Practical Machine Learning / Prediction Assignment

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 data set, the participants 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).

In this project, our goal is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which praticipants did the exercise, which is stored in the target variable "classe".

Download and load the data

Download the data.

```
url <- "http://d396qusza40orc.cloudfront.net/predmachlearn/"
trainingDataFile <- "pml-training.csv"
testingDataFile <- "pml-testing.csv"

dataDir <- "./"

downloadData <- function() {
   print(paste(url,trainingDataFile,sep=""))
   if (!file.exists(trainingDataFile)) {
      print("Downloading data file")
         download.file(paste(url,trainingDataFile,sep=""), destfile = trainingDataFile)
   } else {
     print("Training file already exists. Skipping download...")
   }

   print(paste(url,testingDataFile,sep=""))
   if (!file.exists(testingDataFile)) {
      print("Downloading data file")
         download.file(paste(url,testingDataFile,sep=""), destfile = testingDataFile)
   } else {
     print("Test file already exists. Skipping download...")
   }

   print("Done downloading the data")
}

downloadData()
```

```
## [1] "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
## [1] "Training file already exists. Skipping download..."
```

```
## [1] "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
## [1] "Test file already exists. Skipping download..."
## [1] "Done downloading the data"
```

Load the data:

```
trainingData = read.csv(paste(dataDir,trainingDataFile,sep=""), na.strings=c("", "NA"))
testingData = read.csv(paste(dataDir,testingDataFile,sep=""), na.strings=c("", "NA"))
```

Clean up

Next we check the summary of the data:

```
summary(trainingData)
summary(testingData)
```

Summarizing the data reveals that there are many NAs for certain features in the data (most of them have ~300 non-NAs out of 19622). So by removing them, hopefully, we won't lose much. We can always return to this point if we see that our final prediction model is performing poorly.

```
dim(trainingData)
```

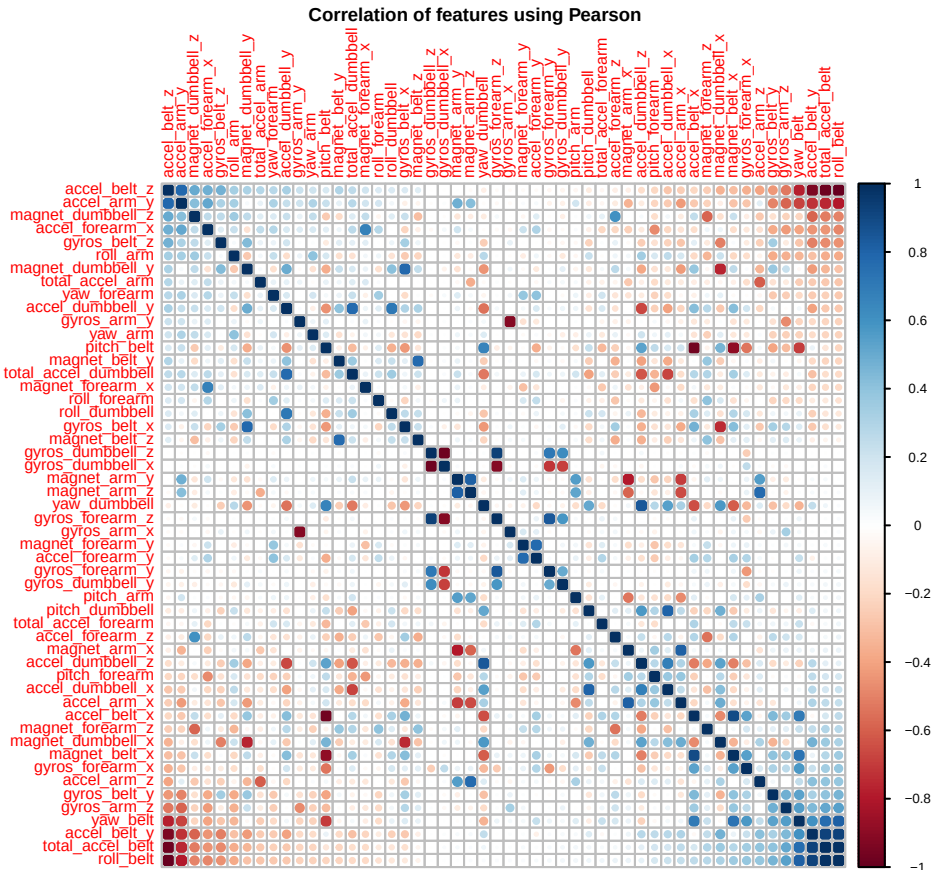
```
## [1] 19622   160
```

```
# print(colSums(is.na(trainingData)))
# keep the ones with less than 100 NAs
trainingData.clean <- trainingData[ , colSums(is.na(trainingData)) < 100]
```

