Practical Machine Learning Project

Thinh Nguyen

February 5, 2018

Overview:

Background

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. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be 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 from the website here:HAR (see the section on the Weight Lifting Exercise Dataset).

The training data for this project are available here:

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

The test data are available here:

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

```
training<-read.csv("./pml-training.csv")
testing<-read.csv("./pml-testing.csv")
```

Data Preprocessing

First of all, we give the data a first look: is there any missing values?

```
unique(sapply(training,function(i) i%>%is.na%>%sum))
```

```
## [1]     0 19216
```

So there are only two types of columns: no missing values and contains 19216 missing values.

```
table(sapply(training,function(i) i%>%is.na%>%sum))
```

```
##
##     0 19216
##    93    67
```

There are 93 columns with no missing values and 67 columns with 19216 missing values. Is there any coincidence here since those columns all have the same number of missing values? We will look at the `new_window` colums to see what happened

```
table(training$new_window)
```

```
##
##    no   yes
## 19216   406
```

We can roughly see that the number of `no` in `new_window` is equal to the observed number of missing values. And indeed by closer looking at the data, we confirm that all missing values belong to the category `new_window=="no"`.

Now have a look at the testing data with variable `new_window`

```
testing$new_window
```

```
##  [1] no no no no no no no no no no no no no no no no no no no no
## Levels: no
```

They are all `no`. Technically we need to divide the data into two parts and build regression models for each part respectively. But in this project, we will only look at the subset of the data with `new_window="no"`. With this subseted data, we can remove all the colums with missing values without losing any information.

```
new_train<-subset(training,new_window=="no")
```

However, the missing values in the data can be the values `#DIV/0!` or `""`. We remove all the colums with missing values

```
new_train[new_train==""]<-NA;
new_train[new_train=="#DIV/0!"]<-NA;
cleanedTrain <- new_train[, colSums(is.na(new_train)) == 0]
cleanedTest <- testing[, colSums(is.na(testing)) == 0]
dim(cleanedTrain)
```

```
## [1] 19216    60
```

```
dim(cleanedTest)
```

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

After cleaning the missing data, the training data and the testing data contains 60 variables. However, the variables containing the information of users, timestamp and windows don't contribute to the regression then we remove all of these variables.

```
cleanedTrain<-cleanedTrain[,-c(1:7)]
cleanedTest<-cleanedTest[,-c(1:7)]
```

Finally, the data contains 53 variables.

```
inTrain <- createDataPartition(cleanedTrain$classe, p = 0.7, list = FALSE)
train <- cleanedTrain[inTrain, ]
valid <- cleanedTrain[-inTrain, ]

x_predictor<-cleanedTest[,-53]
```

Building models

Classification Trees

```
fit_rpart <- train(classe ~ ., data = train, method = "rpart")
print(fit_rpart, digits = 4)
```

```
## CART
##
## 13453 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13453, 13453, 13453, 13453, 13453, 13453, ...
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03668  0.5172    0.37645
##   0.06131  0.4128    0.20274
##   0.11306  0.3228    0.05834
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03668.
```

The accuracy of the algorithm

```
predict_rpart <- predict(fit_rpart, valid)
confusionMatrix(valid$classe, predict_rpart)$overall[1]
```

```
##  Accuracy
## 0.4915842
```

The accuracy of the classification tree algorithm is too poor. We will investigate the accuracy of Random Forest

Random Forest

```
fit_rf<-train(classe ~ ., data = train, method = "rf",trControl=trainControl(method = "cv", number = 5)
print(fit_rpart, digits = 4)
```

```
## CART
##
## 13453 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13453, 13453, 13453, 13453, 13453, 13453, ...
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03668  0.5172    0.37645
##   0.06131  0.4128    0.20274
##   0.11306  0.3228    0.05834
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03668.
```

The accuracy of the algorithm

```
predict_rf <- predict(fit_rf, valid)
confusionMatrix(valid$classe, predict_rf)$overall[1]
```

```
##  Accuracy
## 0.9946209
```

The accuracy of Random Forest is 99.24% which is too high and is the signal of overfitting.

Generalised Boosting Model

Construct the model

```
gbm_fit<-train(classe ~ ., data = train, method = "gbm",trControl=trainControl(method = "repeatedcv", nu
```

The accuracy of the algorithm

```
predict_gbm <- predict(gbm_fit, valid)
confusionMatrix(valid$classe, predict_gbm)$overall[1]
```

```
##  Accuracy
## 0.9635606
```

So we have built three models one with very poor accuracy and one with a sign of overfitting. We decide to use that last one Generalised Boosting Model for the prediction

```
pred<-predict(gbm_fit,x_predictor)
pred
```

```
##  [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
```