Analysis of Accelerometers Data

Sarah

9/17/2020

Introduction

This project uses data recorded from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. These participants were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

The goal is to predict the manner in which the participants did the exercise » classe variable

Note: due to limited memory, the trainig and evaluation was done on sample of the data and also could only train few models

Getting Data

The training and testing sets are downloaded from the online source as below:

```
set.seed(123)
```

```
if(!file.exists("data")){
        dir.create("`data")
        download.file(url="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",destfil
        download.file(url="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",destfile
}
training<-read.csv("data/training.csv")
testing<-read.csv("data/testing.csv")
```

Then subsampled to get random sample:

```
nRowsTrain<-dim(training)[1]
nRowsTest<-dim(training)[1]
partTrain<-sample(nRowsTrain,1000)
partTest<-sample(nRowsTest,1000)


training<-training[partTrain,]
testing<-testing[partTest,]
```

Exploring the training set

In order to build the model we should first explore the training set. The dataset consists of 160 columns and 1000 rows.

```r
checkNAs<-function(v,len){
        sum(is.na(v)*1)/len
}
amountNAs<-sapply(training,FUN = checkNAs,nRows)
colsNAs<-amountNAs[which(amountNAs>0)]
```

1- By looking at the data, we first noticed 67 columns with high percentage of missing values so we decided to track them and exclude them from the data set:

```r
colsNAs
```

```
##            max_roll_belt          max_picth_belt           min_roll_belt
##                    0.984                   0.984                   0.984
##           min_pitch_belt     amplitude_roll_belt    amplitude_pitch_belt
##                    0.984                   0.984                   0.984
##     var_total_accel_belt           avg_roll_belt        stddev_roll_belt
##                    0.984                   0.984                   0.984
##            var_roll_belt          avg_pitch_belt       stddev_pitch_belt
##                    0.984                   0.984                   0.984
##           var_pitch_belt            avg_yaw_belt         stddev_yaw_belt
##                    0.984                   0.984                   0.984
##             var_yaw_belt           var_accel_arm            avg_roll_arm
##                    0.984                   0.984                   0.984
##          stddev_roll_arm            var_roll_arm           avg_pitch_arm
##                    0.984                   0.984                   0.984
##         stddev_pitch_arm           var_pitch_arm             avg_yaw_arm
##                    0.984                   0.984                   0.984
##           stddev_yaw_arm             var_yaw_arm            max_roll_arm
##                    0.984                   0.984                   0.984
##            max_picth_arm             max_yaw_arm            min_roll_arm
##                    0.984                   0.984                   0.984
##            min_pitch_arm             min_yaw_arm      amplitude_roll_arm
##                    0.984                   0.984                   0.984
##      amplitude_pitch_arm       amplitude_yaw_arm       max_roll_dumbbell
##                    0.984                   0.984                   0.984
##       max_picth_dumbbell       min_roll_dumbbell      min_pitch_dumbbell
##                    0.984                   0.984                   0.984
##  amplitude_roll_dumbbell amplitude_pitch_dumbbell      var_accel_dumbbell
##                    0.984                   0.984                   0.984
##        avg_roll_dumbbell    stddev_roll_dumbbell       var_roll_dumbbell
##                    0.984                   0.984                   0.984
##       avg_pitch_dumbbell   stddev_pitch_dumbbell      var_pitch_dumbbell
##                    0.984                   0.984                   0.984
##         avg_yaw_dumbbell     stddev_yaw_dumbbell        var_yaw_dumbbell
##                    0.984                   0.984                   0.984
##         max_roll_forearm       max_picth_forearm        min_roll_forearm
##                    0.984                   0.984                   0.984
##        min_pitch_forearm    amplitude_roll_forearm amplitude_pitch_forearm
##                    0.984                   0.984                   0.984
##        var_accel_forearm        avg_roll_forearm     stddev_roll_forearm
##                    0.984                   0.984                   0.984
##         var_roll_forearm       avg_pitch_forearm    stddev_pitch_forearm
##                    0.984                   0.984                   0.984
```

```
##        var_pitch_forearm          avg_yaw_forearm       stddev_yaw_forearm
##                    0.984                    0.984                    0.984
##          var_yaw_forearm
##                    0.984
```

```
trainingNew<-training[,!(names(training) %in% names(colsNAs))]
testingNew<-training[,!(names(testing) %in% names(colsNAs))]
```

