Practical Machine Learning - Final Project Coursera

Jim Saing

May 29, 2020

Project Introduction

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, your goal will be to 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. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

Data

The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv
(https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv
(https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har). If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

Goal

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. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Summary

- Data Processing
- Model Building
- Conclusion

Data Processing

Libraries

```
library(caret);library(ggplot2);library(randomForest);library(corrplot);library(rattle);library(gbm);library(e1071);library(dplyr)
```

Loading Data

"stringsAsFactors = TRUE" is used to get a factor type instead of strings. It will be usefull later to get the Confusion Matrices.

```
train_test <-  read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"), stringsAsFactors = TRUE)
validation <- read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"), stringsAsFactors = TRUE)
```

Cleaning the training data

```
#Describing Data
str(train_test %>% select(classe, everything()));dim(train_test)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ classe                  : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2
2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1
323084232 1323084232 1323084232 1323084232 1323084232 ...
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390
484323 484434 ...
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9
9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -9
4.4 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1
1 ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0
...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
```

```
##  $ accel_belt_z           : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x          : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y          : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z          : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm               : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm              : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm                : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm        : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ var_accel_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x            : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y            : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03
...
##  $ gyros_arm_z            : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x            : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y            : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z            : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x           : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y           : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z           : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm      : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ kurtosis_picth_arm     : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ kurtosis_yaw_arm       : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ skewness_roll_arm      : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ skewness_pitch_arm     : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ skewness_yaw_arm       : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ max_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell          : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell         : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell           : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1 1 1 1 1 1 1
1 1 1 ...
##  $ kurtosis_picth_dumbbell: Factor w/ 401 levels "","-0.0163","-0.0233",..: 1 1 1 1 1 1 1
1 1 1 ...
##  $ kurtosis_yaw_dumbbell  : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_dumbbell : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1 1 1 1 1 1 1
```

```
1 1 1 ...
##  $ skewness_pitch_dumbbell : Factor w/ 402 levels "","-0.0053","-0.0084",..: 1 1 1 1 1 1 1
1 1 1 ...
##  $ skewness_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1
...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1
...
##   [list output truncated]
```

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

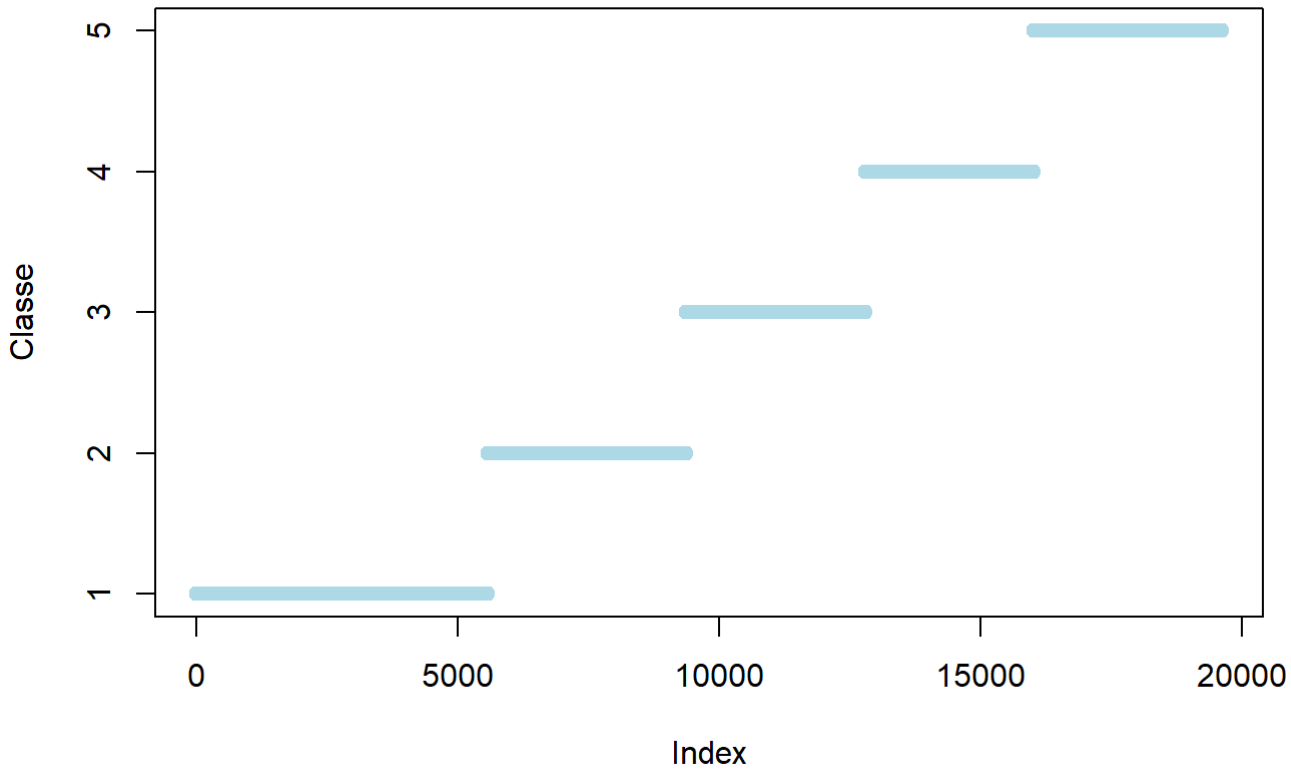
```
class(train_test$classe);summary(train_test %>% select(classe))
```

```
## [1] "factor"
```

```
##  classe
##  A:5580
##  B:3797
##  C:3422
##  D:3216
##  E:3607
```

