Practical_ML

Touqeer Mulla

2022-11-08

Introduction

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.
Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).
Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience. It was made sure that all participants could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg).
My goal here is to predict the *"class"* with the help of other predictors. This project is a part of Coursera Practical Machine Learning Week 4 - Peer-graded Assignment: Prediction Assignment Writeup.

Data

Load the data

Let's load the data. I have downloaded the data already on my local system. Please download the data from here : Training and Testing. And run this code on the same directory as the data.

```
dfTrain <- read.csv("pml-training.csv", stringsAsFactors = F,na.strings = c("","NA","#DIV/0!"))
dfTest <- read.csv("pml-testing.csv", stringsAsFactors = F,na.strings = c("","NA","#DIV/0!"))
dim(dfTrain); dim(dfTest)
```

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

```
## [1]  20 160
```

Let's create a validation for model tuning:

```
#for reproducability
set.seed(101)
inTrain <- createDataPartition(dfTrain$classe, p = 0.8, list = F)
dfVal <- dfTrain[-inTrain,]
dfTrain <- dfTrain[inTrain,]
dim(dfTrain); dim(dfVal)
```

```
## [1] 15699   160
```

```
## [1] 3923  160
```

Now 3 partition of our data is ready, lets dive into analysis but 1st lets look at the proportion of different "classe":

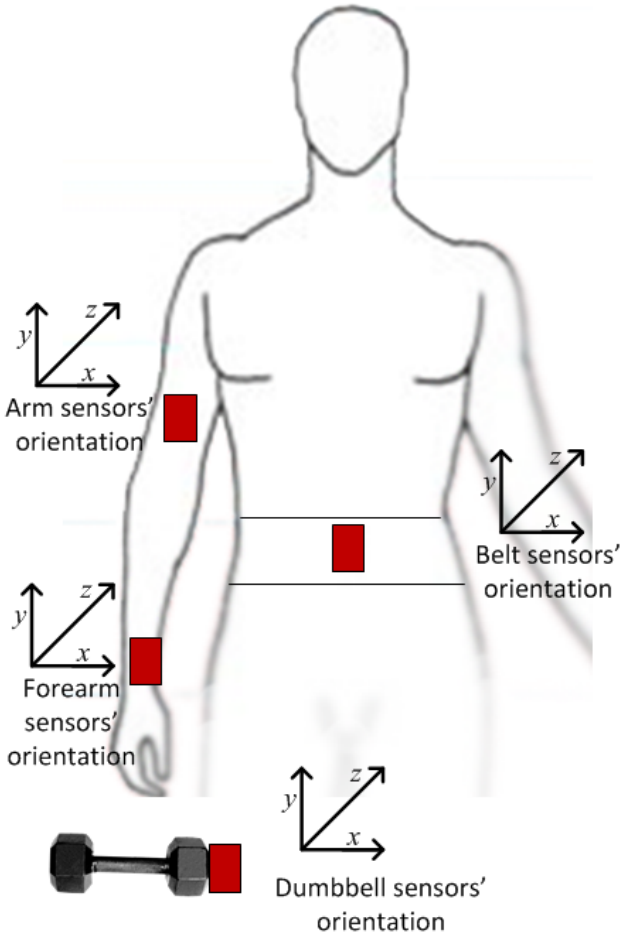
```
table(dfTrain$classe)/nrow(dfTrain)
```

```
##
##         A         B         C         D         E
## 0.2843493 0.1935155 0.1744060 0.1638958 0.1838334
```

From the above it is clear that there are not that much bias in the data in term of different "classe".

Column overview

The data has 160 columns and for training data 15699 rows. Data was collected with the help of 4 sensors, shown in the below diagram (diagram source).



The following symbol " ■ " designates one of the set of sensors described in the text

Few Key points about the columns:

- *"X"* is primary key for the data.
- *"user_name"* is the id of the users. This may help us see interesting patterns for each activity for different users.
- *"classe"* is the target for prediction.
- Column - *3 to 7* is not necessary for this project. (5 features)
- As mentioned above there are 4 different sensors used for data collection. For each sensor there are 38 different features.
- Each sensor("belt","arm","forearm","dumbbell") has raw accelerometer, gyroscope and magnetometer readings for x, y and z axis. (4 sensor * 3 feature * 3 axis = 36 features)
- Each sensor("belt","arm","forearm","dumbbell") has Euler angles (roll, pitch and yaw) feature.(4 sensor * 3 euler angles = 12 features)
- For the Euler angles of each of the four sensors eight features were calculated: mean, variance, standard deviation, max, min, amplitude, kurtosis and skewness. (4 sensor * 3 feature * 8 measures = 96 features)
- For accelerometer we also have "total" and "variance of total" feature for the 4 sensors. But for "belt", "variance of total" is given as "var_total_accel_belt", for the other sensors it is given as ("var_accel_arm","var_accel_dumbbell","var_accel_forearm"). So I am considering the "belt" one as a typo. (4 sensor * 2 feature = 8 features)

- There is another thing to note here. For "belt" Euler angles feature skewness is given as "skewness_roll_belt", "skewness_roll_belt.1" and "skewness_yaw_belt". I am also considering "skewness_roll_belt.1" as a typo and considering it as "skewness_pitch_belt".

Missingness in the data

Let's take a quick look at the missingness of the data. As the no of feature is large, its better to see them by the 4 sensors:

Belt

For Belt sensor:

```
belt_miss <- sapply(select(dfTrain,names(dfTrain)[grepl("_belt",names(dfTrain))]),
                    function(x) sum(is.na(x)))
belt_miss
```

```
##            roll_belt           pitch_belt             yaw_belt
##                    0                    0                    0
##     total_accel_belt   kurtosis_roll_belt  kurtosis_picth_belt
##                    0                15396                15413
##    kurtosis_yaw_belt   skewness_roll_belt skewness_roll_belt.1
##                15699                15395                15413
##    skewness_yaw_belt        max_roll_belt       max_picth_belt
##                15699                15388                15388
##         max_yaw_belt        min_roll_belt       min_pitch_belt
##                15396                15388                15388
##         min_yaw_belt  amplitude_roll_belt amplitude_pitch_belt
##                15396                15388                15388
##   amplitude_yaw_belt var_total_accel_belt        avg_roll_belt
##                15396                15388                15388
##     stddev_roll_belt        var_roll_belt       avg_pitch_belt
##                15388                15388                15388
##    stddev_pitch_belt       var_pitch_belt         avg_yaw_belt
##                15388                15388                15388
##      stddev_yaw_belt         var_yaw_belt         gyros_belt_x
##                15388                15388                    0
##         gyros_belt_y         gyros_belt_z         accel_belt_x
##                    0                    0                    0
##         accel_belt_y         accel_belt_z        magnet_belt_x
##                    0                    0                    0
##        magnet_belt_y        magnet_belt_z
##                    0                    0
```

