Practical Machine Learning Project

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, the goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants and classify their exercise routine into 5 different classes or categories. The Weight Lifting dataset used for this project was downloaded from http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) .

Data Loading

The training data and test data provided as seperate CSV files were loaded with all missing values and dirty entries replaced by a NA to facilitate futher tiding up of the datasets.

```
require(caret)
require(randomForest)
# Set seed for reproducibility of results
set.seed(1250)

#Training data set
pmlData <- read.csv("pml-training.csv",sep=",",header=TRUE,na.strings=c("","NA","#DIV/0!"))
#Independent Valdiation/Test data set
validationData <- read.csv("pml-testing.csv",sep=",",header=TRUE,na.strings=c("","NA","#DIV/0!"))
```

Preprocessing of data and Feature Reduction

Training data set required additional preprocessing that included removal of features with more than 75% of NA values, features with Zero Variance, timestamp columns and other redundant features such as user_name. All preprocessing steps were encapsulated in a utilty function so that the function could be reused later for the validation data set. The data prepation set reduced the number of variables from 160 variables to a tighter set of 53 variables including the outcome variable Classe. Highly correlated variables were not removed from the training set.

```
prepare.data <- function(data) {

  # Remove columns that have 75% or above of NAs
  data <- data[,colSums(is.na(data)) <= nrow(data)*0.25]

  # Remove Timestamp variables
  data <- data[,-grep('timestamp',colnames(data))]

  # Remove unnecessary variables (X, person name,num_window)
  data <- data[,-which(names(data) %in% c("X","user_name","num_window"))]

  # Remove Zero Variance fields
  data <- data[,-nearZeroVar(data)]

  return (data)
}
```

Partioning of Training Data

The preprocessed Training data set was further partitioned into Train (60%) and Test (40%) data sub-sets. The model is built using the Train data subset and in-sample predictions are generated on the Test data subset. An shuffle function shuffles the Train data subset in a random order to generate more randomness in the Training subset.

```r
# Splits data into Train and Test sets
split.data <- function(data,train=TRUE) {

  trainingIndex <- createDataPartition(data$classe, p=0.6, list=FALSE)
  dataTrain <- data[trainingIndex,]
  dataTest <- data[-trainingIndex,]

  if(train == TRUE) return (dataTrain)
  else return (dataTest)
}

# Shuffles data set
shuffle.data <- function(data){
  rand <- sample(nrow(data))
  shuffled <- data[rand,]

  return (shuffled)
}

# Preprocess Orignial Data set
pmlData  <- prepare.data(pmlData)
# Split pre-processed data set and shuffle Train subset
pmlDataTrain <- shuffle.data(split.data(pmlData,TRUE))
# Split pre-processed data set and get the Test subset
pmlDataTest <- split.data(pmlData,FALSE)
```

Classifier Model Comparison and Selection

Based on the problem definition, the categorical outcome variable and provided training dataset, multi-classifier algorithms would be best suited for solving these kind of classification problems. To make things interesting a bit, the data was fitted and performance compared for KNN (K Nearest Neighbour) and Random Forest algorithms.

In order to better estimate the accuracy of the KNN model, a Repeated Cross-Fold validation resampling techniue was used. While for Random forests, there is no need for cross-validation to get an unbiased estimate of the test set error since it is estimated internally during the run.

```r
# KNN Classifier

# Repeated Cross Fold validation for KNN
ctrl <- trainControl(method="repeatedcv",number = 10,repeats=3)

# Build Model
knnFit <- train(classe ~ ., data = pmlDataTrain,method = "knn",trControl = ctrl, preProcess = c("cent
er","scale"))
# Predict outcome for Test data subset
predictionsKNN <- predict(knnFit,pmlDataTest[,-53])
# Compare predicted values with Observed or actual outcome values
confusionMatrixKNN <- confusionMatrix(pmlDataTest$classe,predictionsKNN)

# Random Forest Classifier

# Build Model
randomForestFit <- randomForest(classe ~ ., data = pmlDataTrain, importance=TRUE)
# Predict outcome for Test data subset
predictionsRF <- predict(randomForestFit,pmlDataTest[,-53])
#Compare predicted values with Observed or actual outcome values
confusionMatrixRF <- confusionMatrix(pmlDataTest$classe,predictionsRF)
```