We will next remove the features that doesn't seem to be useful as a predictor, such as timestamps.

```
notVeryUsefulFeatures = c('X', 'user_name', 'raw_timestamp_part_1', 'raw_timestamp_part_2', 'cvtd_times
trainingData.clean <- trainingData.clean[, -which(names(trainingData.clean) %in% notVeryUsefulFeatures)]
```

Now let's check the correlations between the remaining features:



Let's remove the highly correlated (> 0.90) features:

```
highlyCorrelatedFeatures = findCorrelation(corrMatrix, cutoff = .90)
names(trainingData.clean)[highlyCorrelatedFeatures]
```

```
## [1] "roll_belt"         "magnet_forearm_z" "magnet_forearm_x"
## [4] "pitch_forearm"     "accel_arm_y"      "accel_arm_x"
## [7] "roll_dumbbell"
```

```
trainingData.clean = trainingData.clean[,-highlyCorrelatedFeatures]
dim(trainingData.clean)
```

```
## [1] 19622    46
```

The above cleanup removed a total of 114 features out of 160, and now we only have 46 features.

Model building

First let's split the given data set into training and test sets:

```
numRows <- nrow(trainingData.clean)
numCols <- ncol(trainingData.clean)
train <- sample(numRows, 0.8*numRows)
```

```
test <- sample(setdiff(seq_len(numRows), train), 0.2*numRows)
trainingSet <- trainingData.clean[train,]
testSet <- trainingData.clean[test,]
```

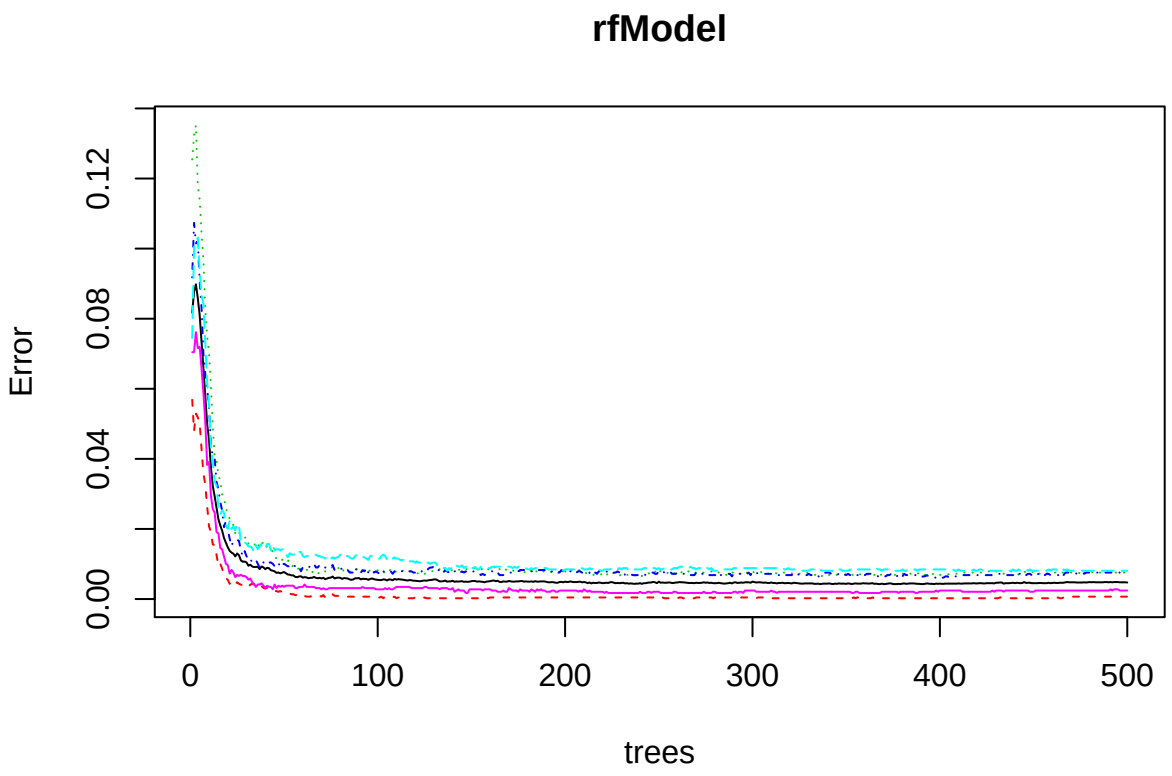
This selection splits the data randomly into 2 sets: 15697 samples for training and 3924 samples for testing.