Arm

For Arm sensor:

```
arm_miss <- sapply(select(dfTrain,names(dfTrain)[grepl("_arm",names(dfTrain))]),
                   function(x) sum(is.na(x)))
arm_miss
```

```
##            roll_arm           pitch_arm             yaw_arm     total_accel_arm
##                   0                   0                   0                   0
##       var_accel_arm        avg_roll_arm     stddev_roll_arm        var_roll_arm
##               15388               15388               15388               15388
##       avg_pitch_arm    stddev_pitch_arm       var_pitch_arm         avg_yaw_arm
##               15388               15388               15388               15388
##      stddev_yaw_arm         var_yaw_arm         gyros_arm_x         gyros_arm_y
##               15388               15388                   0                   0
##         gyros_arm_z         accel_arm_x         accel_arm_y         accel_arm_z
##                   0                   0                   0                   0
##        magnet_arm_x        magnet_arm_y        magnet_arm_z   kurtosis_roll_arm
##                   0                   0                   0               15446
##  kurtosis_picth_arm    kurtosis_yaw_arm   skewness_roll_arm  skewness_pitch_arm
##               15448               15398               15445               15448
##    skewness_yaw_arm        max_roll_arm       max_picth_arm         max_yaw_arm
##               15398               15388               15388               15388
##        min_roll_arm       min_pitch_arm         min_yaw_arm  amplitude_roll_arm
##               15388               15388               15388               15388
## amplitude_pitch_arm   amplitude_yaw_arm
##               15388               15388
```

Forearm

For Forearm sensor:

```
forearm_miss <- sapply(select(dfTrain,
                              names(dfTrain)[grepl("_forearm",names(dfTrain))]),
                       function(x) sum(is.na(x)))
forearm_miss
```

```
##            roll_forearm           pitch_forearm             yaw_forearm
##                       0                       0                       0
##   kurtosis_roll_forearm  kurtosis_picth_forearm    kurtosis_yaw_forearm
##                   15448                   15449                   15699
##   skewness_roll_forearm  skewness_pitch_forearm    skewness_yaw_forearm
##                   15447                   15449                   15699
##        max_roll_forearm       max_picth_forearm         max_yaw_forearm
##                   15388                   15388                   15448
##        min_roll_forearm       min_pitch_forearm         min_yaw_forearm
##                   15388                   15388                   15448
##  amplitude_roll_forearm amplitude_pitch_forearm   amplitude_yaw_forearm
##                   15388                   15388                   15448
##     total_accel_forearm       var_accel_forearm        avg_roll_forearm
##                       0                   15388                   15388
##     stddev_roll_forearm        var_roll_forearm       avg_pitch_forearm
##                   15388                   15388                   15388
##    stddev_pitch_forearm       var_pitch_forearm         avg_yaw_forearm
##                   15388                   15388                   15388
##      stddev_yaw_forearm         var_yaw_forearm         gyros_forearm_x
##                   15388                   15388                       0
##         gyros_forearm_y         gyros_forearm_z         accel_forearm_x
##                       0                       0                       0
##         accel_forearm_y         accel_forearm_z        magnet_forearm_x
```

```
##                        0                        0                        0
##       magnet_forearm_y       magnet_forearm_z
##                        0                        0
```

Dumbbell

For Dumbbell sensor:

```
dumbbell_miss <- sapply(select(dfTrain,
                               names(dfTrain)[grepl("_dumbbell",names(dfTrain))]),
                        function(x) sum(is.na(x)))
dumbbell_miss
```

```
##            roll_dumbbell           pitch_dumbbell             yaw_dumbbell
##                        0                        0                        0
##   kurtosis_roll_dumbbell  kurtosis_picth_dumbbell    kurtosis_yaw_dumbbell
##                    15392                    15390                    15699
##   skewness_roll_dumbbell  skewness_pitch_dumbbell    skewness_yaw_dumbbell
##                    15391                    15389                    15699
##        max_roll_dumbbell       max_picth_dumbbell         max_yaw_dumbbell
##                    15388                    15388                    15392
##        min_roll_dumbbell       min_pitch_dumbbell         min_yaw_dumbbell
##                    15388                    15388                    15392
##  amplitude_roll_dumbbell amplitude_pitch_dumbbell   amplitude_yaw_dumbbell
##                    15388                    15388                    15392
##     total_accel_dumbbell       var_accel_dumbbell        avg_roll_dumbbell
##                        0                    15388                    15388
##     stddev_roll_dumbbell        var_roll_dumbbell       avg_pitch_dumbbell
##                    15388                    15388                    15388
##    stddev_pitch_dumbbell       var_pitch_dumbbell         avg_yaw_dumbbell
##                    15388                    15388                    15388
##      stddev_yaw_dumbbell         var_yaw_dumbbell         gyros_dumbbell_x
##                    15388                    15388                        0
##         gyros_dumbbell_y         gyros_dumbbell_z         accel_dumbbell_x
##                        0                        0                        0
##         accel_dumbbell_y         accel_dumbbell_z        magnet_dumbbell_x
##                        0                        0                        0
##        magnet_dumbbell_y        magnet_dumbbell_z
##                        0                        0
```

So it is very interesting to see that few of the features are over 90% missing, I would drop those columns for further analysis. But the interesting thing is that all of those columns have same no of NA values.

```
column_2drop <- c(names(belt_miss[belt_miss != 0]),
                  names(arm_miss[arm_miss != 0]),
                  names(forearm_miss[forearm_miss != 0]),
                  names(dumbbell_miss[dumbbell_miss != 0]))
length(column_2drop)
```

```
## [1] 100
```

So we can drop 100 column as they are mostly missing. After we drop these column there will be 52 predictors left.