Now compare the two models on their Accuracy, Kappa and Mis-classification rates

```r
# Compare the Accuracy of the Models

# Define a function for reporting Metrics
reportMetrics = function(confusionMatrix,modelName){

  missClassRate <- 1 - (sum(diag(confusionMatrix$table))/sum(confusionMatrix$table))
  print (paste("misClassificate Rate (%) for ", modelName," : ", round(missClassRate*100,3),"%"))
}

# Report Overall Metrics for kNN
confusionMatrixKNN$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      9.713e-01      9.637e-01      9.674e-01      9.749e-01      2.873e-01
## AccuracyPValue  McnemarPValue
##      0.000e+00      6.292e-06
```

```r
reportMetrics(confusionMatrixKNN,"kNN")
```

```
## [1] "misClassificate Rate (%) for  kNN  :  2.868 %"
```

```
# Report Overall Metrics for Random Forest
confusionMatrixRF$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##         0.9976         0.9969         0.9962         0.9985         0.2852
## AccuracyPValue  McnemarPValue
##         0.0000            NaN
```

```
reportMetrics(confusionMatrixRF,"Random Forest")
```

```
## [1] "misClassificate Rate (%) for  Random Forest  :  0.242 %"
```

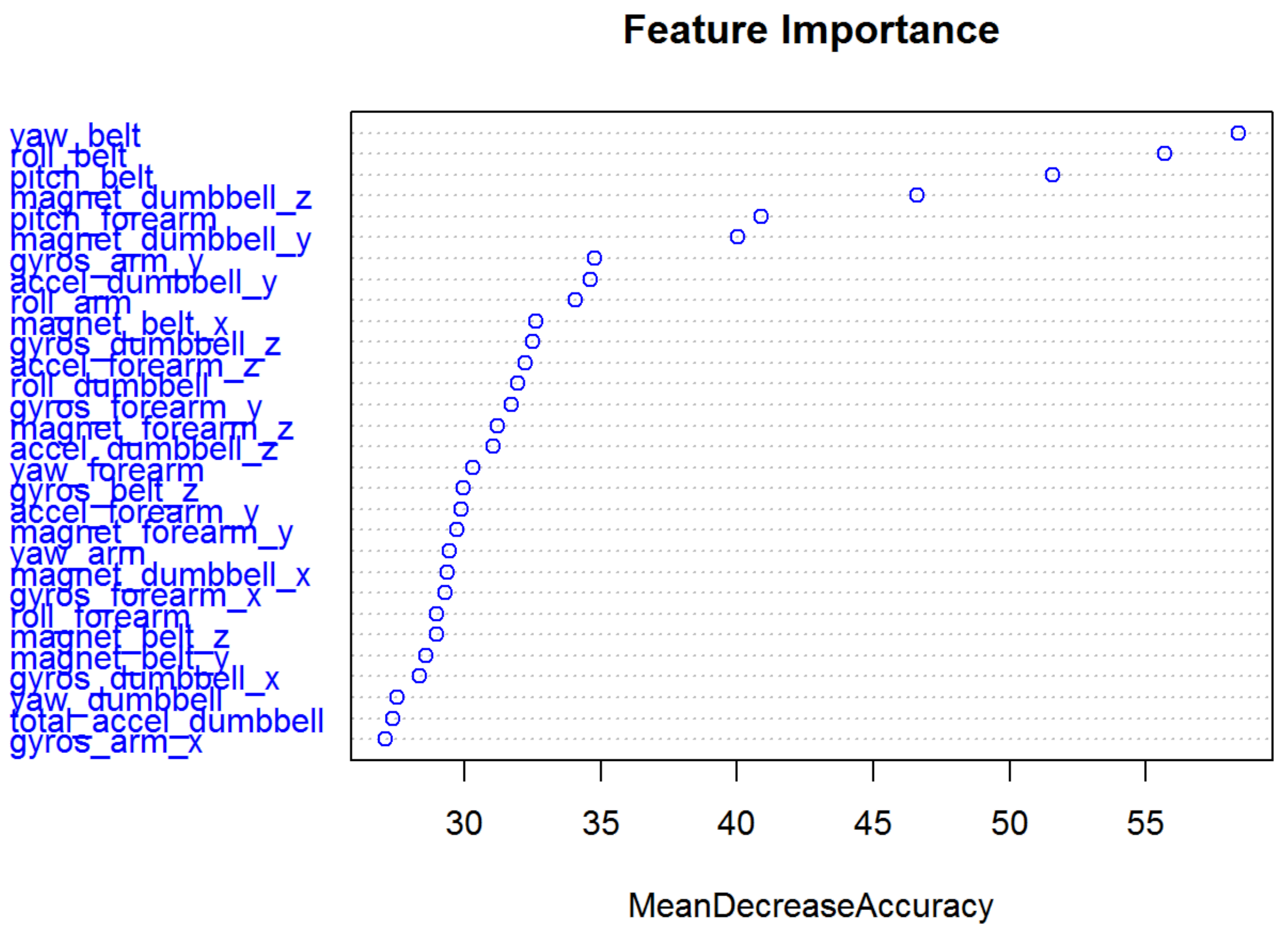
Based on the metric comparison, Random Forest has an order of magnitude lower Misclassification rate and a higher prediction accuracy and will be used for predicting the outcome variable for the out-of-sample or validation dataset.