Let's try to fit a Random Forest to the data using 500 trees and trying 10 variables for each split :

```
set.seed(0)
rfModel <- randomForest::randomForest(classe ~ .,
      data=trainingSet,
      ntree=500,
      mtry=7,
      importance=TRUE,
      na.action=randomForest::na.roughfix,
      replace=FALSE)
rfModel
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = trainingSet, ntree = 500,      mtry = 7, importance = TRU
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.47%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4377    3    0    0    0 0.0006849315
## B   18 3022    5    0    0 0.0075533662
## C    0   19 2760    1    0 0.0071942446
## D    0    0   17 2581    4 0.0080707148
## E    0    0    1    6 2883 0.0024221453
```

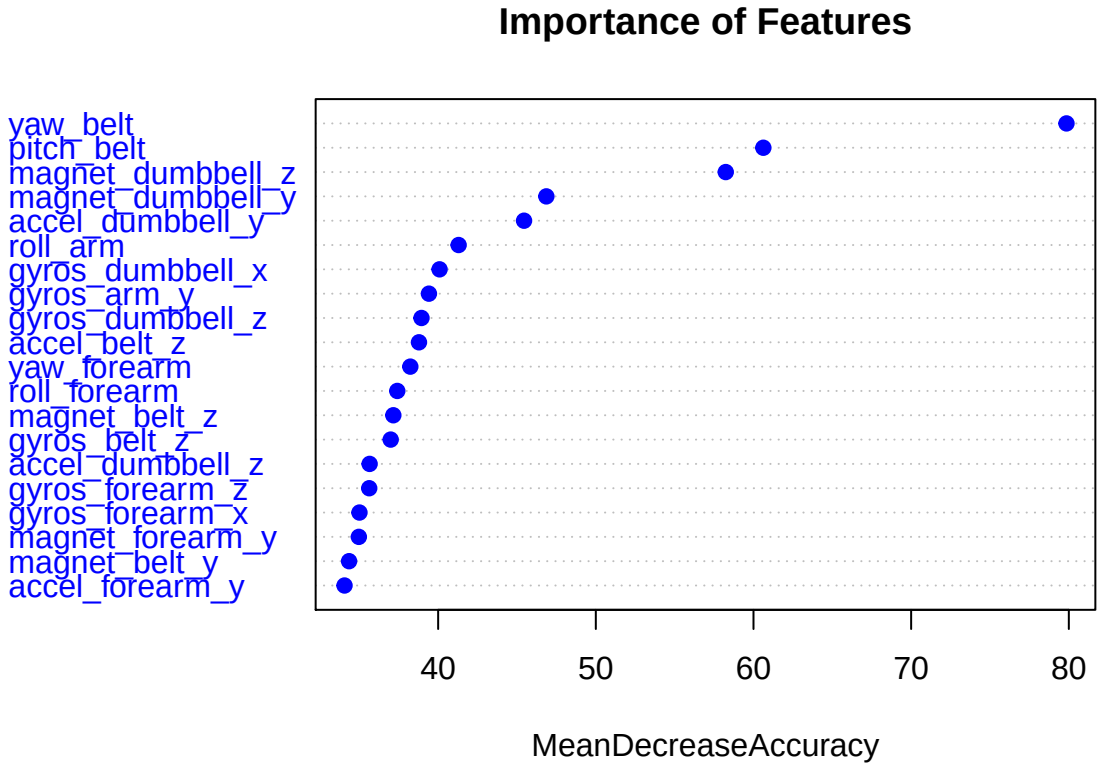
```
plot(rfModel)
```



The above "Error vs. number of trees" plot clearly show that we didn't need all the 500 trees, but could have easily got the same result with ~100 trees.