Analysis

Now lets get into analysis, first let's look at the correlation among the predictors.

```
#dropping the cols
dfAnalize <- tbl_df(dfTrain %>%
                      select(-column_2drop,
                             -c(X,user_name, raw_timestamp_part_1,
                                raw_timestamp_part_2, cvtd_timestamp,
                                new_window,num_window)))
```

```
## Warning: `tbl_df()` was deprecated in dplyr 1.0.0.
## i Please use `tibble::as_tibble()` instead.
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(column_2drop)
##
##   # Now:
##   data %>% select(all_of(column_2drop))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
```

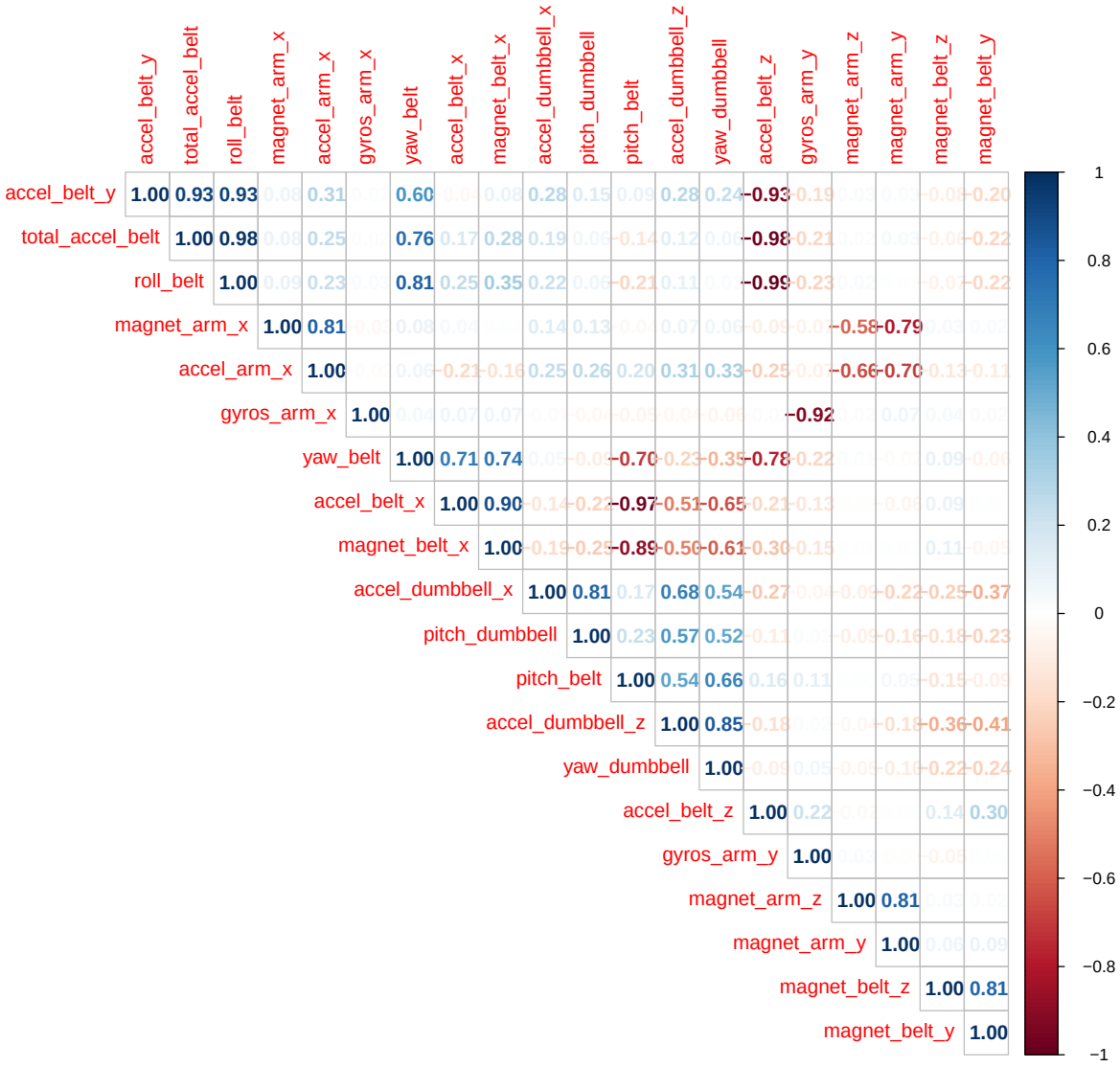
```
dfAnalize$classe <- as.factor(dfAnalize$classe)
dfAnalize[,1:52] <- lapply(dfAnalize[,1:52],as.numeric)
dim(dfAnalize)
```

```
## [1] 15699    53
```

Correlation among predictors

```
corr_col <- cor(select(dfAnalize, -classe))
diag(corr_col) <- 0
corr_col <- which(abs(corr_col)>0.8,arr.ind = T)
corr_col <- unique(row.names(corr_col))
corrplot(cor(select(dfAnalize,corr_col)),
         type="upper", order="hclust",method = "number")
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(corr_col)
##
##   # Now:
##   data %>% select(all_of(corr_col))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
```



Here I have subsetted the data to show only the columns for which absolute correlation is higher than 0.8 with at least one other column. From Correlation plot it is clear that there is lot of columns that are highly correlated. That might be an issue when we will be in modeling phase. Either we can drop those columns or we can perform PCA(Principal Components Analysis). One important thing to note from this graph is that high correlation is only seen between the same sensor i.e. “belt”,“arm”,“forearm” and “dumbbell”.

Correlation with the target

As the target is a categorical variable, we cannot check correlation with the other variables directly. But we can use **correlationfunnel::correlate** to see the correlation with each level of”classe” and other features. Lets go by them one by one.

```
# binarizing data
#correlationfunnel website: https://business-science.github.io/correlationfunnel/
corr_funl_df <- dfAnalize %>% binarize(n_bins = 4, thresh_infreq = 0.01)
```

classe___A

```
corr_a <- corr_funl_df %>% correlate(target = classe__A)
corr_a %>% plot_correlation_funnel(interactive = T,limits = c(-0.5,0.5))
```

For *classe___A* it seems that the "Arm and Forearm" sensors are more important.