Random Forest Classifier Model Tuning

Although the Random Forest implementation performed with relatively high accuracy, there are some performance tweaks that can be made for getting better accuracy.

A feature importance plot below indicates that additional features can be removed from the model to improve the accuracy of the model (However, in the final model all 52 predictor variables were retained).

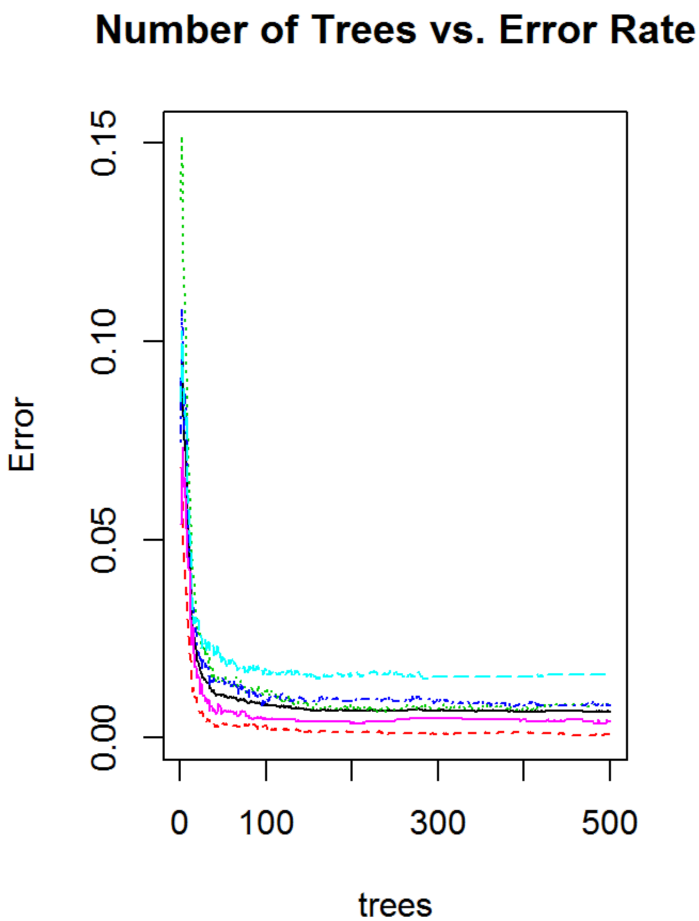
```
varImpPlot(randomForestFit,type=1,main = "Feature Importance",col="blue")
```