Here are the important features:

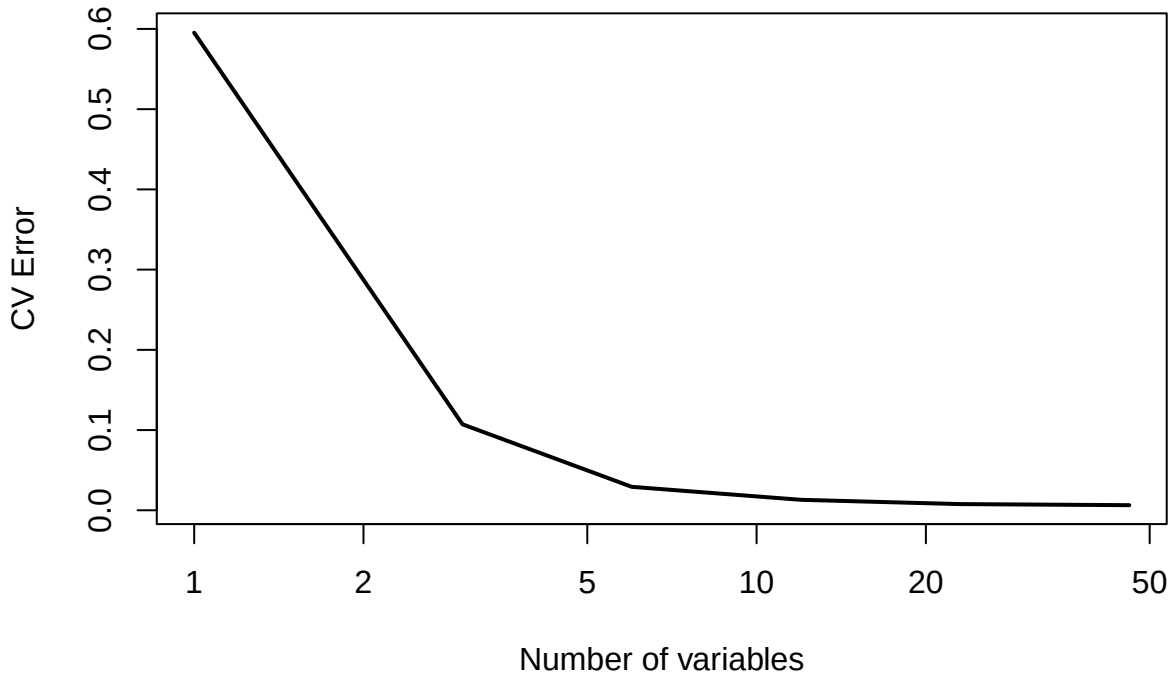
```
varImpPlot(rfModel, n.var=20, sort = TRUE, type = 1, pch = 19, col = "blue", cex = 1, main = "Importanc
```



Cross Validation

We are going to check and optimize the performance of the Random Forest model using cross validation.

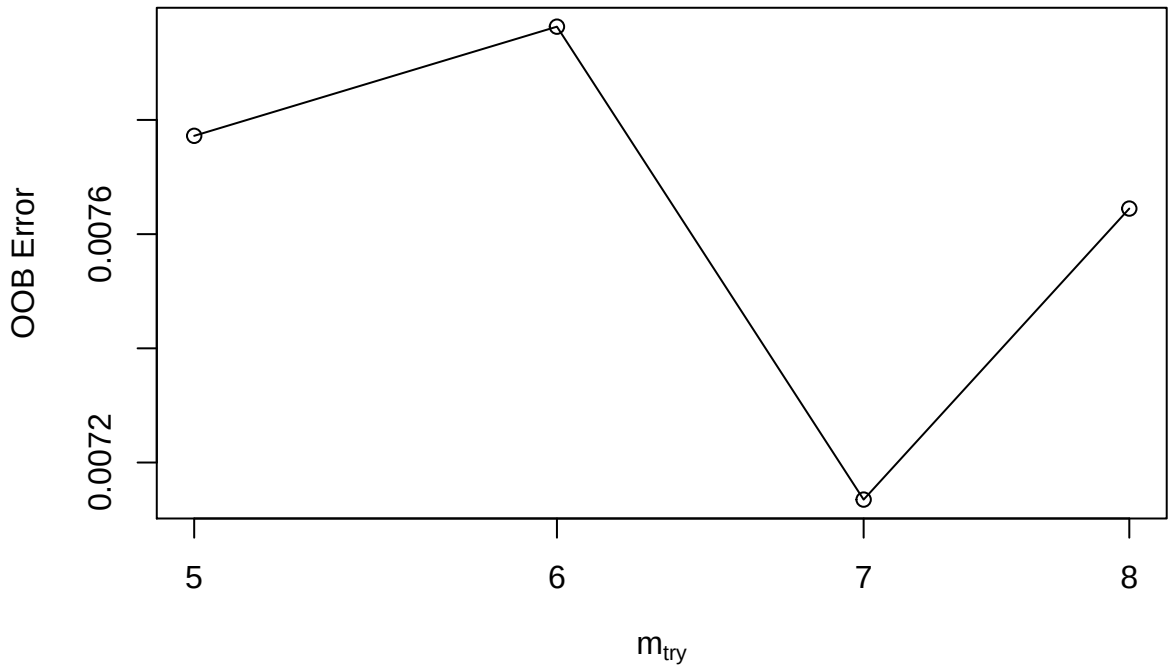
```
set.seed(0)
numColsTS <- ncol(trainingSet)
numRowsTS <- nrow(trainingSet)
extTrainingSet <- cbind(trainingSet[1:numColsTS-1], matrix(runif(numRowsTS), numRowsTS, 1))
result <- replicate(2, rfcv(extTrainingSet, trainingSet$classe), simplify=FALSE)
error.cv <- sapply(result, "[[", "error.cv")
matplot(result[[1]]$n.var, cbind(rowMeans(error.cv), error.cv), type="l",
        lwd=c(2, rep(1, ncol(error.cv))), col=1, lty=1, log="x",
        xlab="Number of variables", ylab="CV Error")
```