Now we are left with 93 columns
2- we will exclude the first 7 columns as they are related to participants information and other information not usefull to be used as predictors.

```
excludedCols<-names(trainingNew)[1:7]
excludedCols
```

```
## [1] "X"                    "user_name"            "raw_timestamp_part_1"
## [4] "raw_timestamp_part_2" "cvtd_timestamp"       "new_window"
## [7] "num_window"
```

```
trainingNew<-trainingNew[,!(names(trainingNew) %in% excludedCols)]
testingNew<-trainingNew[,!(names(trainingNew) %in% excludedCols)]
```

Now we have 86 columns that will try to fit model with

Clear Some Memory

```
rm(training,testing,partTest,partTrain,colsNAs)
```

Building the models

Extracting validation set

We split the training set to training and validation. The testing set is left for final testing on unseen data (we set small portion for training due to memory limit problem)

```
library(caret)
```

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

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

```
inTrain <- createDataPartition(y = trainingNew$classe, p = 0.75, list = FALSE)
trainingNew <- trainingNew[inTrain, ]
validation <- trainingNew[-inTrain, ]

dim(trainingNew);dim(validation)
```

```
## [1] 752  86
```

```
## [1] 192  86
```

Define CV function

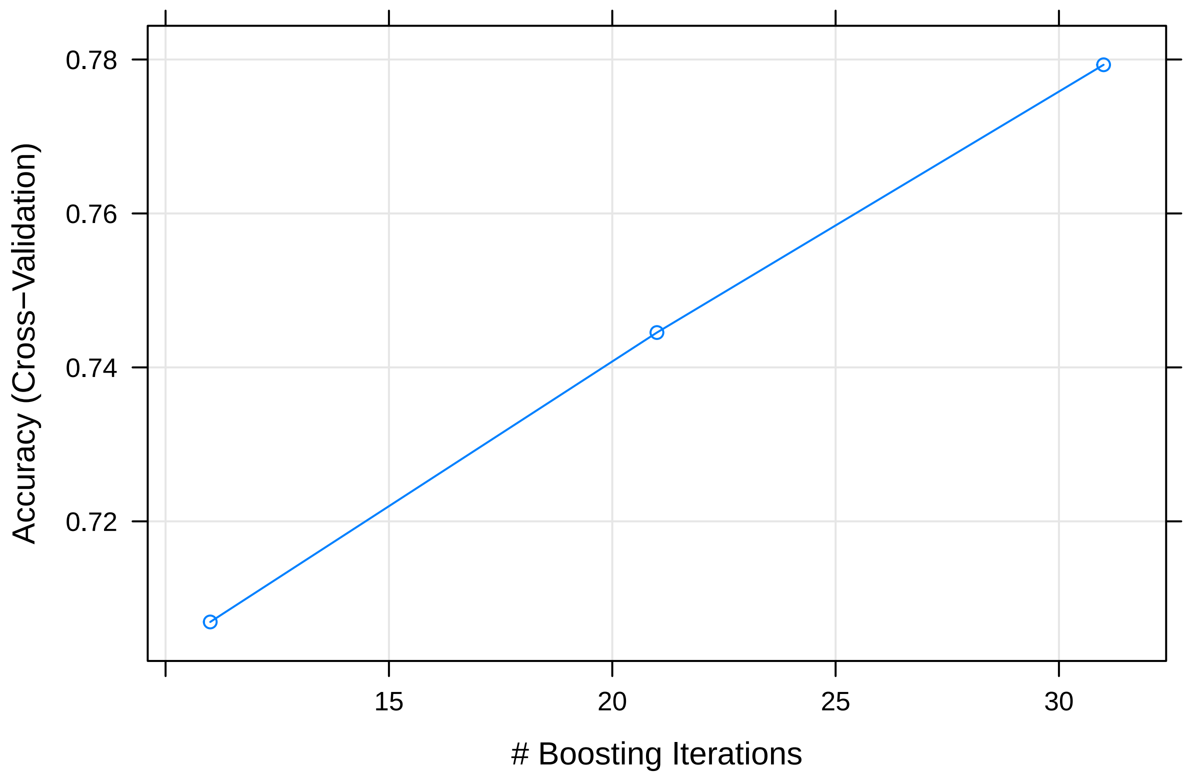
This cross validation function will be used in the train function for all models

```
fitControl <- trainControl(method = "cv", number = 3, returnResamp = "all")
```

