##          B    3 1131   10    0    0
##          C    1    5 1016    5    2
##          D    0    0    0  959    1
##          E    0    0    0    0 1079
##
## Overall Statistics
##
##                Accuracy : 0.9949
##                  95% CI : (0.9927, 0.9966)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9976   0.9930   0.9903   0.9948   0.9972
## Specificity            0.9993   0.9973   0.9973   0.9998   1.0000
## Pos Pred Value         0.9982   0.9886   0.9874   0.9990   1.0000
## Neg Pred Value         0.9991   0.9983   0.9979   0.9990   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2838   0.1922   0.1726   0.1630   0.1833
## Detection Prevalence   0.2843   0.1944   0.1749   0.1631   0.1833
## Balanced Accuracy      0.9984   0.9951   0.9938   0.9973   0.9986
```

The prediction is highly accurate (more than 99%) . This model is a good candidate to be used in predicting the test data set.

Generalized Boosted Model from Caret

The last model I try is Generalized Boosted Model from caret package.

```
###############################################Generalized Boosted Model#####
##################
set.seed(12345)
controlgbm <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
gbm.Train  <- train(classe ~ ., data=Train, method = "gbm", trControl = control
gbm, verbose = FALSE)
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

```
## Loading required package: plyr
```

```
## -------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, the
n dplyr:
## library(plyr); library(dplyr)
```

```
## -------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
gbm.Train$finalModel
```

```
#Cross Validation
predictgbm <- predict(gbm.Train, newdata=Test)
confMatGBM <- confusionMatrix(predictgbm, Test$classe)
confMatGBM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1653   36    0    1    5
##          B    9 1059   47    3    9
##          C    8   42  969   25    9
##          D    2    1   10  928   11
##          E    2    1    0    7 1048
##
## Overall Statistics
##
##                Accuracy : 0.9613
##                  95% CI : (0.956, 0.966)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.951
##  Mcnemar's Test P-Value : 2.884e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9875   0.9298   0.9444   0.9627   0.9686
## Specificity            0.9900   0.9857   0.9827   0.9951   0.9979
## Pos Pred Value         0.9752   0.9397   0.9202   0.9748   0.9905
## Neg Pred Value         0.9950   0.9832   0.9882   0.9927   0.9930
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2809   0.1799   0.1647   0.1577   0.1781
## Detection Prevalence   0.2880   0.1915   0.1789   0.1618   0.1798
## Balanced Accuracy      0.9887   0.9577   0.9636   0.9789   0.9832
```

The prediction of this model is also highly accurate (more than 96%), however is not as high as Random Forest method. Therefore I will use Random Forest for the last part of the project which is predicting the manner of exercise in test dataset.

```
predictRF_test <- predict(rf.Train, newdata=data_test)
predictRF_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
```