Peer graded assignment: Machine Learning

Predicting exersise type using devices

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. In this project, the 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.

- **Build predicting models and compare their efficiency.**
- **Discuss sample errors.**
- **Predict 20 samples for project deliverable.**

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har

I'm behind a proxy , so I download files over HTTP and load them from drive

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

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

Load data

```
file_testing <- paste(getwd(),"files","pml-testing.csv",sep="/")
file_training<- paste(getwd(),"files","pml-training.csv",sep="/")

data_testing <- read.csv(file_testing, na.strings=c("NA",""), header=TRUE)
data_training <- read.csv(file_training, na.strings=c("NA",""), header=TRUE)

inTrain <- createDataPartition(y=data_training$classe, p=0.6, list=FALSE)
```

I will have 3 tier work flow:

- 1. Training - model fit
- 2. Testing - model check
- 3. Production - predicting 20 observations for course project

```
training <- data_training[inTrain, ]
testing <- data_training[-inTrain, ]
production <- data_testing
```

Preprocessing and cleaning

```
cleanData <- function(dataset)
{
  print(paste("Cleaning data with number of columns: ", ncol(dataset)))
  # Drop near zero variable columns
  nzvdata <- nearZeroVar(dataset, saveMetrics=TRUE)
  bad_columns_nzv <- nzvdata[nzvdata$nzv==TRUE | nzvdata$zeroVar==TRUE,]
  bad_colnames_nzv <- row.names(bad_columns_nzv)
```

```
  #count NA percentage per column of data
  rows <- nrow(dataset)
  na_count <-sapply(dataset, function(y) sum(length(which(is.na(y))))/rows)

  #Some NA ratios are very high... exclude them too
  columns_na_count <- data.frame(colnames(dataset), na_count)
  table(columns_na_count[,2])

  #100 of 160 variables are NA 98% of the time. This cannot be benefitial to the analysis
  #Delete them

  column_names_na <- subset(columns_na_count,na_count>0)

  bad_column_names <- c(bad_colnames_nzv,as.character(column_names_na[,1]))
  dataset <- dataset[,-which(names(dataset) %in% bad_column_names)]
  #ID and Usename column is not relevant to data analysis, drop it
  dataset <- dataset[,-c(1,2)]

  debugging <- FALSE
  if(debugging)
  {
    dataset <- dataset[1:20,]
  }
  print(paste("Returning data with number of columns: ", ncol(dataset)))
  dataset
}

training      <- cleanData(training)
```

```
## [1] "Cleaning data with number of columns:  160"
## [1] "Returning data with number of columns:  57"
```

```
testing       <- cleanData (testing)
```

```
## [1] "Cleaning data with number of columns:  160"
## [1] "Returning data with number of columns:  57"
```

```
production    <- cleanData (production)
```

```
## [1] "Cleaning data with number of columns:  160"
## [1] "Returning data with number of columns:  57"
```

Fitting decision tree model - rpart

```
modelRPart <- train(classe ~ ., data=training, method="rpart")
accuracy_rpart_is <- modelRPart$results[[2]][1]
#fancyRpartPlot(modelRPart$finalModel) # no need to print if it's no good

pred <- predict(modelRPart, testing)
matrix <- confusionMatrix(pred, testing$classe)
print(matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1710  162    4    0    1
##          B  394  870  250  483  126
##          C  112  422 1102  694  338
##          D    0    0    0    0    0
##          E   16   64   12  109  977
##
## Overall Statistics
##
##                Accuracy : 0.5938
##                  95% CI : (0.5828, 0.6047)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4876
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7661   0.5731   0.8056   0.0000   0.6775
## Specificity            0.9703   0.8020   0.7583   1.0000   0.9686
## Pos Pred Value         0.9110   0.4098   0.4130      NaN   0.8294
## Neg Pred Value         0.9125   0.8868   0.9486   0.8361   0.9303
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2179   0.1109   0.1405   0.0000   0.1245
## Detection Prevalence   0.2392   0.2706   0.3400   0.0000   0.1501
## Balanced Accuracy      0.8682   0.6876   0.7819   0.5000   0.8231
```

```
accuracy_rpart_oos <- matrix$overall[[1]]

cat("\nIn sample accuracy: ", accuracy_rpart_is,"\nOut of sample accuracy: ",accuracy_rpart_oos, "\n")
```

```
##
## In sample accuracy:  0.5102432
## Out of sample accuracy:  0.5938058
```

Not the greatest accuracy for the rpart model - 0.5681. In sample errors are less than out of sample as expected. Which makes it even worse for the production application.

Fitting random forest model

```
modelRF <- randomForest(classe ~. , data=training)
modelRF
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training)
##                Type of random forest: classification
```

```
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.16%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3348    0    0    0    0 0.000000000
## B    2 2276    1    0    0 0.001316367
## C    0    3 2049    2    0 0.002434275
## D    0    0    6 1923    1 0.003626943
## E    0    0    0    4 2161 0.001847575
```

```r
predictRF<- predict(modelRF, testing, type = "class")
print(confusionMatrix(predictRF, testing$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    0    0    0    0
##          B    1 1517    6    0    0
##          C    0    1 1355    5    0
##          D    0    0    7 1281    0
##          E    0    0    0    0 1442
##
## Overall Statistics
##
##                Accuracy : 0.9975
##                  95% CI : (0.9961, 0.9984)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9968
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9993   0.9905   0.9961   1.0000
## Specificity            1.0000   0.9989   0.9991   0.9989   1.0000
## Pos Pred Value         1.0000   0.9954   0.9956   0.9946   1.0000
## Neg Pred Value         0.9998   0.9998   0.9980   0.9992   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1933   0.1727   0.1633   0.1838
## Detection Prevalence   0.2843   0.1942   0.1735   0.1642   0.1838
## Balanced Accuracy      0.9998   0.9991   0.9948   0.9975   1.0000
```

Amazing improvement with random forrest model - 99.75% accuracy! Out-of-bag estimate of error rate of the model was 0.15%, which is exactly how it turned out with out-of-sample data!

Production application

Here i'm battling problems where Production dataset has different factor levels .. and random forest really does not like that This solution is from http://stackoverflow.com/a/36170319

```r
common <- intersect(names(training), names(production))
for (p in common) {
  if (class(training[[p]]) == "factor")
  {
    levels(production[[p]]) <- levels(training[[p]])
  }
}

predictAssignment <- predict(modelRF, production, type = "class")
predictAssignment
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

The answer was 100% match. Out of sample error on production dataset 0% (j/k)