So the the optimum value of mtry, the number of variables randomly sampled as candidates at each split, is around 10. We can tune it using tuneRF:

```
set.seed(0)
tuneRF(trainingSet[,-numColsTS], trainingSet[,numColsTS], stepFactor=0.9, improve=0.000001, trace=TRUE,
```

```
## mtry = 6  OOB error = 0.8%
## Searching left ...
## mtry = 7     OOB error = 0.71%
## 0.104 1e-06
## mtry = 8     OOB error = 0.76%
## -0.07142857 1e-06
## Searching right ...
## mtry = 5     OOB error = 0.78%
## -0.08928571 1e-06
```




```
##       mtry    OOBError
## 5.OOB    5 0.007772186
## 6.OOB    6 0.007963305
## 7.OOB    7 0.007135121
## 8.OOB    8 0.007644773
```

As can be seen above the best value, 7, is exactly what we have used in our model, so our model should give the minimum OOB (Out-of-Bag) error, which was estimated to be less than 1% for the training data. Now we let's try to verify this using the test set:

```
pred=predict(rfModel,testSet,type="class")
```

Here is the confusion matrix:

```
confusionMatrix = confusionMatrix(pred,testSet$classe)
confusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1199    2    0    0    0
##          B    0  749    1    0    0
##          C    0    0  639    8    0
##          D    1    0    2  606    0
##          E    0    0    0    0  717
##
## Overall Statistics
##
##                Accuracy : 0.9964
##                  95% CI : (0.994, 0.998)
```

```
##     No Information Rate : 0.3058
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9955
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9992   0.9973   0.9953   0.9870   1.0000
## Specificity            0.9993   0.9997   0.9976   0.9991   1.0000
## Pos Pred Value         0.9983   0.9987   0.9876   0.9951   1.0000
## Neg Pred Value         0.9996   0.9994   0.9991   0.9976   1.0000
## Prevalence             0.3058   0.1914   0.1636   0.1565   0.1827
## Detection Rate         0.3056   0.1909   0.1628   0.1544   0.1827
## Detection Prevalence   0.3061   0.1911   0.1649   0.1552   0.1827
## Balanced Accuracy      0.9992   0.9985   0.9964   0.9930   1.0000
```

```
accuracy <- confusionMatrix$overall[1]
accuracy
```

```
##  Accuracy
## 0.9964322
```

As seen above, random forest shows 99.6432212% accuracy rate on the test set (i.e. the error rate is less than 1% as expected), which shows that the model generalizes well beyond the training set.

Predictions for the Assignment Test Set

The random forest model seems to predict the target well for the training and test sets. Now let's compute the prediction values for the assignment test set:

```
answers <- predict(rfModel, testingData)
answers
```

```
##  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
```

Submitting the above results to the website showed 100% success rate, which also verified the quality of the random forest model we built.

```
path <- "prediction_files/"
unlink(path, recursive = TRUE, force = TRUE)
dir.create(path)
pml_write_files = function(x) {
    n = length(x)
    for (i in 1:n) {
        filename = paste0("problem_id_", i, ".txt")
        write.table(x[i], file = file.path(path,filename), quote = FALSE, row.names = FALSE, col.names =
    }
```

```
}

pml_write_files(answers)
```