Prediction Assignment Writeup

Amandeep Singh

October 24, 2017

1. Objective

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, we will 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. The goal of your project is to predict the manner in which they did the exercise. This is the “classe” variable in the training set.

2. Data Analysis and Processing

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

```
library(FactoMineR)
library(factoextra)
```

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

```
##
## Attaching package: 'ggplot2'
```

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

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at http
s://goo.gl/13EFCZ
```

```
library(ggplot2)
library(lattice)
library(caret)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Load Data

```
training_data  <- read.csv("pml-training.csv")
testing_data <- read.csv("pml-testing.csv")
```

The *training_data* will be used for building the model whereas *testing_data* will just be used to answer Quiz questions.

Explore and Clean Data

```
dim(training_data)
```

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

The dataset is huge with 160 columns. Let us first try to eliminate all the columns and rows with NA. This will help reducing dimensionality.

```
training_data <- training_data[, colSums(is.na(training_data)) == 0]
testing_data <- testing_data[, colSums(is.na(testing_data)) == 0]
dim(training_data)
```

```
## [1] 19622    93
```

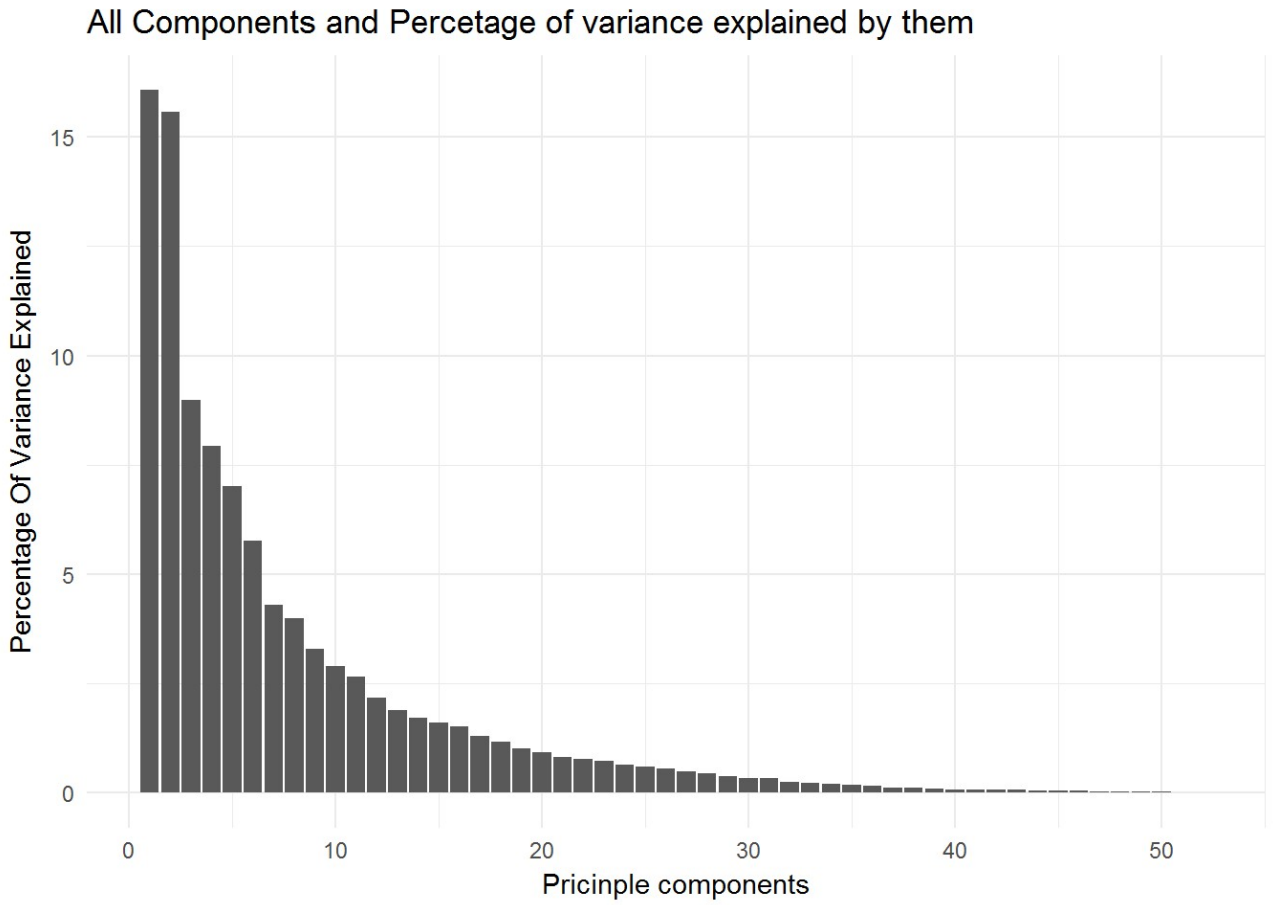
Further reduce the number of columns by removing all the columns related to timestamps, window. and also remove all the non-numerical columns.

```
#remove timestamps and window related columns
training_data <- training_data[, !grepl("^X|timestamp|window", names(training_d
ata))]
testing_data <- testing_data[, !grepl("^X|timestamp|window", names(testing_dat
a))]
training_classe <- training_data$classe
testing_classe <- testing_data$classe
#remove all non numeric fields
training_data <- training_data[, sapply(training_data, is.numeric)]
testing_data <- testing_data[, sapply(testing_data, is.numeric)]
training_data$classe <- training_classe
testing_data$classe <- testing_classe
dim(training_data)
```

```
## [1] 19622    53
```