- "accel_arm_x" is correlated with "magnet_arm_x", so wont consider.
- "gyros_arm_y" is correlated with "gyros_arm_x", so wont consider.
- So top 5 significant features for "classe___A" are - (magnet_arm_x, pitch_forearm , magnet_dumbbell_y, roll_forearm, gyros_dumbbell_y)

classe___B

```
corr_b <- corr_funl_df %>% correlate(target = classe__B)
corr_b %>% plot_correlation_funnel(interactive = T,limits = c(-0.5,0.5))
```

For *classe___B* it seems that the "Dumbbell and Belt" sensors are more important.

- So top 5 significant features for "classe___A" are - (magnet_dumbbell_y, magnet_dumbbell_x , roll_dumbbell , magnet_belt_y , accel_dumbbell_x)

classe___C

```
corr_c <- corr_funl_df %>% correlate(target = classe__C)
corr_c %>% plot_correlation_funnel(interactive = T,limits = c(-0.5,0.5))
```

For *classe___C* it seems that the "Dumbbell" sensors are more important.

- So top 5 significant features for "classe___A" are - (magnet_dumbbell_y, roll_dumbbell , accel_dumbbell_y , magnet_dumbbell_x, magnet_dumbbell_z)

classe___D

```
corr_d <- corr_funl_df %>% correlate(target = classe__D)
corr_d %>% plot_correlation_funnel(interactive = T,limits = c(-0.5,0.5))
```

For *classe___D* it seems that the "Forearm, Arm and Dumbbell" sensors are more important.

- So top 5 significant features for "classe___A" are - (pitch_forearm , magnet_arm_y , magnet_forearm_x, accel_dumbbell_y, accel_forearm_x)

classe___E

```
corr_e <- corr_funl_df %>% correlate(target = classe__E)
corr_e %>% plot_correlation_funnel(interactive = T,limits = c(-0.5,0.5))
```

For *classe___E* it seems that the "Belt" sensors are more important.

- "total_accel_belt" is correlated with "roll_belt", so wont consider.
- "yaw_belt" is correlated with "roll_belt", so wont consider.
- "accel_belt_z" is correlated with "roll_belt", so wont consider.
- So top 5 significant features for "classe___A" are - (magnet_belt_y , magnet_belt_z , roll_belt, gyros_belt_z , magnet_dumbbell_y)

Let's make some plots

This document is already too long coursera assignment, so for this section I'll work on top 5 features for each class selected in the last section. So lets select only those columns.

```
#subseting dfAnalize
col_a <- c("magnet_arm_x", "pitch_forearm" , "magnet_dumbbell_y",
           "roll_forearm", "gyros_dumbbell_y")
col_b <- c("magnet_dumbbell_y", "magnet_dumbbell_x" , "roll_dumbbell" ,
           "magnet_belt_y" , "accel_dumbbell_x" )
col_c <- c("magnet_dumbbell_y", "roll_dumbbell" , "accel_dumbbell_y" ,
           "magnet_dumbbell_x", "magnet_dumbbell_z")
col_d <- c("pitch_forearm" , "magnet_arm_y" , "magnet_forearm_x",
           "accel_dumbbell_y", "accel_forearm_x")
col_e <- c("magnet_belt_y" , "magnet_belt_z" , "roll_belt",
           "gyros_belt_z" , "magnet_dumbbell_y")
final_cols <- character()
for(c in c(col_a,col_b,col_c,col_d,col_e)){
  final_cols <- union(final_cols, c)
}
dfAnalize2 <- dfAnalize %>% select(final_cols, classe)
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(final_cols)
##
##   # Now:
##   data %>% select(all_of(final_cols))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
```

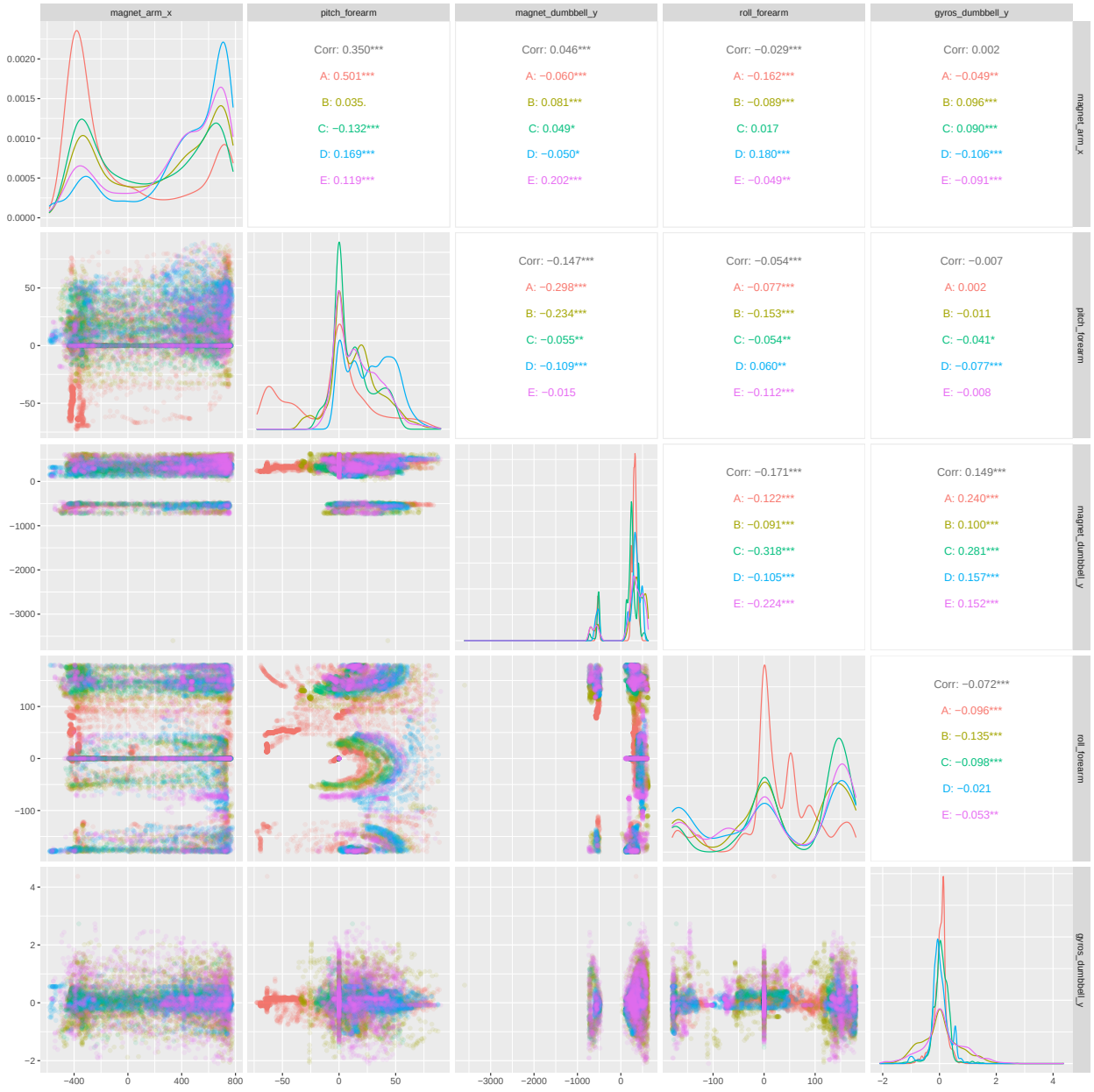
```
data.frame("arm" = sum(grepl("_arm",final_cols)),
           "forearm" = sum(grepl("_forearm",final_cols)),
           "belt" = sum(grepl("_belt",final_cols)),
           "dumbbell" = sum(grepl("_dumbbell",final_cols)))
```

```
##   arm forearm belt dumbbell
## 1   2       4    4        7
```