We are going to predict the variable ***classe*** that is composed of 5 possible outputs A, B, C, D and E as *factor* type.

```
par(mfrow=c(1,1))
plot(train_test$X,train_test$classe,xlab = "Index",ylab = "Classe",col="light blue")
```



By looking at the data, it can be noticed that the first seven columns are not related to the ***classe*** variable that is to be predicted, and there are columns containing NAs. More specifically, with the example of the index, the ***classe*** variable is ordered according to the index. By adding the index to our models, it can bias them as ***X*** should not have links with ***classe*** except by the way the data are ordered.

Let's remove these first seven columns and all the column containing NAs.

```
train_test <- train_test[,-c(1:7)]
train_test <- train_test[ , colSums(is.na(train_test)) == 0]
```

Then we remove the near zero variance variables to clean even more the data.

```
nzcv <- nearZeroVar(train_test)
train_test <- train_test[,-nzcv]
```

Data Partitioning

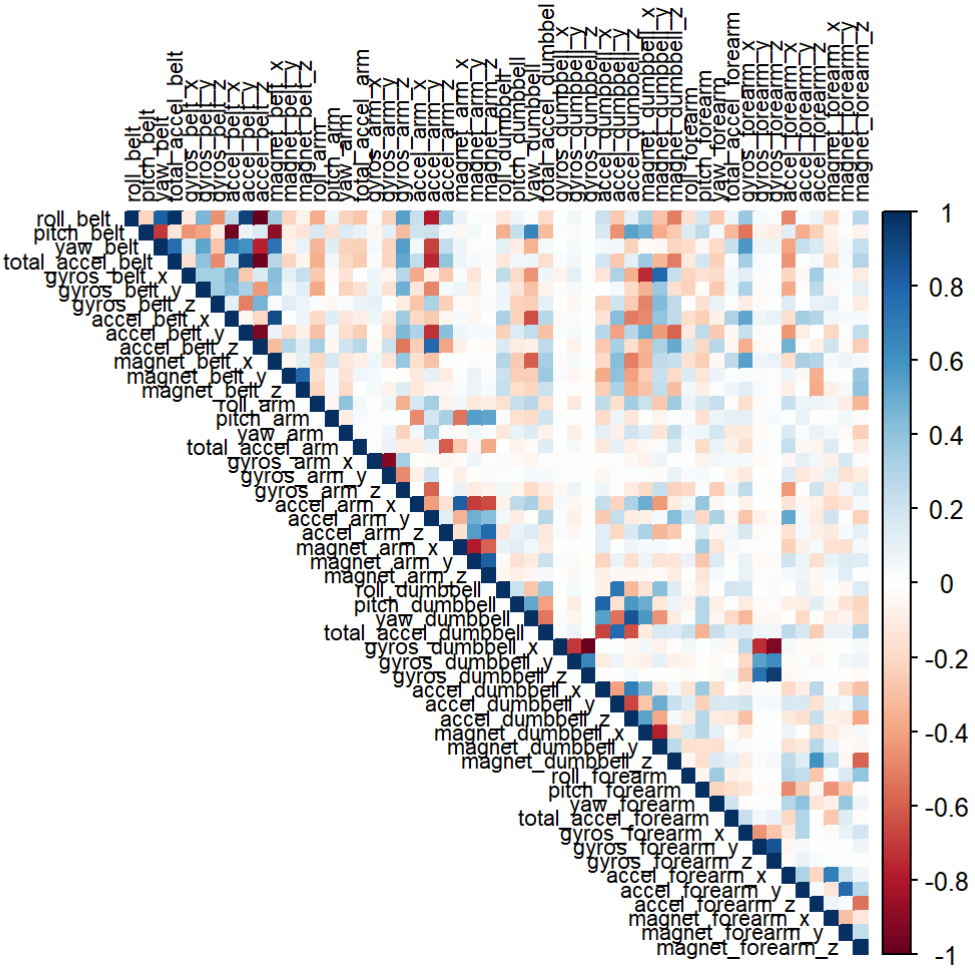
We decide to split it to sub-training and sub-testing set, with 85% going to the first one as the initial training set is very large.

```
set.seed(33244)
inTrain <- createDataPartition(y=train_test$classe, p=0.85, list = FALSE)
training <- train_test[inTrain,]
testing <- train_test[-inTrain,]

dim(training)
```

```
## [1] 16680    53
```