Train model :logistic regression with boosting

```
mod.LogitBoost<- train(classe ~ ., data = trainingNew, method = "LogitBoost"
              , trControl = fitControl)
```

```
plot(mod.LogitBoost)
```

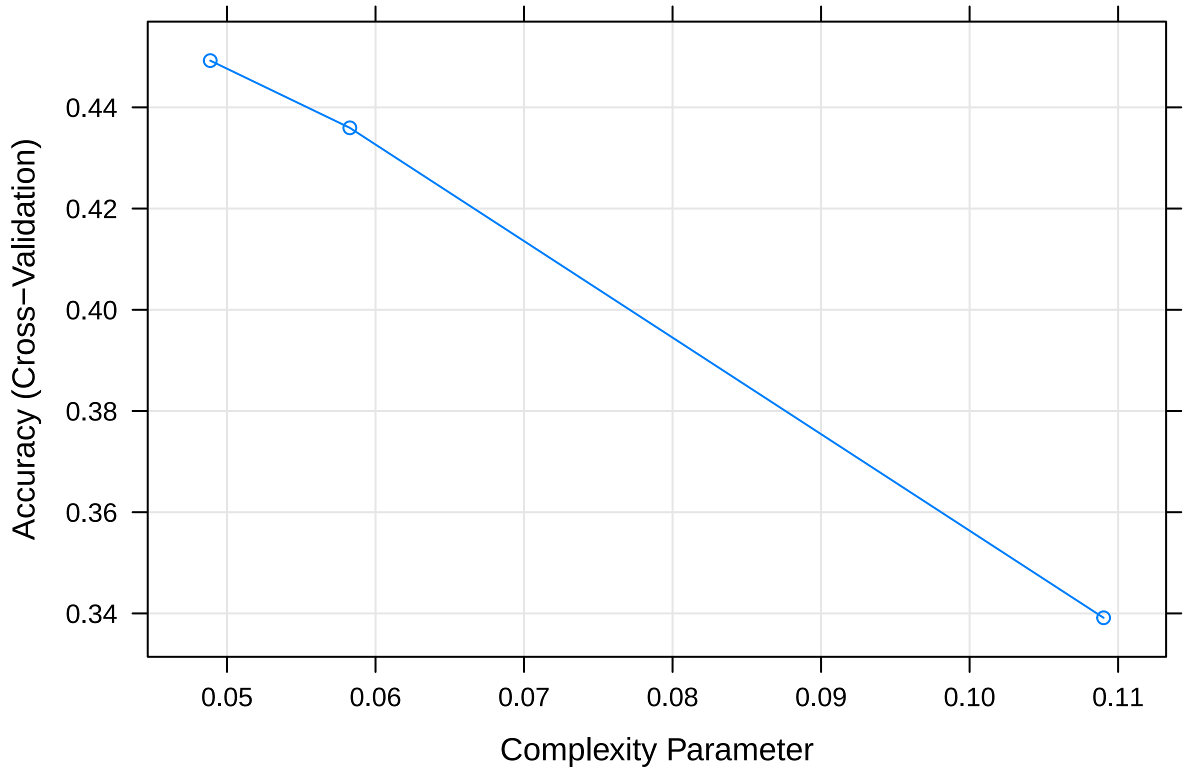


The LogitBoost shows relatively good accuracy over all cross validation sets

Train model :classification tree

```
mod.rpart<- train(classe ~ ., data = trainingNew, method = "rpart"
              , trControl = fitControl)
```

```r
plot(mod.rpart)
```



The rpart model shows worse results than LogitBoost over all the cross validation sets.