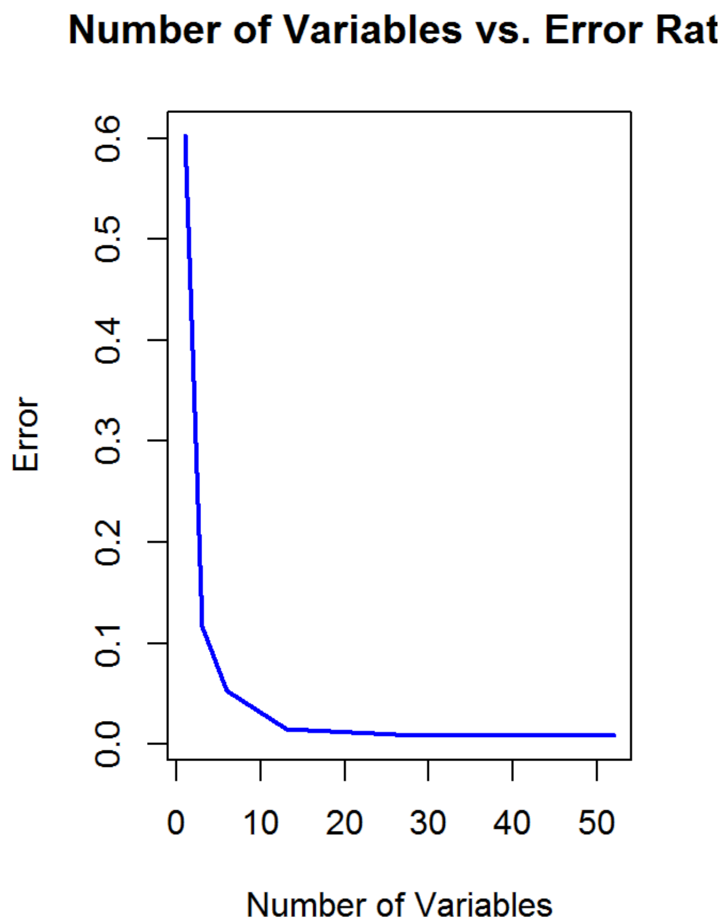


The following Error Rate plots against the Number of Trees in the ensemble and Number of variables indicate that the number of trees could be lowered (ntree =300) and further supports the hypothesis that the number of features can be reduced without any discernible increase in the Error rate.

```
# Function for generating error plots
genRandomForestPlots <- function(data,outcomeVarIdx,model){
  par(mfrow=c(1,2))
  plot(model,main="Number of Trees vs. Error Rate")
  pred <- rfcv(trainx=data[,-outcomeVarIdx],trainy=data[,outcomeVarIdx],data=data,cv.fold = 5)
  plot(pred$n.var,pred$error.cv,type="l",lwd="2",col="blue",
       xlab = "Number of Variables", ylab="Error",main = " Number of Variables vs. Error Rate")
  }

genRandomForestPlots(pmlDataTrain,53,randomForestFit)
```





Final Model and Results

The complete metrics for the final model are below

```
# Report the entire Confusion Matrix Table and OOB for Random Forest model

randomForestFit
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = pmlDataTrain, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.66%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3345    3    0    0    0   0.0008961
## B   14 2261    4    0    0   0.0078982
## C    0   16 2037    1    0   0.0082765
## D    0    0   28 1899    3   0.0160622
## E    0    0    1    8 2156   0.0041570
```

```
confusionMatrixRF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    6 1510    2    0    0
##          C    0    1 1367    0    0
##          D    0    0    6 1279    1
##          E    0    0    0    3 1439
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.996, 0.999)
##     No Information Rate : 0.285
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.997
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.997    0.999    0.994    0.998    0.999
## Specificity             1.000    0.999    1.000    0.999    1.000
## Pos Pred Value          1.000    0.995    0.999    0.995    0.998
## Neg Pred Value          0.999    1.000    0.999    1.000    1.000
## Prevalence              0.285    0.193    0.175    0.163    0.184
## Detection Rate          0.284    0.192    0.174    0.163    0.183
## Detection Prevalence    0.284    0.193    0.174    0.164    0.184
## Balanced Accuracy       0.999    0.999    0.997    0.998    0.999
```

Application of the final model against the Validation Data Set making sure to apply the same preprocessing rules to the validation data set

```
# Apply same pre-processing rules applied to Training data set
validationData <- prepare.data(validationData)



# Predict Classe variable using the Random Forest model
result <- predict(randomForestFit,validationData)

# Display prediction results

result
```

```
##  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
```

Random Forest classifier algorithm was chosen for classifiying the classe outcome variable that gave high level of prediction accuracy for the validation data set. The predicted values matched all observed values for the validation dataset.