Practical Machine Learning Course Project Write-up

Sylvia Seow

December 23, 2015

Executive Summary

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).

Input Data

We will initialise by loading neccessary library

```
library(caret)
```

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

```
library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(rpart)
set.seed(8888)
```

High level description of the raw data and detailed data description for this project has come from source: http://groupware.les.inf.puc-rio.br/har.

We will load the csv file downloaded into R

```
train<-read.csv("pml-training.csv",na.strings=c("NA",""), strip.white = T)
test <-read.csv("pml-testing.csv", na.strings=c("NA",""), strip.white = T)
```

Formatting and cleaning data

Below code fragment is to clean and prepare the dataset for further processing, that step including the treatment of null value for data

```
isNA.train <- apply(train, 2, function(x) { sum(is.na(x)) })
isNA.test <- apply(test, 2, function(x) { sum(is.na(x)) })
training <- subset(train[, which(isNA.train == 0)],
                   select=-c(X, user_name, new_window, num_window, raw_timestamp_part_1, raw_timestamp_
testing <- subset(test[, which(isNA.test == 0)],
                              select=-c(X, user_name, new_window, num_window, raw_timestamp_part_1, raw_
```

Cross Validation/data spilting

We will create data partitition 60% for training and testing data. The method we use here is just simple hold-out, by spilting data into 2 set, which is training and another for testing

```
pml.training.index <- createDataPartition(y=training$classe,p=0.6,list=FALSE)
pml.training.train <- training[pml.training.index,]
pml.training.test <- training[-pml.training.index,]

tc <- trainControl("cv",10,savePred=T)
dim(pml.training.train)
```

```
## [1] 11776    53
```

```
dim(pml.training.test)
```

```
## [1] 7846   53
```

Analyse (Model Testing & Selection)

Below model used as shown:

Linear Discriminative Analysis

```
model.lda <- train(classe ~., method="lda",trControl=tc, data=pml.training.train)
```

```
## Loading required package: MASS
```

```
confusionMatrix(pml.training.train$classe, predict(model.lda, pml.training.train))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2757   67  267  247   10
##          B  345 1445  292   88  109
##          C  188  213 1364  242   47
##          D  110   73  250 1413   84
##          E   77  372  213  198 1305
##
```

```
## Overall Statistics
##
##                Accuracy : 0.7035
##                  95% CI : (0.6951, 0.7117)
##     No Information Rate : 0.2953
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6248
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7929   0.6659   0.5717   0.6458   0.8392
## Specificity            0.9288   0.9132   0.9265   0.9461   0.9159
## Pos Pred Value         0.8235   0.6341   0.6641   0.7321   0.6028
## Neg Pred Value         0.9146   0.9237   0.8949   0.9213   0.9740
## Prevalence             0.2953   0.1843   0.2026   0.1858   0.1320
## Detection Rate         0.2341   0.1227   0.1158   0.1200   0.1108
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.8609   0.7895   0.7491   0.7959   0.8775
```

Trees

```
model.tree <- train(classe ~., method="rpart",trControl=tc, data=pml.training.train)
confusionMatrix(pml.training.train$classe, predict(model.tree, pml.training.train))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2071   10  646  611   10
##          B  359  402  396 1122    0
##          C   48   49 1375  582    0
##          D  114   13  550 1253    0
##          E   29   15  409  731  981
##
## Overall Statistics
##
##                Accuracy : 0.5165
##                  95% CI : (0.5074, 0.5255)
##     No Information Rate : 0.3651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3981
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7902  0.82209   0.4073   0.2915  0.98991
## Specificity            0.8605  0.83370   0.9192   0.9095  0.89022
```

```
## Pos Pred Value         0.6186  0.17639   0.6694   0.6492  0.45312
## Neg Pred Value         0.9347  0.99084   0.7942   0.6906  0.99896
## Prevalence             0.2226  0.04153   0.2867   0.3651  0.08415
## Detection Rate         0.1759  0.03414   0.1168   0.1064  0.08331
## Detection Prevalence   0.2843  0.19353   0.1744   0.1639  0.18385
## Balanced Accuracy      0.8253  0.82789   0.6632   0.6005  0.94006
```

