Introduction

Introduction

With the increase in the usage of activity trackers we have a lot of data available with us in regards to fitness and activities. In this paper I am going to look one such huge data generated from the usage of few people. We will look at all the parameters in the dataset, cleanse them, use only the required parameters and build a model based on that to predict the activity based on the measurements. We will then use the same model to predict what the activity on a test dataset.

Exploring the data

We will load the required library and read the complete dataset into R

```
library(caret)
library(e1071)
set.seed(1234)
training_initial <- read.csv("pml-training.csv")
```

When looking at the file we can see that there are many columns with missing values. Let us see how many columns have more than 90% of missing values. These predictors with so much missing values will not add any value to the model. Instead it may affect the accuracy of the models. We will remove them from our training dataset

```
missing_cols <- colnames(training_initial)[(colSums(is.na(training_initial))/nrow(training_i
trainingv1 <- training_initial[ , !(names(training_initial) %in% missing_cols)]
```

Again there are many columns with blank values. We will do the same what we did for the missing values to these blank values.

```
blank_cols <- colnames(trainingv1)[(colSums(trainingv1 == "")/nrow(trainingv1)) > 0.9]
trainingv2 <- trainingv1[ , !(names(trainingv1) %in% blank_cols)]
table(trainingv2$new_window)
```

```
##
##    no   yes
## 19216   406
```

Now looking at the remining predictors the name and timestamp will not add anything to model for predicting what activity they are performing. Also the new_window columns are having very less variability so we will remove those columns and form our final training dataset using which we can predict our model.

```
trainingv3 <- trainingv2[, -c(1:7)]
head(trainingv3)
```

```
##   roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x gyros_belt_y
## 1      1.41       8.07    -94.4                3         0.00         0.00
## 2      1.41       8.07    -94.4                3         0.02         0.00
## 3      1.42       8.07    -94.4                3         0.00         0.00
## 4      1.48       8.05    -94.4                3         0.02         0.00
## 5      1.48       8.07    -94.4                3         0.02         0.02
## 6      1.45       8.06    -94.4                3         0.02         0.00
##   gyros_belt_z accel_belt_x accel_belt_y accel_belt_z magnet_belt_x
## 1        -0.02          -21            4           22            -3
## 2        -0.02          -22            4           22            -7
## 3        -0.02          -20            5           23            -2
## 4        -0.03          -22            3           21            -6
## 5        -0.02          -21            2           24            -6
## 6        -0.02          -21            4           21             0
##   magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm total_accel_arm
## 1           599          -313     -128      22.5    -161              34
## 2           608          -311     -128      22.5    -161              34
## 3           600          -305     -128      22.5    -161              34
## 4           604          -310     -128      22.1    -161              34
## 5           600          -302     -128      22.1    -161              34
## 6           603          -312     -128      22.0    -161              34
##   gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x accel_arm_y accel_arm_z
## 1        0.00        0.00       -0.02        -288         109        -123
## 2        0.02       -0.02       -0.02        -290         110        -125
## 3        0.02       -0.02       -0.02        -289         110        -126
## 4        0.02       -0.03        0.02        -289         111        -123
## 5        0.00       -0.03        0.00        -289         111        -123
## 6        0.02       -0.03        0.00        -289         111        -122
##   magnet_arm_x magnet_arm_y magnet_arm_z roll_dumbbell pitch_dumbbell
## 1         -368          337          516      13.05217      -70.49400
## 2         -369          337          513      13.13074      -70.63751
## 3         -368          344          513      12.85075      -70.27812
```

```
## 4         -372          344          512      13.43120      -70.39379
## 5         -374          337          506      13.37872      -70.42856
## 6         -369          342          513      13.38246      -70.81759
##   yaw_dumbbell total_accel_dumbbell gyros_dumbbell_x gyros_dumbbell_y
## 1    -84.87394                  37                0            -0.02
## 2    -84.71065                  37                0            -0.02
## 3    -85.14078                  37                0            -0.02
## 4    -84.87363                  37                0            -0.02
## 5    -84.85306                  37                0            -0.02
## 6    -84.46500                  37                0            -0.02
##   gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y accel_dumbbell_z
## 1             0.00             -234               47             -271
## 2             0.00             -233               47             -269
## 3             0.00             -232               46             -270
## 4            -0.02             -232               48             -269
## 5             0.00             -233               48             -270
## 6             0.00             -234               48             -269
##   magnet_dumbbell_x magnet_dumbbell_y magnet_dumbbell_z roll_forearm
## 1              -559               293               -65         28.4
## 2              -555               296               -64         28.3
## 3              -561               298               -63         28.3
## 4              -552               303               -60         28.1
## 5              -554               292               -68         28.0
## 6              -558               294               -66         27.9
##   pitch_forearm yaw_forearm total_accel_forearm gyros_forearm_x
## 1         -63.9       -153                 36            0.03
## 2         -63.9       -153                 36            0.02
## 3         -63.9       -152                 36            0.03
## 4         -63.9       -152                 36            0.02
## 5         -63.9       -152                 36            0.02
## 6         -63.9       -152                 36            0.02
##   gyros_forearm_y gyros_forearm_z accel_forearm_x accel_forearm_y
## 1            0.00           -0.02             192             203
## 2            0.00           -0.02             192             203
## 3           -0.02            0.00             196             204
## 4           -0.02            0.00             189             206
## 5            0.00           -0.02             189             206
## 6           -0.02           -0.03             193             203
##   accel_forearm_z magnet_forearm_x magnet_forearm_y magnet_forearm_z
## 1            -215              -17              654              476
## 2            -216              -18              661              473
## 3            -213              -18              658              469
## 4            -214              -16              658              469
## 5            -214              -17              655              473
## 6            -215               -9              660              478
##   classe
```

```
## 1      A
## 2      A
## 3      A
## 4      A
## 5      A
## 6      A
```

```
training <- trainingv3
```

