# Practical Machine Learning (Final Project)

Varun Ginde

21/06/2020

## 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, my goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants.

I aim to use a model to identify (as accurately as possible) the manner of exercise (the classe) using the given training data. Below are some of the libraries I will use to complete my project.

```
library(caret)
```

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

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

```
library(e1071)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(ggthemes)
```

Firstly, I read in the training and testing data which I have downloaded. The training data has close to 20000 observations and 160 columns (including classe). the first task will be to get rid of the useless ones. In this case, the first 7 columns are useless, so we get rid of them first.

```
train_data = read.csv('pml-training.csv')
final_test_data = read.csv('pml-testing.csv')

train_data = select(train_data, -(1:7))
final_test_data = select(final_test_data, -(1:7))
```

## Missing Values

Several columns are useless because they have several missing values, so I have written code below to get rid of these columns. This is also done with both the datasets.

```
trainData<- train_data[, colSums(is.na(train_data)) == 0]
finalData <- final_test_data[, colSums(is.na(final_test_data)) == 0]
```

## The Training Set

I won't touch the actual dataset. Instead, I will split the training dataset into 2 subsets, sub-training and sub-testing. I prefer a split ratio of 75%.

```
set.seed(101)
intrain = createDataPartition(trainData$classe, p = 0.75, list = 0)
sub_train = trainData[intrain,]
sub_test = trainData[-intrain,]
```

I also want to remove variables with zero variance (or close to zero variance). For this purpose, I have used the nzv function below.

```
NZV = nzv(trainData)
trainData = trainData[,-NZV]

sub_test = sub_test[,-NZV]
sub_train = sub_train[,-NZV]
```

# Model1 - Support Vector Machines

The 1st model I will use is SVM. I find it extremely useful as far as accuracy is concerned. I have trained this model with the sub-training data and predicted the sub-test values using this trained model. The confusion matrix will show how accurate the model has been.

I have used a radial kernel with gamma value 0.1 (most of the times, 0.1 tends to give the best model, for a given cost).

```
modelsvm = svm(classe~., data = sub_train, kernel = 'radial', cost = 10, gamma = 0.1)

pred_svm = predict(modelsvm, sub_test)

confusionMatrix(pred_svm, sub_test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1385    5    0    0    0
##          B    2  940    1    0    0
##          C    1    3  848    3    0
##          D    0    0    6  799    0
##          E    7    1    0    2  901
##
## Overall Statistics
##
##                Accuracy : 0.9937
##                  95% CI : (0.991, 0.9957)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.992
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9928   0.9905   0.9918   0.9938   1.0000
## Specificity            0.9986   0.9992   0.9983   0.9985   0.9975
## Pos Pred Value         0.9964   0.9968   0.9918   0.9925   0.9890
## Neg Pred Value         0.9972   0.9977   0.9983   0.9988   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2824   0.1917   0.1729   0.1629   0.1837
## Detection Prevalence   0.2834   0.1923   0.1743   0.1642   0.1858
## Balanced Accuracy      0.9957   0.9949   0.9950   0.9962   0.9988
```

The accuracy is astoundingly good, which can be a bit suspicious at times. Which is why I will try out another model to see if we get similar results or whether SVM is actually the best method.

# Model 2 - Random Forests

Another beautiful model for classification, random forests are a great method to use for such problems. I have written code to train this model using sub-training data and predict the values for the sub-test data. Below is a plot, which shows the error rate of the random forests model varying with number of trees used.