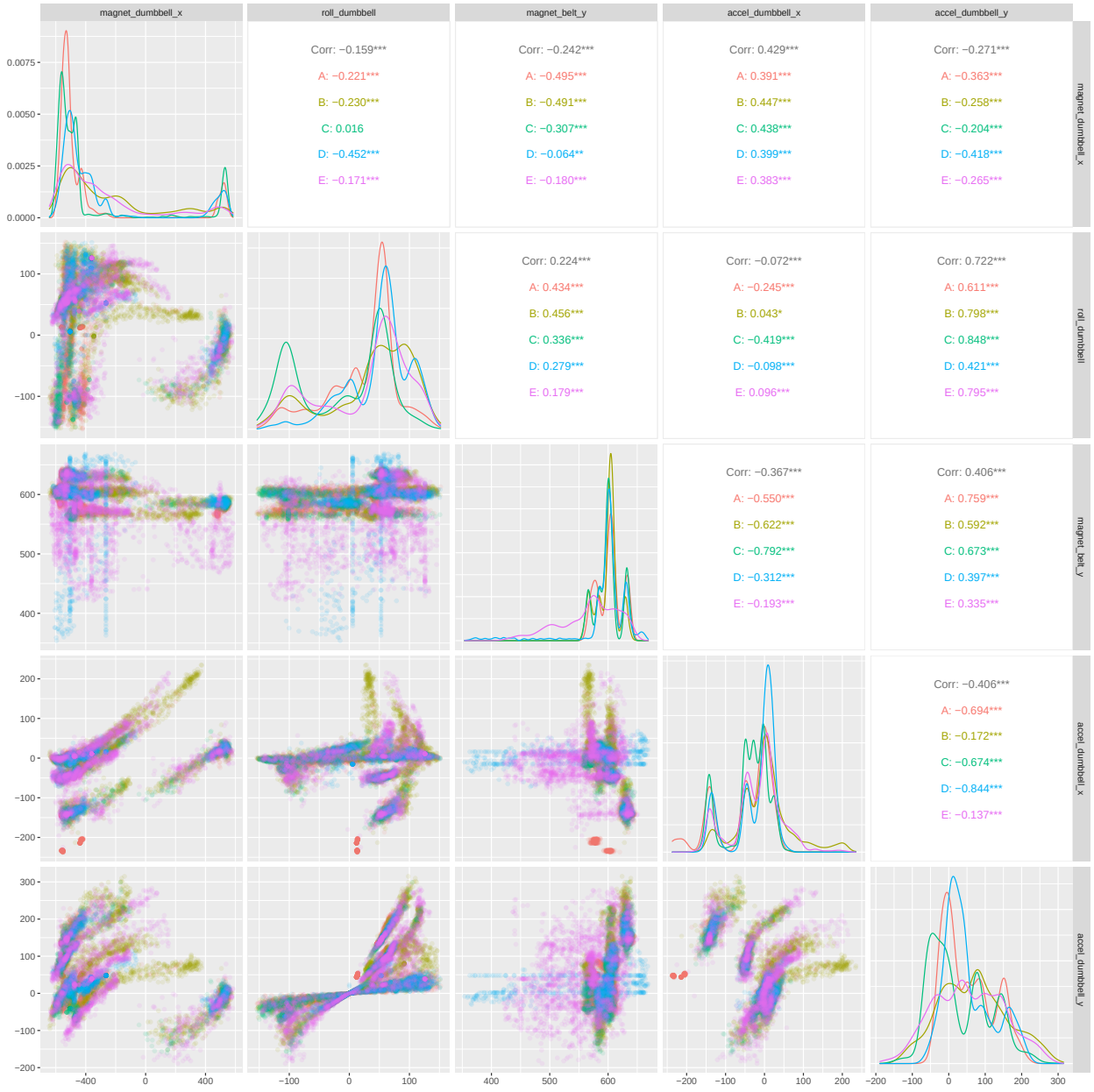
One interesting thing to note here is that the dumbbell sensor turned out to be the most important sensor among the 4. I would like to explore that in future works.