After cleaning, we are down to 53 variables. Let's have a look on their correlation

```
par(mfrow=c(1,1))
correl_plot <- corrplot(cor(training[,-53]),method="color",type="upper",tl.col="black",tl.cex
=0.7)
```




```
M <- abs(cor(training[,-53]))
diag(M) <- 0
# Highly correlated variables
which(M>0.9, arr.ind=T)
```

```
##                  row col
## total_accel_belt   4   1
## accel_belt_y       9   1
## accel_belt_z      10   1
## accel_belt_x       8   2
## roll_belt          1   4
## accel_belt_y       9   4
## accel_belt_z      10   4
## pitch_belt         2   8
## roll_belt          1   9
## total_accel_belt   4   9
## accel_belt_z      10   9
## roll_belt          1  10
## total_accel_belt   4  10
## accel_belt_y       9  10
## gyros_arm_y       19  18
## gyros_arm_x       18  19
## gyros_dumbbell_z  33  31
## gyros_forearm_z   46  31
## gyros_dumbbell_x  31  33
## gyros_forearm_z   46  33
## gyros_dumbbell_x  31  46
## gyros_dumbbell_z  33  46
```

Model Building

For this project, by looking at the structure of the data and the result that we are looking for, we will at first use models based on trees, and then we will try model based predictions.
Five models will be used:

1. Decision Tree
2. Random Forest
3. Gradient Boosting with trees
4. Linear Discriminant Analysis
5. Naive Bayes

Furthermore, for each model, we will use the raw data and the principal components data to see if the Principal Component Analysis generates a good prediction.

The algorithms will be only explained for the tree model as for the other models, it is the same principle by using the same **caret** functions.

Preprocessing with PCA

As we have lots of variables, we try to use PCA to reduce dimensionality and by the same time the complexity.

```
preProc <- preProcess(training[,-53],method = "pca",pcaComp=20)
trainPC <- predict(preProc,training[,-53])
testPC <- predict(preProc,testing[,-53])

sum(var(trainPC))/52
```

```
## [1] 0.9201397
```

```
## [1] "We choose 20 Principal components as they explain  92.014 % of the variance"
```