Model Selection

Now we have got all the necessary and only required data for our analysis we will split the model into a training and validation dataset. Since we have close to 20K observations we can go for a 50:50 model for training and validation.

```
intrain <- createDataPartition(y=training$classe,
                               p=0.5, list=FALSE)
newtrainDF <- training[intrain,]
validDF <- training[-intrain,]
newvalidDF <- validDF[,!(names(validDF) %in% "classe")]
```

Now we get into what algorithm we can use to build our model. Since the output is a factor variable it and the predictors are all numeric variable it will be best to use a tree type of model. The decision tree may be used for simple models with less predictors. Random forest will be the best suited algorithm for this kind of data that we want to predict. Let us build a model and see how it works on our validation data

```
RFfit <- train(classe ~ ., method = "rf", data = newtrainDF)
testmodel <- predict(RFfit,newvalidDF)
confusionMatrix(testmodel,validDF$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2788   26    0    0    0
##          B    0 1860   15    1    3
##          C    2   10 1691   24    4
##          D    0    2    5 1581    5
##          E    0    0    0    2 1791
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9899
##                  95% CI : (0.9877, 0.9918)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9872
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9800   0.9883   0.9832   0.9933
## Specificity            0.9963   0.9976   0.9951   0.9985   0.9998
## Pos Pred Value         0.9908   0.9899   0.9769   0.9925   0.9989
## Neg Pred Value         0.9997   0.9952   0.9975   0.9967   0.9985
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1896   0.1724   0.1612   0.1826
## Detection Prevalence   0.2869   0.1915   0.1765   0.1624   0.1828
## Balanced Accuracy      0.9978   0.9888   0.9917   0.9909   0.9965
```

The accuracy of this model is 99% on the validation dataset which is very good for a predictive model. Let us also look at other models and see how do they perform and see if there are any other better models that we can create.

```
dectreefit <- train(classe ~ ., method = "rpart", data = newtrainDF)
testdectree <- predict(dectreefit,newvalidDF)
confusionMatrix(testdectree,validDF$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2536  790  779  712  254
##          B   52  635   60  299  251
##          C  195  473  872  597  497
##          D    0    0    0    0    0
##          E    7    0    0    0  801
##
## Overall Statistics
##
##                Accuracy : 0.4938
##                  95% CI : (0.4838, 0.5037)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.3387
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9090  0.33456  0.50964   0.0000  0.44426
## Specificity            0.6389  0.91633  0.78244   1.0000  0.99913
## Pos Pred Value         0.5001  0.48959  0.33106      NaN  0.99134
## Neg Pred Value         0.9464  0.85164  0.88308   0.8361  0.88869
## Prevalence             0.2844  0.19348  0.17441   0.1639  0.18379
## Detection Rate         0.2585  0.06473  0.08889   0.0000  0.08165
## Detection Prevalence   0.5169  0.13221  0.26850   0.0000  0.08236
## Balanced Accuracy      0.7739  0.62545  0.64604   0.5000  0.72169
```

```
svmfit <- svm(classe ~ ., data = newtrainDF)
testsvm <- predict(svmfit,newvalidDF)
confusionMatrix(testsvm,validDF$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2764  183    4    2    5
##          B    7 1652   76    6   25
##          C   16   60 1609  171   83
##          D    0    2   21 1426   56
##          E    3    1    1    3 1634
##
## Overall Statistics
##
##                Accuracy : 0.9261
##                  95% CI : (0.9207, 0.9312)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9063
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9907   0.8704   0.9404   0.8868   0.9063
## Specificity            0.9724   0.9856   0.9593   0.9904   0.9990
```

```
## Pos Pred Value         0.9344   0.9354   0.8298   0.9475   0.9951
## Neg Pred Value         0.9962   0.9694   0.9870   0.9781   0.9793
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2818   0.1684   0.1640   0.1454   0.1666
## Detection Prevalence   0.3015   0.1800   0.1977   0.1534   0.1674
## Balanced Accuracy      0.9815   0.9280   0.9498   0.9386   0.9526
```

