Predicting Class (State) of a Person

Anil Kumar J

27 June 2018

Overview

As part of this project we trained models using various Machine Learning Algorithms to use quantified self movement variables as predictors to get the state of a person. Compared these models and Random Forest seemed to give the best Accuracy. Used Random Forest Model to predict on test data.

Simulations

Loading the datasets

```
training_data <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
,na.strings=c("","NA"))
testing_data<-read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
```

Basic Exploratory Data Analyses

```
dim(training_data)
```

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

There seem to be various columns that can be used as predictors but from the notes mentioned it is better to use columns that have on of these terms - belt,arm,forearm,dumbbell in the name and also avoiding various variable that have lot of NA's (non measured values) should help us decrease the variables (features) count.

```
na_stats<-lapply(training_data, function(x) sum(is.na(x)))
na_stats<-do.call(rbind.data.frame, na_stats)
colnames(na_stats)<-c("no_of_nas")
na_stats <- which(na_stats$no_of_nas>15000)
col_rm<-c(na_stats)
col_av<-c(grep(("belt|arm|forearm|dumbbell"),names(training_data)))
col_consi<-setdiff(col_av, col_rm)
```

Building Models

Using the columns which we had considered to be important, we predict the classe variable using various Algorithms like Random Forest, Gradient Dissent, Linear Discrimante Analysis, Navie Bayes, Trees.

```
library(caret)
library(parallel)
library(doParallel)
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
fitControl <- trainControl(method = "cv",number = 5,allowParallel = TRUE)
model_fit_rf <- train(training_data[,c(col_consi)],as.factor(training_data[,c("classe")]), metho
d="rf",trControl = fitControl)
model_fit_gbm <- train(training_data[,c(col_consi)],as.factor(training_data[,c("classe")]), meth
od="gbm",trControl = fitControl)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2331
##      2        1.4604             nan     0.1000    0.1606
##      3        1.3586             nan     0.1000    0.1277
##      4        1.2789             nan     0.1000    0.1101
##      5        1.2098             nan     0.1000    0.0955
##      6        1.1500             nan     0.1000    0.0685
##      7        1.1065             nan     0.1000    0.0638
##      8        1.0665             nan     0.1000    0.0568
##      9        1.0293             nan     0.1000    0.0630
##     10        0.9913             nan     0.1000    0.0446
##     20        0.7626             nan     0.1000    0.0268
##     40        0.5318             nan     0.1000    0.0108
##     60        0.4112             nan     0.1000    0.0093
##     80        0.3291             nan     0.1000    0.0046
##    100        0.2736             nan     0.1000    0.0052
##    120        0.2301             nan     0.1000    0.0022
##    140        0.1968             nan     0.1000    0.0016
##    150        0.1828             nan     0.1000    0.0016
```

```
model_fit_lda <- train(training_data[,c(col_consi)],as.factor(training_data[,c("classe")]), meth
od="lda",trControl = fitControl)
model_fit_nb <- train(training_data[,c(col_consi)],as.factor(training_data[,c("classe")]), metho
d="nb",trControl = fitControl)
model_fit_rpart<- train(training_data[,c(col_consi)],as.factor(training_data[,c("classe")]), met
hod="rpart",trControl = fitControl)
stopCluster(cluster)
registerDoSEQ()
```

From stats in Appendix it can be seen that Random Forest has the best Accuracy.

Predicting on test data

Using Random FOrest to predict Classe for test data.

```
predict(model_fit_rf,testing_data)
```

```
##  [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
```

Appendix

Random Forest Stats

```
confusionMatrix.train(model_fit_rf)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 28.4  0.1  0.0  0.0  0.0
##          B  0.0 19.2  0.1  0.0  0.0
##          C  0.0  0.0 17.3  0.3  0.0
##          D  0.0  0.0  0.0 16.1  0.0
##          E  0.0  0.0  0.0  0.0 18.3
##
##  Accuracy (average) : 0.9943
```

Gradient Dissent Stats

```
confusionMatrix.train(model_fit_gbm)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 28.0  0.6  0.0  0.0  0.0
##          B  0.3 18.3  0.5  0.1  0.2
##          C  0.1  0.4 16.6  0.6  0.2
##          D  0.1  0.0  0.2 15.6  0.2
##          E  0.0  0.0  0.0  0.1 17.7
##
##  Accuracy (average) : 0.9623
```

Linear Discrimnate Analysis

```
confusionMatrix.train(model_fit_lda)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 23.2  3.0  1.7  0.9  0.7
##          B  0.6 12.4  1.7  0.7  3.1
##          C  2.3  2.3 11.4  1.9  1.7
##          D  2.2  0.8  2.1 12.2  1.8
##          E  0.1  0.9  0.4  0.7 11.0
##
##  Accuracy (average) : 0.7027
```

Navie Bayes

```
confusionMatrix.train(model_fit_nb)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 24.5  3.5  3.4  2.8  0.9
##          B  0.7 12.9  1.2  0.0  1.7
##          C  1.2  1.8 12.1  2.2  0.7
##          D  1.9  1.0  0.7 10.6  0.6
##          E  0.2  0.2  0.1  0.8 14.4
##
##  Accuracy (average) : 0.7449
```

Predicting with Trees

```
confusionMatrix.train(model_fit_rpart)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 25.8  8.2  8.1  7.3  2.6
##          B  0.5  6.5  0.6  2.9  2.5
##          C  1.8  3.7  8.3  4.6  3.9
##          D  0.2  0.9  0.5  1.5  1.0
##          E  0.2  0.0  0.0  0.0  8.4
##
##  Accuracy (average) : 0.5044
```