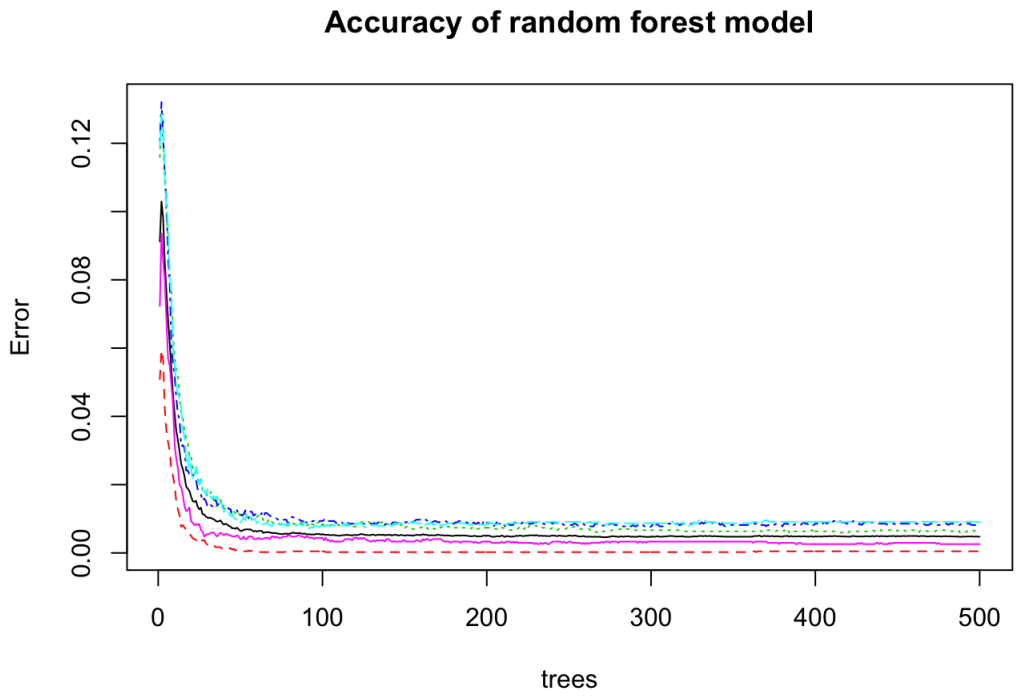
```
modelrf = randomForest(classe~., data = sub_train)

pred_rf = predict(modelrf, sub_test)

confusionMatrix(pred_rf, sub_test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1393    5    0    0    0
##          B    1  943    3    0    0
##          C    0    1  850    7    1
##          D    0    0    2  796    0
##          E    1    0    0    1  900
##
## Overall Statistics
##
##                Accuracy : 0.9955
##                  95% CI : (0.9932, 0.9972)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9943
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9986   0.9937   0.9942   0.9900   0.9989
## Specificity            0.9986   0.9990   0.9978   0.9995   0.9995
## Pos Pred Value         0.9964   0.9958   0.9895   0.9975   0.9978
## Neg Pred Value         0.9994   0.9985   0.9988   0.9981   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2841   0.1923   0.1733   0.1623   0.1835
## Detection Prevalence   0.2851   0.1931   0.1752   0.1627   0.1839
## Balanced Accuracy      0.9986   0.9963   0.9960   0.9948   0.9992
```

```
plot(modelrf, main = 'Accuracy of random forest model')
```




# Model 3 - Combining the 2 models

Since the above model too provides near-perfect results (and slightly better than SVM), I have combined the two models and run an SVM on these models to get the average of the models.

```
combo = data.frame(pred_svm, pred_rf, sub_test$classe)

combined_model = svm(sub_test.classe~. , data = combo)

pred_combined = predict(combined_model, sub_test)

confusionMatrix(pred_combined, sub_test$classe)
```