Random Forest with cross validation using random subsampling

```
##model.randForest <- train(classe ~., model=FALSE, method="rf",trControl=tc, data=pml.training.train,n
##confusionMatrix(pml.training.train$classe, predict(model.randForest, pml.training.train))

first_seed <- 888
accuracies <-c()
for (i in 1:3){
       set.seed(first_seed)
       first_seed <- first_seed+1
       trainIndex <- createDataPartition(y=training$classe, p=0.6, list=FALSE)
       trainingSet<- training[trainIndex,]
       testingSet<- training[-trainIndex,]
       modelFit <- randomForest(classe ~., data = trainingSet)
       prediction <- predict(modelFit, testingSet)
       testingSet$rightPred <- prediction == testingSet$classe
       t<-table(prediction, testingSet$classe)
       print(t)
       accuracy <- sum(testingSet$rightPred)/nrow(testingSet)
       accuracies <- c(accuracies,accuracy)
       print(accuracy)
}
```

```
##
## prediction    A    B    C    D    E
##          A 2232   11    0    0    0
##          B    0 1506   11    0    0
##          C    0    1 1356   17    1
##          D    0    0    1 1269    5
##          E    0    0    0    0 1436
## [1] 0.9940097
##
## prediction    A    B    C    D    E
##          A 2225    9    0    0    0
##          B    3 1506    9    0    0
##          C    0    3 1353   12    0
##          D    0    0    6 1274    2
##          E    4    0    0    0 1440
## [1] 0.9938822
##
## prediction    A    B    C    D    E
##          A 2232   14    0    0    0
##          B    0 1503   12    0    0
##          C    0    1 1353   22    0
##          D    0    0    3 1264    6
```

```
##          E    0    0    0    0 1436
## [1] 0.9926077
```