Pairs plot

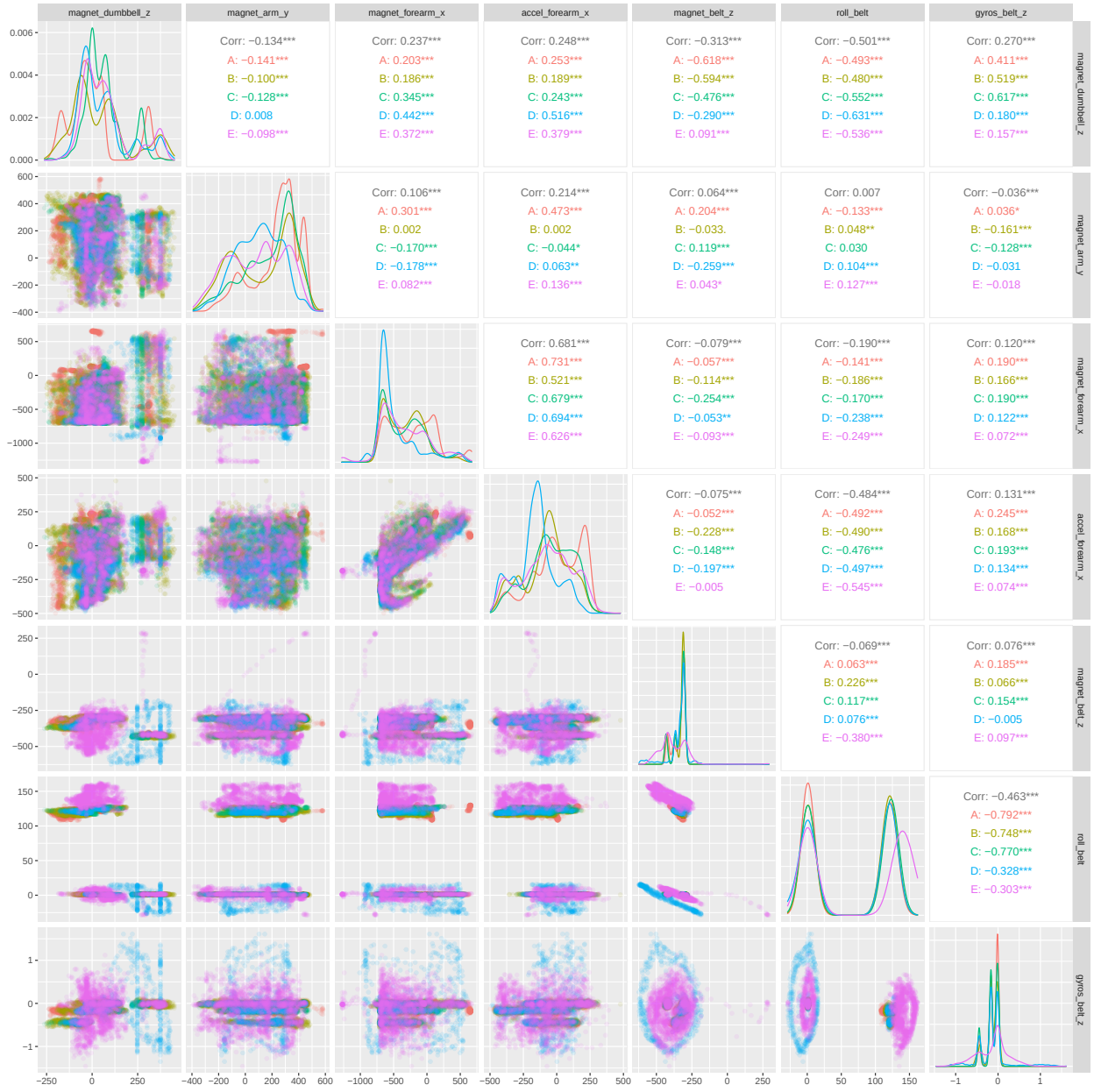
```
my_dens <- function(data, mapping, ...) {
  ggplot(data = data, mapping=mapping) +
    geom_density(..., alpha = 0.3)+scale_fill_brewer(palette="Set2")
}
my_point <- function(data, mapping, ...) {
  ggplot(data = data, mapping=mapping) +
    geom_point(..., alpha = 0.1)+ scale_fill_brewer(palette="Set2")
}
ggpairs(dfAnalize2, columns = 1:5,aes(color = classe),
        lower = list(continuous = my_point),diag = list(continuous = my_dens))
```




```
ggpairs(dfAnalize2, columns = 6:10,aes(color = classe),
        lower = list(continuous = my_point),diag = list(continuous = my_dens))
```




```
ggpairs(dfAnalize2, columns = 11:17,aes(color = classe),
        lower = list(continuous = my_point),diag = list(continuous = my_dens))
```



So we can see that most of the features are very skewed, so as a preprocessing step we have to "center", "rescale" and use "BoxCox" the features.
Some of the features have very interesting scatter plot. But as I am in a time constrain for this project I'll revisit this later.

Its model time

In the above section we have narrowed down to 17 predictors and also we have decided to use 3 preprocessing steps. In this section we will build on the analyzed data to create models for prediction. We will use Classification tree, Random Forest, Generalized Linear regression and SVM , them we will stack it with a Random forest to get the final model.

```
dfTrainF <- dfTrain %>% select(final_cols,classe)
dfValF <- dfVal %>% select(final_cols,classe)
dfTrainF[,1:17] <- sapply(dfTrainF[,1:17],as.numeric)
dfValF[,1:17] <- sapply(dfValF[,1:17],as.numeric)
levels <- c("A", "B", "C", "D", "E")
preprop_obj <- preProcess(dfTrainF[,-18],method = c("center","scale","BoxCox"))
xTrain <- predict(preprop_obj,select(dfTrainF,-classe))
yTrain <- factor(dfTrainF$classe,levels=levels)
xVal <- predict(preprop_obj,select(dfValF,-classe))
yVal <- factor(dfValF$classe,levels=levels)
trControl <- trainControl(method="cv", number=5)
#CFtree
modelCT <- train(x = xTrain,y = yTrain,
                 method = "rpart", trControl = trControl)
#RF
modelRF <- train(x = xTrain,y = yTrain,
                 method = "rf", trControl = trControl,verbose=FALSE, metric = "Accuracy")
#GBM
#taking too long
modelGBM <- train(x = xTrain,y = yTrain,
                  method = "gbm",trControl=trControl, verbose=FALSE)
#SVM
modelSVM <- svm(x = xTrain,y = yTrain,
                kernel = "polynomial", cost = 10)
```

