Predict Exercise Quality Through Wearable Device Data

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. 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: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Read Data

We first read the training and testing data sets.

```
setwd("/Users/liang.huang/Documents/R/Coursera/machine learning/")
library(caret)
library(randomForest)
data<-read.csv("pml-training.csv", na.strings=c("NA",""))
testing <- read.csv("pml-testing.csv", na.strings=c("NA",""))
dim(data); dim(testing) # dimensions of the data sets read in
```

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

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

Create Training and Validation Sets

Next, we divide the first data set into training and validation data sets, with 70% records in the testing set, and 30% in the validation set.

```
# create training set and validation set
set.seed(13398)
inTraining<-createDataPartition(data$classe, p=0.7, list=FALSE)
training<-data[inTraining,]
validation<-data[-inTraining,]
```

Clean Data

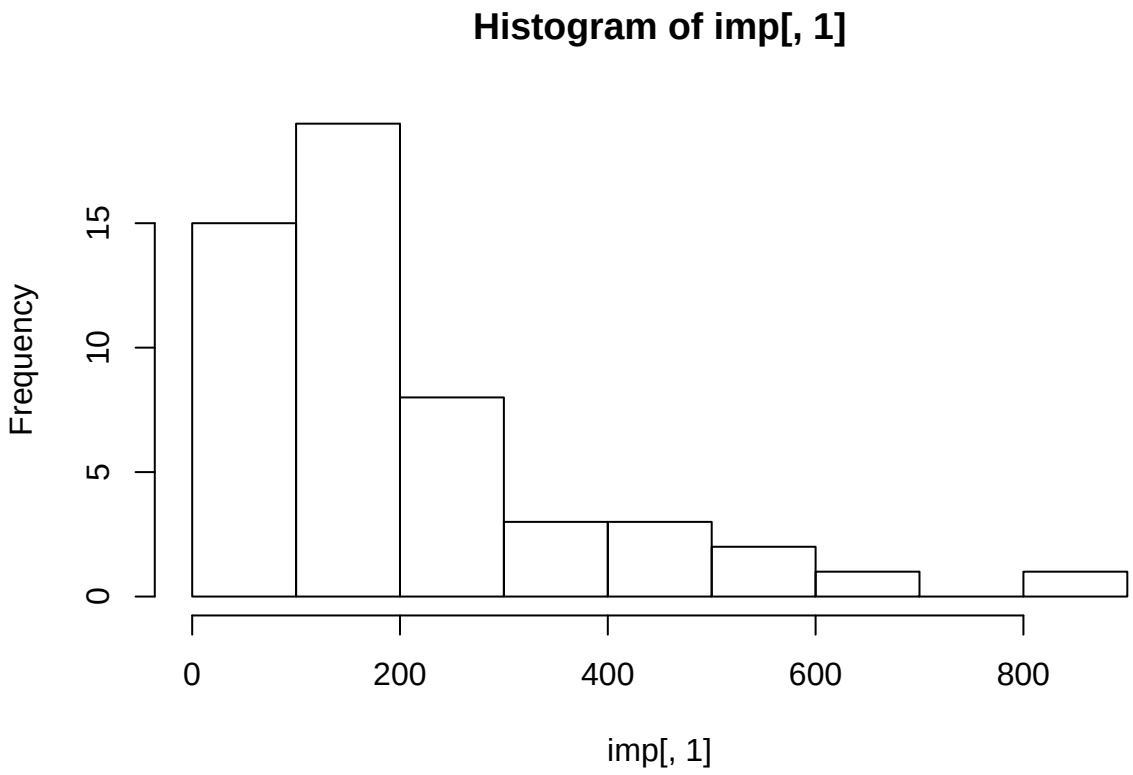
Exploratory analysis reveals that some columns have over 90% rows with N/A or missing value, yet others have no N/A value. We remove these mostly N/A columns. We also remove the first 7 columns that are not suitable to be predictors, such as user names, time stamps, etc.

```
training<-training[,colSums(is.na(training)) == 0] # remove columns with N/As
training<-training[,-c(1:7)] # remove the first 7 columns that are not predictors
```

Initial Model Fitting and Feature Selection

The random forest algorithm is a popular classification method and is good with nonlinear features. However, it can also overfit. Through constructing an initial random forest and visualizing the distribution of each variable's importance, we noticed that 250 separates a small number (13) of important variables from a large number of less important variables. We will keep those higher than 250 for model prediction.

```
model<-randomForest(classe~.,data=training) # initial random forest model
imp<-importance(model) # get importance values
hist(imp[,1]) # plot a distribution histogram
```




```
# keep important variables and classe
training<-training[,c(imp[,1]>=250,TRUE)]
```

Model Training

Now we fit the reduced training data set with a random forest model.

```
model<-randomForest(classe~.,data=training)
confusionMatrix(training$classe,predict(model,training))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3906    0    0    0    0
##          B    0 2658    0    0    0
```

```
##          C    0    0 2396    0    0
##          D    0    0    0 2252    0
##          E    0    0    0    0 2525
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

The model correctly classifies 100% observations in the training set.

Model Validation

We run this model on the validation data set.

```
confusionMatrix(validation$classe,predict(model,validation))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1668    3    0    2    1
##          B    3 1111   15   10    0
##          C    4   12 1000   10    0
##          D    0    0    6  950    8
##          E    0    5    1    3 1073
##
## Overall Statistics
##
##                Accuracy : 0.9859
##                  95% CI : (0.9825, 0.9888)
##     No Information Rate : 0.2846
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9822
##  Mcnemar's Test P-Value : 0.003005
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9958   0.9823   0.9785   0.9744   0.9917
## Specificity            0.9986   0.9941   0.9947   0.9971   0.9981
## Pos Pred Value         0.9964   0.9754   0.9747   0.9855   0.9917
## Neg Pred Value         0.9983   0.9958   0.9955   0.9949   0.9981
## Prevalence             0.2846   0.1922   0.1737   0.1657   0.1839
## Detection Rate         0.2834   0.1888   0.1699   0.1614   0.1823
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9972   0.9882   0.9866   0.9858   0.9949
```

The model achieves excellent results on the validation data set.

Note that **we do not** need to remove columns in the validation and testing data sets. Apparently predict(model,newdata) matches column names in newdata with those in the model.

Test Data

Finally we predict the 20 test case observations

```
answers<-predict(model,testing)
```

And we generate the answer files for submission.

```
# generate answers
pml_write_files = function(x){
    n = length(x)
    for(i in 1:n){
        filename = paste0("problem_id_",i,".txt")
        write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
    }
}
pml_write_files(answers)
```