Training Control Parameters - Cross validations

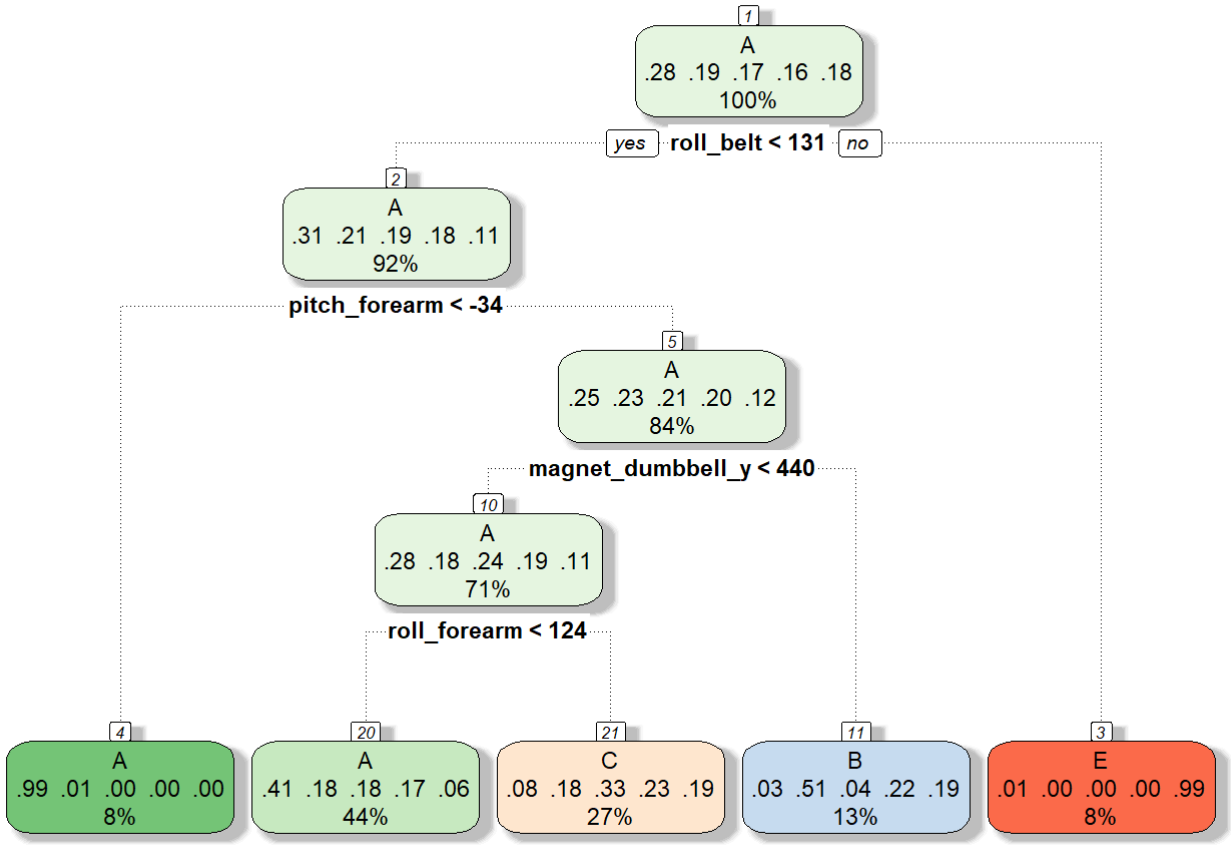
We only use the 5-folds CV to reduce the overfitting in the different model while not adding too much complexity in our models (especially for the random forest).

```
fitControl <- trainControl(method="cv", number=5)
```

1. Decision Tree

The Decision Tree model iteratively splits variables into groups and evaluates homogeneity within each group

```
#Model Building - Predicting with trees

#Set seed to have the same result each time as it is pseudo-random
set.seed(33244)
#Fit predictive model over different tuning parameters, here in the training data, we fit the
'classe' variable on the 'rpart' method with the 'fitControl' as the train control for the cr
oss validation
fit_tree <- train(classe~., method='rpart',data=training, trControl=fitControl)
#Use the model to predict on the testing set
pred_tree <- predict(fit_tree,newdata=testing)
#Create the confusion matrix to analyze the result on the testing set
cm_tree <- confusionMatrix(pred_tree,testing$classe)

#Plot the tree
par(mfrow=c(1,1))
library(rattle)
fancyRpartPlot(fit_tree$finalModel)
```




```
#With PCA
fit_tree_pc <- train(x=trainPC,y=training$classe, method='rpart',trControl=fitControl)
cm_tree_pc <- confusionMatrix(testing$classe,predict(fit_tree_pc,testPC))
```

2. Random Forest

The Random Forest is more complex than the Decision Tree but based on this one. It bootstraps samples, at each split bootstraps variables, grows multiples trees then votes for the best.

```
#Model Building - Predicting with Random Forest
set.seed(1234)
fit_rf <- train(classe~., method="rf", data=training,trControl= fitControl)
pred_rf <- predict(fit_rf,newdata=testing)
cm_rf <- confusionMatrix(pred_rf,testing$classe)

#With PCA
fit_rf_pc <- train(x=trainPC, y=training$classe, method="rf",trControl=fitControl)
cm_rf_pc <- confusionMatrix(testing$classe, predict(fit_rf_pc,testPC))
```