Evaluate on validation set

LogitBoost

```r
pred.valid.LogitBoost<-predict(mod.LogitBoost,validation)
confusionMatrix(pred.valid.LogitBoost,validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E
##          A 55  1  0  1  0
##          B  0 28  1  0  0
##          C  0  0 25  1  0
##          D  0  2  0 28  0
##          E  0  0  0  0 35
##
## Overall Statistics
##
```

```
##                 Accuracy : 0.9661
##                   95% CI : (0.9277, 0.9875)
##      No Information Rate : 0.3107
##      P-Value [Acc > NIR] : < 2.2e-16
##
##                    Kappa : 0.9566
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9032   0.9615   0.9333   1.0000
## Specificity            0.9836   0.9932   0.9934   0.9864   1.0000
## Pos Pred Value         0.9649   0.9655   0.9615   0.9333   1.0000
## Neg Pred Value         1.0000   0.9797   0.9934   0.9864   1.0000
## Prevalence             0.3107   0.1751   0.1469   0.1695   0.1977
## Detection Rate         0.3107   0.1582   0.1412   0.1582   0.1977
## Detection Prevalence   0.3220   0.1638   0.1469   0.1695   0.1977
## Balanced Accuracy      0.9918   0.9482   0.9775   0.9599   1.0000
```

classification tree

```r
pred.valid.rpart<-predict(mod.rpart,validation)
confusionMatrix(pred.valid.rpart,validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E
##          A 51 14  7 13  5
##          B  1  9  4  6  6
##          C  2 14 19 13 13
##          D  0  0  0  0  0
##          E  1  0  0  0 14
##
## Overall Statistics
##
##                 Accuracy : 0.4844
##                   95% CI : (0.4118, 0.5574)
##      No Information Rate : 0.2865
##      P-Value [Acc > NIR] : 5.577e-09
##
##                    Kappa : 0.3343
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9273  0.24324  0.63333   0.0000  0.36842
## Specificity            0.7153  0.89032  0.74074   1.0000  0.99351
```

```
## Pos Pred Value         0.5667  0.34615  0.31148      NaN  0.93333
## Neg Pred Value         0.9608  0.83133  0.91603   0.8333  0.86441
## Prevalence             0.2865  0.19271  0.15625   0.1667  0.19792
## Detection Rate         0.2656  0.04688  0.09896   0.0000  0.07292
## Detection Prevalence   0.4688  0.13542  0.31771   0.0000  0.07812
## Balanced Accuracy      0.8213  0.56678  0.68704   0.5000  0.68096
```

Prediction on test set

LogitBoost

```r
pred.test.LogitBoost<-predict(mod.LogitBoost,testingNew)
table(pred.test.LogitBoost,testingNew$classe)
```

```
##
## pred.test.LogitBoost   A   B   C   D   E
##                    A 271   4   4   5   0
##                    B   3 141   3   0   2
##                    C   1   3 118   5   4
##                    D   3   3   3 128   2
##                    E   0   3   2   0 164
```

classification tree

```r
pred.test.rpart<-predict(mod.rpart,testingNew)
table(pred.test.rpart,testingNew$classe)
```

```
##
## pred.test.rpart   A   B   C   D   E
##               A 269  80  36  72  18
##               B   4  56  29  34  22
##               C  18  57  97  60  59
##               D   0   0   0   0   0
##               E   2   0   0   0  87
```

The "logistic regression with boosting" model give a very high accuracy (~97%) on validation and testing set with only few classes predicted wrong. The comparion of actual and predicted classes showing minor error

In sample vs. out of sample error of LogitBoost (best accuracy model)

```r
#In Sample
pred.train.LogitBoost<-predict(mod.LogitBoost,trainingNew)
NAS<- which(is.na(pred.train.LogitBoost))
1-sum((pred.train.LogitBoost[-NAS]==trainingNew[-NAS,'classe'])*1)/length(pred.train.LogitBoost)
```

```
## [1] 0.1303191
```

```r
#Out Sample
NAS<- which(is.na(pred.test.LogitBoost))
1-sum((pred.test.LogitBoost[-NAS]==testingNew[-NAS,'classe'])*1)/length(pred.test.LogitBoost)
```

```
## [1] 0.178
```

The out of sample error is higher than the in sample error because it was tested on new unseen data but generaly it is very low