Details on Random Forest Model

It seems that random forest provide the result with best "accuracy". the We then use the model to predict the classe value for the 6 participants in the testing dataset. We also apply the model on the validation dataset to determine the accuracy of the selected model. The OOB estimate of error is 0.65% which is excellent, the Confusion matrix looks good too. Next, we will take the look at the variable importance.

```
var.imp <- varImp(modelFit)
var.imp$variable_name <- row.names(var.imp)
var.imp[order(var.imp$Overall, decreasing=TRUE),]
```

```
##                       Overall        variable_name
## roll_belt           735.41452           roll_belt
## pitch_forearm       500.47121        pitch_forearm
## yaw_belt            499.80266             yaw_belt
## magnet_dumbbell_z   452.65830    magnet_dumbbell_z
## magnet_dumbbell_y   418.52440    magnet_dumbbell_y
## pitch_belt          393.89334          pitch_belt
## roll_forearm        341.83259         roll_forearm
## magnet_dumbbell_x   281.05149    magnet_dumbbell_x
## roll_dumbbell       259.72326        roll_dumbbell
## accel_dumbbell_y    247.03647     accel_dumbbell_y
## accel_belt_z        245.44865         accel_belt_z
## magnet_belt_z       245.31080        magnet_belt_z
## magnet_belt_y       224.47173        magnet_belt_y
## roll_arm            202.76952             roll_arm
## gyros_belt_z        202.13377         gyros_belt_z
## accel_dumbbell_z    195.71808     accel_dumbbell_z
## accel_forearm_x     195.04247      accel_forearm_x
## magnet_forearm_z    164.64038     magnet_forearm_z
## total_accel_dumbbell 159.03434 total_accel_dumbbell
## yaw_dumbbell        158.74499         yaw_dumbbell
## magnet_arm_x        158.27977         magnet_arm_x
## gyros_dumbbell_y    153.91726     gyros_dumbbell_y
## accel_arm_x         152.68140          accel_arm_x
## accel_dumbbell_x    148.16263     accel_dumbbell_x
## magnet_belt_x       145.42956        magnet_belt_x
## total_accel_belt    143.83698     total_accel_belt
## magnet_arm_y        143.69548         magnet_arm_y
## accel_forearm_z     141.36289      accel_forearm_z
## yaw_arm             133.33940              yaw_arm
## magnet_forearm_y    130.61111     magnet_forearm_y
## magnet_forearm_x    126.75482     magnet_forearm_x
## magnet_arm_z        113.82206         magnet_arm_z
## pitch_dumbbell      105.50313       pitch_dumbbell
## pitch_arm           103.96155            pitch_arm
## yaw_forearm          99.60425          yaw_forearm
## accel_arm_y          95.98355          accel_arm_y
## accel_forearm_y      85.51384      accel_forearm_y
## gyros_arm_x          81.33736          gyros_arm_x
```

```
## accel_arm_z           80.95126          accel_arm_z
## gyros_arm_y           79.32727          gyros_arm_y
## accel_belt_y          78.72467         accel_belt_y
## gyros_dumbbell_x      77.47898     gyros_dumbbell_x
## total_accel_forearm   75.56275  total_accel_forearm
## gyros_forearm_y       75.04185      gyros_forearm_y
## accel_belt_x          73.68805         accel_belt_x
## gyros_belt_y          68.00562         gyros_belt_y
## total_accel_arm       64.16575      total_accel_arm
## gyros_belt_x          58.98826         gyros_belt_x
## gyros_dumbbell_z      55.17736     gyros_dumbbell_z
## gyros_forearm_z       51.87992      gyros_forearm_z
## gyros_forearm_x       45.00914      gyros_forearm_x
## gyros_arm_z           37.48512          gyros_arm_z
```

we will apply the model to validation dataset and testing dataset from csv file

```
pml.val <- predict(modelFit,newdata=pml.training.test)
pml.pred <- predict(modelFit,newdata=testing)
result.test <-predict(modelFit,testing)
```

Testing & Result

Let's calculate the Out of Sample Error rate, or generalisation error, and the accuracy of the model based on the validation sub set of data that was used.

```
#calculate error rate and accuracy of the validation
ose.acc <- sum(pml.val == pml.training.test$classe)/length(pml.val)
ose.err <- (1 - ose.acc)
##show confusion matrix
confusionMatrix(pml.training.test$classe,pml.val)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    6 1512    0    0    0
##          C    0    1 1366    1    0
##          D    0    0    8 1278    0
##          E    0    0    0    2 1440
##
## Overall Statistics
##
##                Accuracy : 0.9977
##                  95% CI : (0.9964, 0.9986)
##     No Information Rate : 0.2852
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9971
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9973   0.9993   0.9942   0.9977   1.0000
## Specificity            1.0000   0.9991   0.9997   0.9988   0.9997
## Pos Pred Value         1.0000   0.9960   0.9985   0.9938   0.9986
## Neg Pred Value         0.9989   0.9998   0.9988   0.9995   1.0000
## Prevalence             0.2852   0.1928   0.1751   0.1633   0.1835
## Detection Rate         0.2845   0.1927   0.1741   0.1629   0.1835
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9987   0.9992   0.9969   0.9982   0.9998
```

```
## Accuracy
ose.acc
```

```
## [1] 0.9977058
```

```
## Error Rate
ose.err
```

```
## [1] 0.002294163
```

Evaluation

The achieved error value is below 5% and the prediction accurary close to 100%. So this is the best model to be used, although it does a long time to generate the model.

The final result on the testing dataset (test csv) is 20 correct prediction out of 20. So the accuracy is 100%

Submission for grading

```
## evaluate testing
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(result.test)
```