The dimentions are still high, lets perfrom PCA to findout the pricipal components.

```
res.pca <- PCA(training_data[,-ncol(training_data)], graph = FALSE)
results <- as.data.frame(res.pca$eig)
names(results) <- c("eigenvalue","percentage_of_variance","cumulative_percentag
e_of_variance" )
ggplot(results, aes(x=1:52, y=percentage_of_variance)) + geom_bar(stat="identit
y")+theme_minimal()+
  xlab("Pricinple components") + ylab("Percentage Of Variance Explained") +ggti
tle("All Components and Percetage of variance explained by them")
```



From the plot above we see that about first 10 pricinple component explain 75% of the variations. Although we can work with just first 10 components. But for the sake of more accuracy I have chosen to work with first 36 components, which explain about 99% of variablity.

```
pre_processed <- preProcess(training_data[,-ncol(training_data)], method="pc
a", thresh= 0.99)
training_data <- predict(pre_processed, training_data[,-ncol(training_data)])
training_data$classe <- training_classe

# create training set with 70% of data
set.seed(123)
inTrain <- createDataPartition(y=training_data$classe, p=0.7, list=FALSE)
training_set <- training_data[inTrain,]
testing_set <- training_data[-inTrain,]
dim(training_set)
```

```
## [1] 13737    37
```

```
dim(testing_set)
```

```
## [1] 5885   37
```

Now we have the data completely pre-processed with just 36 factors. We have also divided the final *training_data* into *training_set* and *testing_set* for our model developement process.

3. Predictive Modelling

We will explore three different Machine Learning algorithms and choose the best model to answer quiz questions: 1. Random Forests 2. SVM (Support Vector Machines) 3. Decision Trees

1. Random Forest

```
library(e1071)
set.seed(123)
control_rf <- trainControl(method="cv", number=3, verboseIter=FALSE,allowParall
el=TRUE)
model_rf<-train(classe ~ ., data=training_set, trControl = control_rf , method
="rf", prox=TRUE)
model_rf$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.21%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3895    3    2    3    3  0.00281618
## B   57 2572   26    0    3  0.03235515
## C    4   31 2343   14    4  0.02212020
## D    3    2   97 2144    6  0.04795737
## E    2    9   22   12 2480  0.01782178
```

Prediction on testing set

```
prediction_rf <- predict(model_rf, testing_set, type="raw")
conf_matrix_rf <- confusionMatrix(testing_set$classe,prediction_rf)
# Confusion matrix
conf_matrix_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1667    2    1    1    3
##          B   21 1110    8    0    0
##          C    0   21  993   12    0
##          D    1    1   41  917    4
##          E    0    4    5    3 1070
##
## Overall Statistics
##
##                Accuracy : 0.9782
##                  95% CI : (0.9742, 0.9818)
##     No Information Rate : 0.287
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9725
##  Mcnemar's Test P-Value : 1.392e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9870   0.9754   0.9475   0.9829   0.9935
## Specificity            0.9983   0.9939   0.9932   0.9905   0.9975
## Pos Pred Value         0.9958   0.9745   0.9678   0.9512   0.9889
## Neg Pred Value         0.9948   0.9941   0.9887   0.9967   0.9985
## Prevalence             0.2870   0.1934   0.1781   0.1585   0.1830
## Detection Rate         0.2833   0.1886   0.1687   0.1558   0.1818
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9927   0.9846   0.9703   0.9867   0.9955
```

Our Random Forests model showed accuracy of 97.8%, which is pretty good.

2. SVM (Support Vector Machines)

```
set.seed(123)
control_svm <- trainControl(method="cv", number=3, verboseIter=FALSE,allowParal
lel=TRUE)
model_svm <- train(classe ~ ., data=training_set, trControl = control_svm , mod
el="svm", prox=TRUE)
model_svm$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, proximity = TRUE,      model
= "svm")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.21%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3895    3    2    3    3  0.00281618
## B   57 2572   26    0    3  0.03235515
## C    4   31 2343   14    4  0.02212020
## D    3    2   97 2144    6  0.04795737
## E    2    9   22   12 2480  0.01782178
```