3. Gradient Boosting with trees

The GBM method takes lots of trees, weights them and creates a classifier that combines them.

```
#Model Building - Gradient Boosting with trees
set.seed(33244)
fit_gbm <- train(classe~., method="gbm", data=training, trControl=fitControl, verbose=FALSE)
pred_gbm <- predict(fit_gbm,newdata=testing)
cm_gbm <- confusionMatrix(pred_gbm,testing$classe)

#With PCA
fit_gbm_pc <- train(x=trainPC, y=training$classe, method="gbm", trControl=fitControl, verbose
=FALSE)
cm_gbm_pc <- confusionMatrix(testing$classe, predict(fit_gbm_pc, testPC))
```

4. Linear Discriminant Analysis

```
#Model building - Linear Discriminant Analysis
set.seed(33244)
fit_lda <- train(classe~., method="lda", data=training, trControl=fitControl)
pred_lda <- predict(fit_lda,newdata=testing)
cm_lda <- confusionMatrix(pred_lda,testing$classe)

#With PCA
fit_lda_pc <- train(x=trainPC, y=training$classe, method="lda",trControl=fitControl)
cm_lda_pc <- confusionMatrix(testing$classe, predict(fit_lda_pc,testPC))
```

5. Naive Bayes

```
set.seed(33244)
fit_nb <- train(classe~., method="nb", data=training, trControl=fitControl)
pred_nb <- predict(fit_nb,newdata=testing)
cm_nb <- confusionMatrix(pred_nb,testing$classe)

fit_nb_pc <- train(x=trainPC, y=training$classe, method="nb",trControl=fitControl)
cm_nb_pc <- confusionMatrix(testing$classe, predict(fit_nb_pc,testPC))
```

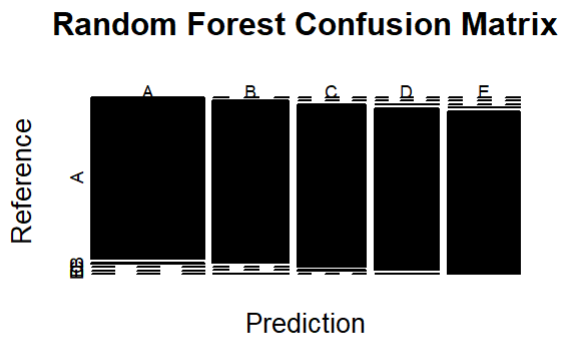
Overall Comparison between the models

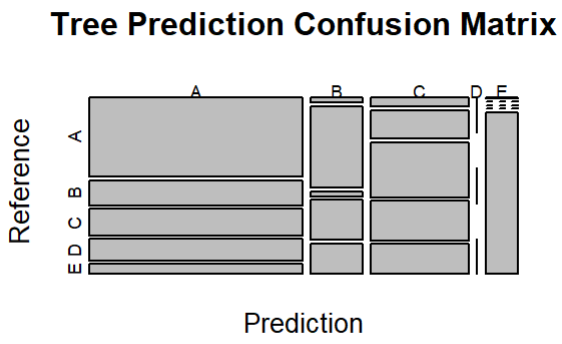
The comparison between the models is made in the testing sample to get the out of sample error.

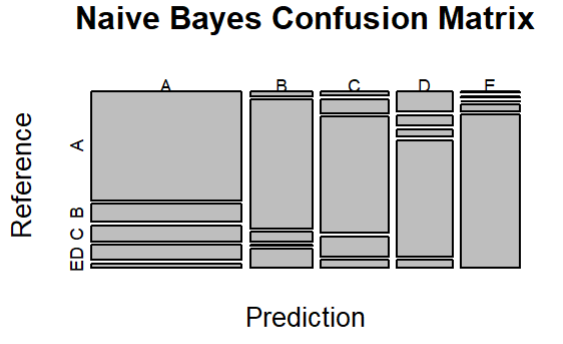
```
overall_table <- rbind(cm_tree$overall, cm_tree_pc$overall, cm_rf$overall, cm_rf_pc$overall,
                       cm_gbm$overall, cm_gbm_pc$overall, cm_lda$overall, cm_lda_pc$overall,
                       cm_nb$overall, cm_nb_pc$overall)
rownames(overall_table) <- c("Tree","Tree PCA","RF","RF PCA","GBM","GBM PCA",
                             "LDA","LDA PCA","NB","NB PCA")

overall_table
```