The decision tree model is not that good and it has only 54% accuracy. The SVM model is also very good model but none of these are better than Random forest so we can conclude that it is the best model for the given set of predictors

Applying the model to the test dataset

Now lets read the test dataset and the read only the columns that are required for us to predict the activity

```
testing_initial <- read.csv("pml-testing.csv")
testing <- testing_initial[, colnames(newvalidDF)]
finaltest <- predict(RFfit,testing)
finaltest
```

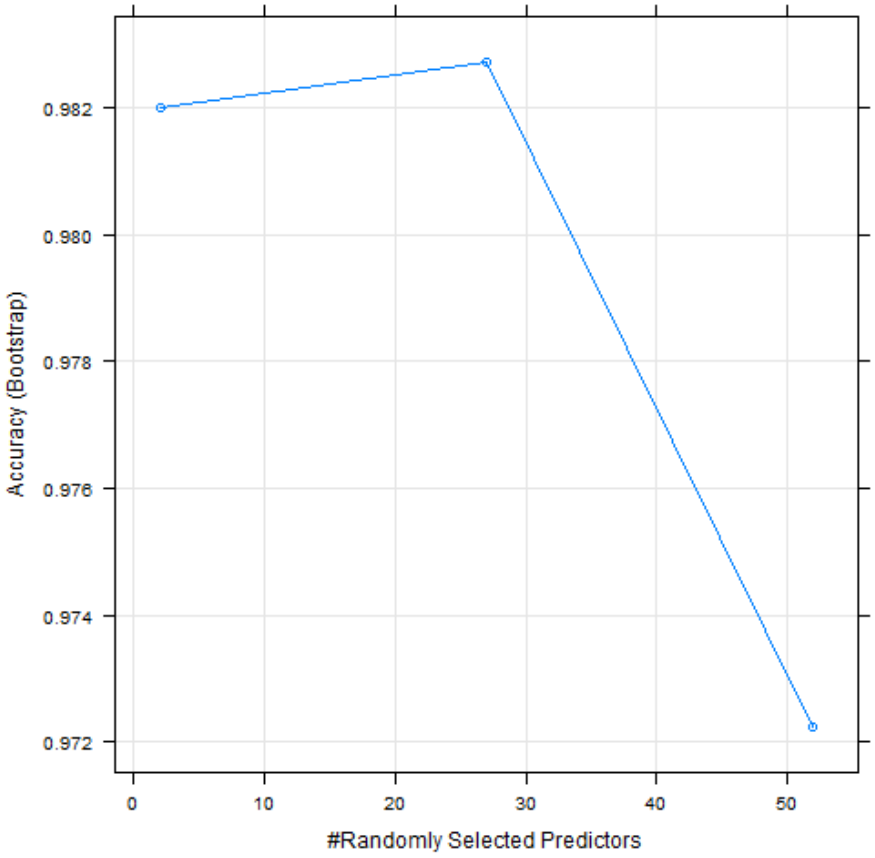
```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

So we are able to predict the activity of the test dataset as well. Since the model is 98% accurate these predictions will be almost accurate.

Accuracy plot of the model

The model still shows that if we can reduce the predictors the accuracy will still improve. We need to do more analysis and study the data to remove the predictors which are similar in nature

```
par(mfrow = c(2,2))
plot(RFfit)
```



Conclusion

Hence by cleaning the data and using all the necessary predictors we are able to build an accurate model. This model can now be used to predict the activity based on the parameters available.