Prediction on testing set

```
prediction_svm <- predict(model_svm, testing_set, type="raw")
conf_matrix_svm <- confusionMatrix(testing_set$classe,prediction_svm)
# Confusion matrix
conf_matrix_svm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1667    2    1    1    3
##          B   21 1110    8    0    0
##          C    0   21  993   12    0
##          D    1    1   41  917    4
##          E    0    4    5    3 1070
##
## Overall Statistics
##
##                Accuracy : 0.9782
##                  95% CI : (0.9742, 0.9818)
##     No Information Rate : 0.287
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9725
##  Mcnemar's Test P-Value : 1.392e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9870   0.9754   0.9475   0.9829   0.9935
## Specificity            0.9983   0.9939   0.9932   0.9905   0.9975
## Pos Pred Value         0.9958   0.9745   0.9678   0.9512   0.9889
## Neg Pred Value         0.9948   0.9941   0.9887   0.9967   0.9985
## Prevalence             0.2870   0.1934   0.1781   0.1585   0.1830
## Detection Rate         0.2833   0.1886   0.1687   0.1558   0.1818
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9927   0.9846   0.9703   0.9867   0.9955
```

our SVM model also showed accuracy of 97.8%, which is again pretty good.

3. Decision Trees

```
library(rpart)
set.seed(123)
model_dt <- rpart(classe ~ ., data=training_set, method="class")
model_dt
```

```
## n= 13737
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 13737 9831 A (0.28 0.19 0.17 0.16 0.18)
##     2) PC14>=-0.9174821 12371 8510 A (0.31 0.19 0.19 0.17 0.14)
##       4) PC8< 1.276288 10066 6503 A (0.35 0.17 0.22 0.11 0.14)
##         8) PC12>=-1.375251 9454 5901 A (0.38 0.18 0.23 0.11 0.11)
##          16) PC21< 0.1088582 5187 2617 A (0.5 0.17 0.16 0.099 0.082) *
##          17) PC21>=0.1088582 4267 2889 C (0.23 0.19 0.32 0.12 0.14)
##            34) PC35>=0.2996162 599  200 A (0.67 0.12 0.095 0.018 0.1) *
##            35) PC35< 0.2996162 3668 2347 C (0.16 0.2 0.36 0.13 0.14)
##              70) PC36< -0.08553442 1314  818 B (0.1 0.38 0.23 0.13 0.16) *
##              71) PC36>=-0.08553442 2354 1338 C (0.19 0.11 0.43 0.14 0.13)
##               142) PC22>=0.4702432 376  162 A (0.57 0.098 0.16 0.077 0.096)
*
##               143) PC22< 0.4702432 1978 1022 C (0.12 0.11 0.48 0.15 0.14) *
##         9) PC12< -1.375251 612  206 E (0.016 0.13 0.016 0.18 0.66) *
##       5) PC8>=1.276288 2305 1355 D (0.13 0.24 0.063 0.41 0.16)
##        10) PC36< 0.006015347 1150  711 B (0.22 0.38 0.03 0.16 0.2) *
##        11) PC36>=0.006015347 1155  394 D (0.036 0.099 0.096 0.66 0.11) *
##     3) PC14< -0.9174821 1366  614 E (0.033 0.26 0.031 0.13 0.55) *
```

Prediction on testing set

```
prediction_dt <- predict(model_dt, testing_set, type="class")
conf_matrix_dt <- confusionMatrix(testing_set$classe,prediction_dt)
# Confusion matrix
conf_matrix_dt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1332  177  121   19   25
##          B  437  372   94   46  190
##          C  410  133  412   53   18
##          D  231  172  136  295  130
##          E  211  194  130   68  479
##
## Overall Statistics
##
##                Accuracy : 0.4911
##                  95% CI : (0.4782, 0.5039)
##     No Information Rate : 0.4454
##     P-Value [Acc > NIR] : 1.078e-12
##
##                   Kappa : 0.3414
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.5082  0.35496  0.46137  0.61331  0.56888
## Specificity            0.8952  0.84143  0.87700  0.87620  0.88043
## Pos Pred Value         0.7957  0.32660  0.40156  0.30602  0.44270
## Neg Pred Value         0.6939  0.85756  0.90101  0.96220  0.92442
## Prevalence             0.4454  0.17808  0.15174  0.08173  0.14308
## Detection Rate         0.2263  0.06321  0.07001  0.05013  0.08139
## Detection Prevalence   0.2845  0.19354  0.17434  0.16381  0.18386
## Balanced Accuracy      0.7017  0.59820  0.66918  0.74475  0.72466
```

We can see that Decision Tree algorithm perfom pretty bad on our dataset, with only 49.11% accuracy.

4. Cross Validation used

The cross validation technique we used n our models is called k-fold cross validation - The k-fold cross validation method involves splitting the dataset into k-subsets. For each subset is held out while the model is trained on all other subsets. It is a robust method for estimating accuracy, and the size of k and tune the amount of bias in the estimate, with popular values set to 3, 5, 7 and 10.

For Ramdom Forests and SVM models we used K = 3.

4. Conclusion

We saw that in our case Random Forests and SVM models performed with same level of accuracy. We can choose either of these models, but on my PC i found Random Forests took lesser time then SVM. Given the same level of accuracy, i am picking up Random Forests Model to answer Quiz questions.

5. Answering Quiz Questions

```
testing_data_processed <- predict(pre_processed, testing_data[,-ncol(testing_da
ta)])
predition_results <- predict(model_rf, testing_data_processed)
predition_results
```

```
##  [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
```