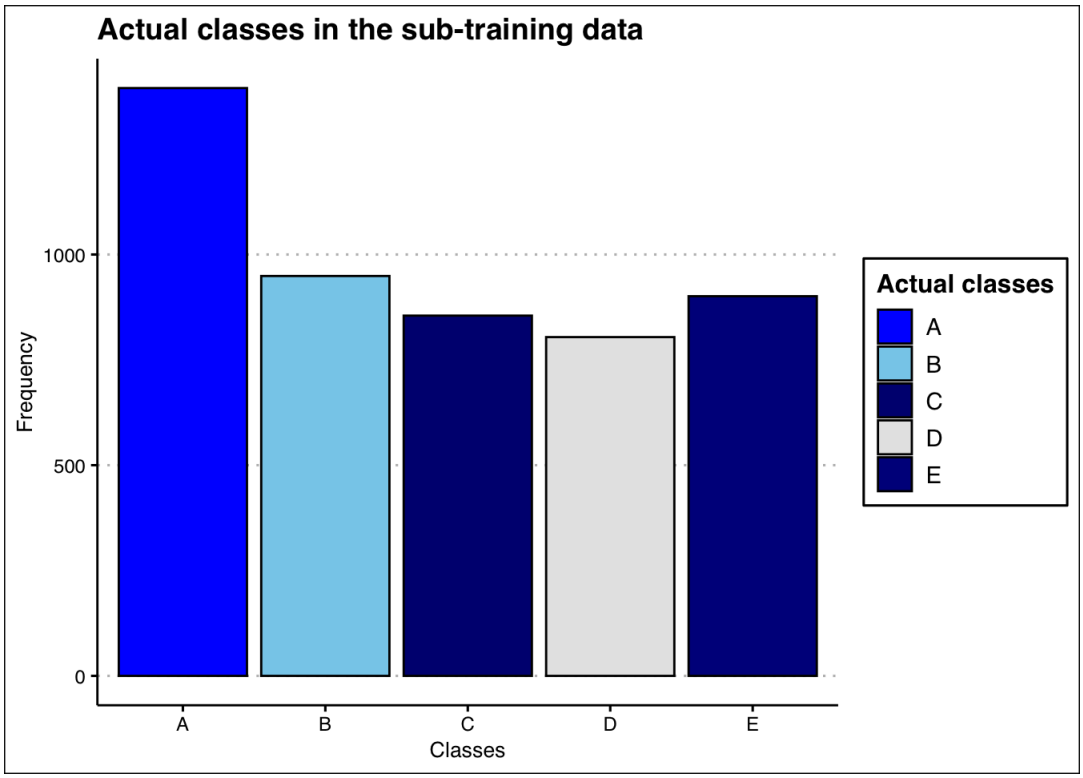
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1391    0    0    1    1
##          B    2  948    4    0    0
##          C    1    1  850    8    0
##          D    0    0    1  794    0
##          E    1    0    0    1  900
##
## Overall Statistics
##
##                Accuracy : 0.9957
##                  95% CI : (0.9935, 0.9973)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9946
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9971   0.9989   0.9942   0.9876   0.9989
## Specificity            0.9994   0.9985   0.9975   0.9998   0.9995
## Pos Pred Value         0.9986   0.9937   0.9884   0.9987   0.9978
## Neg Pred Value         0.9989   0.9997   0.9988   0.9976   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2836   0.1933   0.1733   0.1619   0.1835
## Detection Prevalence   0.2841   0.1945   0.1754   0.1621   0.1839
## Balanced Accuracy      0.9983   0.9987   0.9958   0.9937   0.9992
```

This model too has near-perfect accuracy. The results are exceptionally good and it is a good indicator of how easy it is to classify this particular dataset.

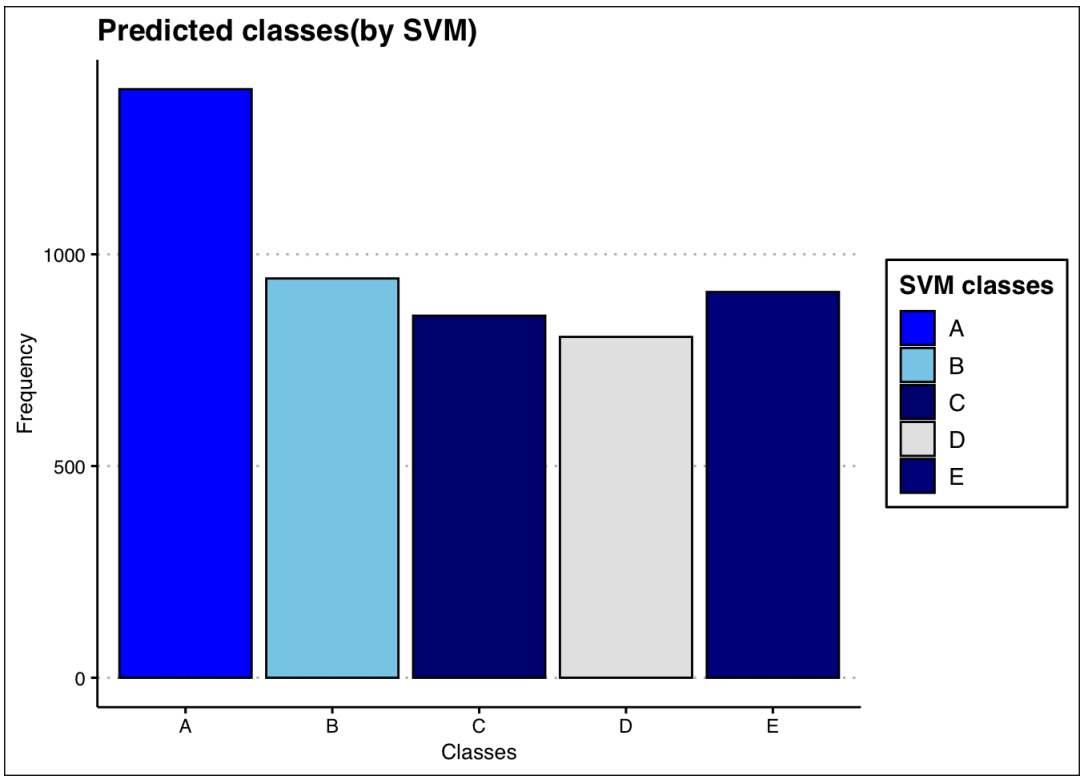
Below, I have plotted 2 barplots, one for the actual sub-testing classe and one for the predictions provided by model 1. This is a visual confirmation of how similar the predictions are, with the 2 graphs looking exactly the same (although there are minute differences). I haven't plotted the model 2 predictions since that will give us 3 graphs that look exactly the same.

Thus I conclude that it is slightly better to use a combination of the two models to predict the values of classe for the variables.

```
ggplot(data = combo, aes(x = sub_test.classe)) + geom_bar(aes(fill = sub_test.classe), color = 'black') + gg
title(label = 'Actual classes in the sub-training data') + theme(plot.title = element_text(hjust = 0.5, face
= 'bold')) + theme_clean() + xlab('Classes') + ylab('Frequency') + scale_fill_manual('Actual classes', value
s = c('blue','sky blue','navy blue','grey90','dark blue'))
```




```
ggplot(data = combo, aes(x = pred_svm)) + geom_bar(aes(fill = pred_svm), color = 'black') + ggtitle(label = 'Predicted classes(by SVM)') + theme(plot.title = element_text(hjust = 0.5, face = 'bold')) + theme_clean() + xlab('Classes') + ylab('Frequency') + scale_fill_manual('SVM classes', values = c('blue','sky blue','navy blue','grey90','dark blue'))
```




```
Results = predict(combined_model, finalData)
```