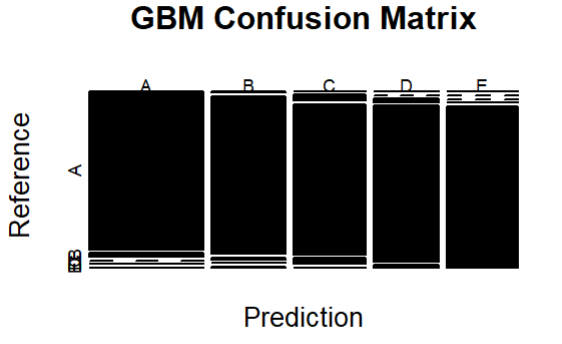
```
##           Accuracy     Kappa AccuracyLower AccuracyUpper AccuracyNull
## Tree     0.4932019 0.3356676     0.4749805     0.5114368    0.2845003
## Tree PCA 0.3810333 0.1673938     0.3634404     0.3988632    0.7535690
## RF       0.9962610 0.9952699     0.9933199     0.9981321    0.2845003
## RF PCA   0.9830048 0.9784964     0.9776547     0.9873602    0.2868797
## GBM      0.9643100 0.9548426     0.9569584     0.9707182    0.2845003
## GBM PCA  0.8130523 0.7632046     0.7984857     0.8269935    0.2987763
## LDA      0.6957852 0.6149481     0.6788017     0.7123784    0.2845003
## LDA PCA  0.4921822 0.3533047     0.4739624     0.5104176    0.3422842
## NB       0.7433719 0.6713072     0.7271820     0.7590762    0.2845003
## NB PCA   0.6094494 0.5050603     0.5915484     0.6271324    0.3052345
##          AccuracyPValue McnemarPValue
## Tree      1.009679e-125           NaN
## Tree PCA   1.000000e+00           NaN
## RF         0.000000e+00           NaN
## RF PCA     0.000000e+00           NaN
## GBM        0.000000e+00  5.930514e-04
## GBM PCA    0.000000e+00  6.250094e-20
## LDA        0.000000e+00  4.259102e-31
## LDA PCA    8.246001e-63  2.673594e-39
## NB         0.000000e+00  1.641261e-60
## NB PCA    2.048762e-253  1.283481e-17
```

```
par(mfrow=c(2,2))
plot(cm_rf$table,cm_rf$byClass,main="Random Forest Confusion Matrix",col="black")
plot(cm_tree$table,cm_tree$byClass,main="Tree Prediction Confusion Matrix")
plot(cm_nb$table,cm_nb$byClass,main="Naive Bayes Confusion Matrix")
plot(cm_gbm$table, cm_gbm$byClass,main="GBM Confusion Matrix", col="black")
```









We can see that the PCA lower the accuracy in each case. It is hard to interpret why PCA reduces the accuracy, though it can be explained by the fact that PCA is linear while it's possible that we have non-linear dependencies.

According to the different charts above, the ranking by Accuracy and Kappa of the different method without PCA is
Random Forest > Boosting with trees > Naive Bayes > LDA > Decision Tree

The poor accuracy of the tree decision can be understood through its tree chart, it has only taken into account very few variables.

Conclusion

Chosen model - Random Forest without PCA

With the overall table, the accuracy of the Random Forest model is the highest, so its out of sample error is the lowest. Even if this model is very long to compute, we choose that one to predict the ***classe*** of the validation set.

Random Forest Out of sample error:

```
## [1] "Out Sample error is  0.3739 %"
```

Variables by Importance in the Random Forest model:

```
varImp(fit_rf)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 52)
##
##                      Overall
## roll_belt            100.000
## pitch_forearm         58.616
## yaw_belt              53.219
## magnet_dumbbell_y     44.011
## pitch_belt            43.715
## magnet_dumbbell_z     43.468
## roll_forearm          43.453
## accel_dumbbell_y      21.503
## accel_forearm_x       17.396
## magnet_dumbbell_x     16.193
## roll_dumbbell         14.884
## magnet_belt_z         14.749
## magnet_forearm_z      14.730
## accel_belt_z          13.753
## total_accel_dumbbell  13.341
## accel_dumbbell_z      12.930
## magnet_belt_y         12.694
## yaw_arm               11.237
## gyros_belt_z          10.339
## magnet_belt_x          9.715
```

Validation Prediction

Now, as the model is selected, we apply it to the validation test to predict to which ***classe*** the 20 test cases belong to.

```
pred_validation <- predict(fit_rf,validation)
pred_validation
```

```
##  [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
```

```
paste("The expected Accuracy is ", round((cm_rf$overall[1])*100,digits = 4),"%")
```

```
## [1] "The expected Accuracy is  99.6261 %"
```