Let's look the results:

Classification Tree

```
confusionMatrix(predict(modelCT,xVal),yVal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1003  330  319  294  106
##          B   19  256   20  109  103
##          C   93  173  345  240  212
##          D    0    0    0    0    0
##          E    1    0    0    0  300
##
## Overall Statistics
##
##                Accuracy : 0.4853
##                  95% CI : (0.4696, 0.5011)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3271
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8987  0.33729  0.50439   0.0000  0.41609
## Specificity            0.6263  0.92067  0.77833   1.0000  0.99969
## Pos Pred Value         0.4888  0.50493  0.32455      NaN  0.99668
## Neg Pred Value         0.9396  0.85275  0.88147   0.8361  0.88377
## Prevalence             0.2845  0.19347  0.17436   0.1639  0.18379
## Detection Rate         0.2557  0.06526  0.08794   0.0000  0.07647
## Detection Prevalence   0.5231  0.12924  0.27097   0.0000  0.07673
## Balanced Accuracy      0.7625  0.62898  0.64136   0.5000  0.70789
```

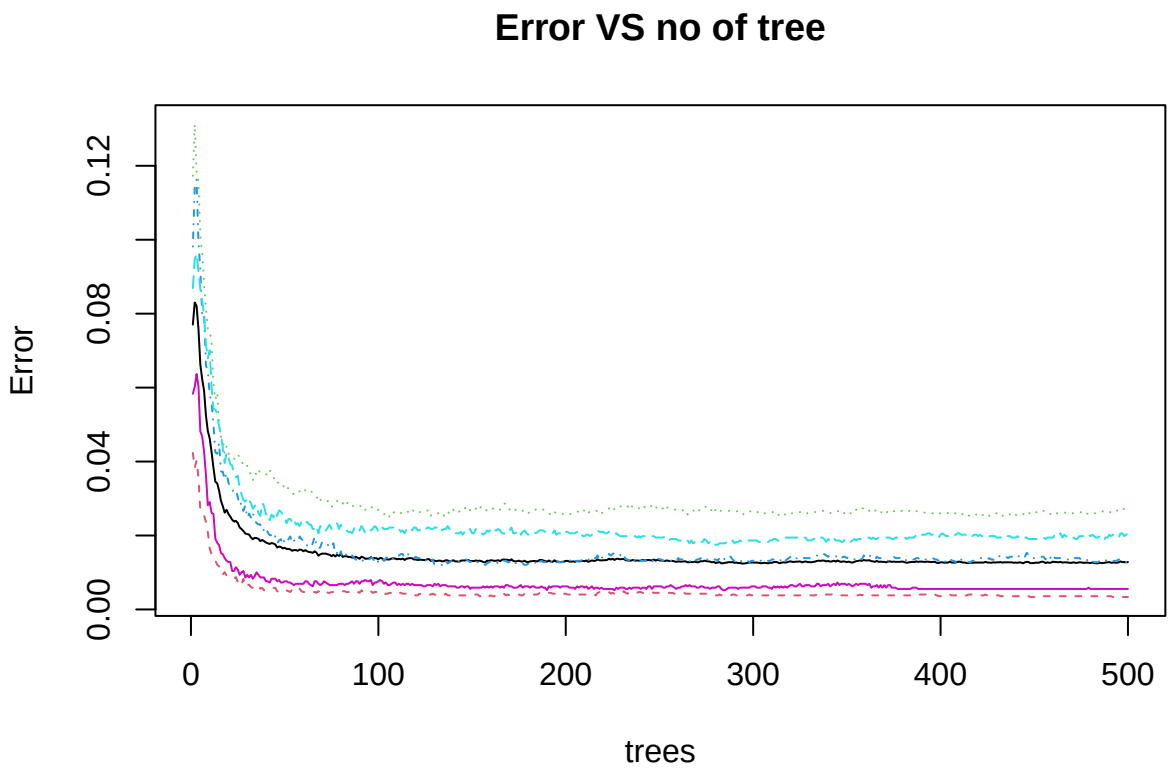
Clearly Classification tree is not performing well, accuracy is very low. One thing to note here is that True classe_A are detected with high accuracy, but other classe are incorrectly predicted as classe_A.

Random Forest

```
confusionMatrix(predict(modelRF,xVal),yVal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1112    7    0    0    0
##          B    3  741    5    3    1
##          C    1    7  676   15    4
##          D    0    4    3  625    1
##          E    0    0    0    0  715
##
## Overall Statistics
##
##                Accuracy : 0.9862
##                  95% CI : (0.9821, 0.9896)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9826
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9763   0.9883   0.9720   0.9917
## Specificity            0.9975   0.9962   0.9917   0.9976   1.0000
## Pos Pred Value         0.9937   0.9841   0.9616   0.9874   1.0000
## Neg Pred Value         0.9986   0.9943   0.9975   0.9945   0.9981
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2835   0.1889   0.1723   0.1593   0.1823
## Detection Prevalence   0.2852   0.1919   0.1792   0.1614   0.1823
## Balanced Accuracy      0.9970   0.9862   0.9900   0.9848   0.9958
```

```
plot(modelRF$finalModel,main="Error VS no of tree")
```



Random Forest took the lead with 98%+ accuracy.

GBM

```
confusionMatrix(predict(modelGBM,xVal),yVal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1083   40    2    5    3
##          B   18  642   32   16   11
##          C    8   54  635   39   10
##          D    4   21   14  582    9
##          E    3    2    1    1  688
##
## Overall Statistics
##
##                Accuracy : 0.9253
##                  95% CI : (0.9166, 0.9333)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9055
##
##  Mcnemar's Test P-Value : 2.509e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9704   0.8458   0.9284   0.9051   0.9542
## Specificity            0.9822   0.9757   0.9657   0.9854   0.9978
## Pos Pred Value         0.9559   0.8929   0.8512   0.9238   0.9899
## Neg Pred Value         0.9882   0.9635   0.9846   0.9815   0.9898
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2761   0.1637   0.1619   0.1484   0.1754
## Detection Prevalence   0.2888   0.1833   0.1902   0.1606   0.1772
## Balanced Accuracy      0.9763   0.9108   0.9470   0.9452   0.9760
```

Clearly GBM is also doing good but RF is still the best.

SVM

```
confusionMatrix(predict(modelSVM,xVal),yVal)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1096   40   18   17    2
##          B    1  676   15    5    6
##          C    9   40  640   45    3
##          D   10    3    9  575    9
##          E    0    0    2    1  701
##
## Overall Statistics
##
##                Accuracy : 0.9401
##                  95% CI : (0.9322, 0.9473)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9241
##
##  Mcnemar's Test P-Value : 1.808e-15
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9821   0.8906   0.9357   0.8942   0.9723
## Specificity            0.9726   0.9915   0.9701   0.9905   0.9991
## Pos Pred Value         0.9344   0.9616   0.8684   0.9488   0.9957
```

```
## Neg Pred Value         0.9927   0.9742   0.9862   0.9795   0.9938
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2794   0.1723   0.1631   0.1466   0.1787
## Detection Prevalence   0.2990   0.1792   0.1879   0.1545   0.1795
## Balanced Accuracy      0.9773   0.9411   0.9529   0.9424   0.9857
```

So it clear that SVM is giving good accuracy, though it not as good as random forest.

Results

So from the above analysis it is clear that Random Forest is taking the lead in term of prediction. Now lets see how good it does in the Test set given in the Coursera project.

```
dfTest2 <- dfTest %>% select(final_cols,problem_id)
xTest <- dfTest2 %>% select(final_cols)

result <- data.frame("problem_id" = dfTest$problem_id,
                     "PREDICTION_RF" = predict(modelRF,xTest),
                     "PREDICTION_GBM" = predict(modelGBM,xTest),
                     "PREDICTION_SVM" = predict(modelSVM,xTest))
result
```

```
##    problem_id PREDICTION_RF PREDICTION_GBM PREDICTION_SVM
## 1           1             E              E              C
## 2           2             A              E              A
## 3           3             A              D              B
## 4           4             E              E              C
## 5           5             E              E              A
## 6           6             E              D              C
## 7           7             E              E              B
## 8           8             B              D              A
## 9           9             A              B              E
## 10         10             E              E              E
## 11         11             A              E              C
## 12         12             A              D              C
## 13         13             E              B              E
## 14         14             A              D              B
## 15         15             E              E              B
## 16         16             E              E              A
## 17         17             E              E              C
## 18         18             B              E              A
## 19         19             E              E              A
## 20         20             E              E              E
```

This is odd- UPDATE

My validation accuracy is good but my score in the quize was not correct. Maybe I should have not droped the columns when I selected 17 features. As Random Forest is doing so go I'll train one more RF with all the data and see if it helps. But still I dont understand why even after my accuracy on validation is above 95, how it is so bad. Need to look into it.

So for this update I will use all the predictors. And as there are so many columns I'll make it parallel so it doesnot take that long.

```
dfTrainF2 <- tbl_df(dfTrain %>%
                      select(-column_2drop,
                             -c(X,user_name, raw_timestamp_part_1,
                                raw_timestamp_part_2, cvtd_timestamp,
                                new_window,num_window)))
xTrain2 <- dfTrainF2 %>% select(-classe)
xTrain2 <- sapply(xTrain2,as.numeric)
yTrain2 <- factor(dfTrainF2$classe,levels=levels)
dfValF2 <- tbl_df(dfVal %>%
                      select(-column_2drop,
                             -c(X,user_name, raw_timestamp_part_1,
                                raw_timestamp_part_2, cvtd_timestamp,
                                new_window,num_window)))
xVal2 <- dfValF2 %>% select(-classe)
xVal2 <- sapply(xVal2,as.numeric)
yVal2 <- factor(dfValF2$classe,levels=levels)
dfTestF2 <- tbl_df(dfTest %>%
                      select(-column_2drop,
                             -c(X,user_name, raw_timestamp_part_1,
                                raw_timestamp_part_2, cvtd_timestamp,
                                new_window,num_window)))
xTest2 <- dfTestF2 %>% select(-problem_id)
xTest2 <- sapply(xTest2,as.numeric)
pb_id <- dfValF2$classe
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
ncores <- makeCluster(detectCores() - 1)
registerDoParallel(cores=ncores)
getDoParWorkers()
```

```
## [1] 3
```

```
modelRF2 <- train(x = xTrain2,y = yTrain2, method = "rf",
                  metric = "Accuracy",
                  trControl=trainControl(method = "cv", number = 4,
                                         p= 0.60, allowParallel = TRUE ))
```

```
#Check the result
result2 <- data.frame("problem_id" = dfTest$problem_id,
                      "PREDICTION_RF" = predict(modelRF,xTest),
                      "PREDICTION_GBM" = predict(modelGBM,xTest),
                      "PREDICTION_SVM" = predict(modelSVM,xTest),
                      "PREDICTION_RF2_ALL_COL"=predict(modelRF2,xTest2))
result2
```

```
##    problem_id PREDICTION_RF PREDICTION_GBM PREDICTION_SVM
## 1           1             E              E              C
## 2           2             A              E              A
## 3           3             A              D              B
## 4           4             E              E              C
## 5           5             E              E              A
## 6           6             E              D              C
## 7           7             E              E              B
## 8           8             B              D              A
## 9           9             A              B              E
## 10         10             E              E              E
## 11         11             A              E              C
## 12         12             A              D              C
## 13         13             E              B              E
## 14         14             A              D              B
## 15         15             E              E              B
## 16         16             E              E              A
## 17         17             E              E              C
## 18         18             B              E              A
## 19         19             E              E              A
## 20         20             E              E              E
##    PREDICTION_RF2_ALL_COL
## 1                       B
## 2                       A
## 3                       B
## 4                       A
## 5                       A
## 6                       E
## 7                       D
## 8                       B
## 9                       A
## 10                      A
## 11                      B
## 12                      C
## 13                      B
## 14                      A
## 15                      E
## 16                      E
## 17                      A
## 18                      B
## 19                      B
## 20                      B
```