Predictions using the Weight Lifting Exercises Dataset

Amr Hatem

August-2018

Executive Summary

Based on a dataset provide by HAR [http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)](http://groupware.les.inf.puc-rio.br/har) we will try to train a predictive model to predict what exercise was performed using a dataset with 159 features

We'll take the following steps:

- Process the data, for use of this project
- Explore the data, especially focussing on the two paramaters we are interested in
- Model selection, where we try different models to help us answer our questions
- Model examination, to see wether our best model holds up to our standards
- A Conclusion where we answer the questions based on the data
- Predicting the classification of the model on test set

Processing

First change 'am' to factor (0 = automatic, 1 = manual) And make cylinders a factor as well (since it is not continious)

```
training.raw<-read.csv("pml-training.csv",
testing.raw<-read.csv("pml-testing.csv",
```

Exploratory data analyses

Look at the dimensions & head of the dataset to get an idea

```
# Res 1
dim(training.raw)
```

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

```
# Res 2 - excluded because excessivness
# head(training.raw)

# Res 3 - excluded because excessivness
#str(training.raw)

# Res 4 - excluded because excessivness
#summary(training.raw)
```

What we see is a lot of data with NA / empty values. Let's remove those

```
maxNAPerc = 20
maxNACount <- nrow(training.raw)/100*maxNAPerc
removeColumns <- which(colSums(is.na(training.raw)|training.raw=="")>maxNACount)
training.cleaned01 <- training.raw[,-removeColumns]
testing.cleaned01 <- testing.raw[,-removeColumns]
```

Also remove all time related data, since we won't use those

```
removeColumns <- grep("timestamp", names(training.cleaned01))
training.cleaned02 <- training.cleaned01[,-c(1, removeColumns)]
testing.cleaned02 <- testing.cleaned01[,-c(1, removeColumns)]
```

Then convert all factors to integers

```
classeLevels <- levels(training.cleaned02$classe)
training.cleaned03 <- data.frame(data.matrix(training.cleaned02))
training.cleaned03$classe <- factor(training.cleaned03$classe, labels=classeLevels)
testing.cleaned03 <- data.frame(data.matrix(testing.cleaned02))
```

Finally set the dataset to be explored

```
training.cleaned <- training.cleaned03
testing.cleaned <- testing.cleaned03
```

Exploratory data analyses

Since the test set provided is the the ultimate validation set, we will split the current training in a test and train set to work with.

```
set.seed(19791108)
library(caret)

classeIndex <- which(names(training.cleaned) == "classe")

partition <- createDataPartition(y=training.cleaned$classe, p=0.75, list=FALSE)
training.subSetTrain <- training.cleaned[partition, ]
training.subSetTest <- training.cleaned[-partition, ]
```

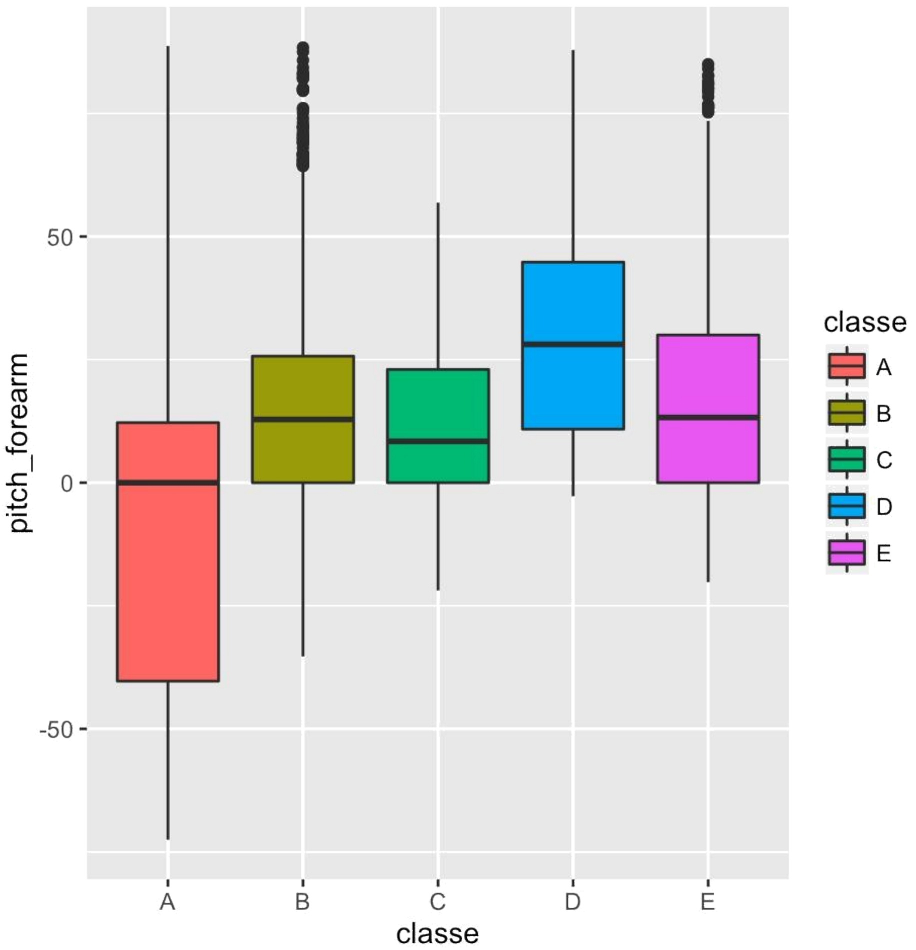
What are some fields that have high correlations with the classe?

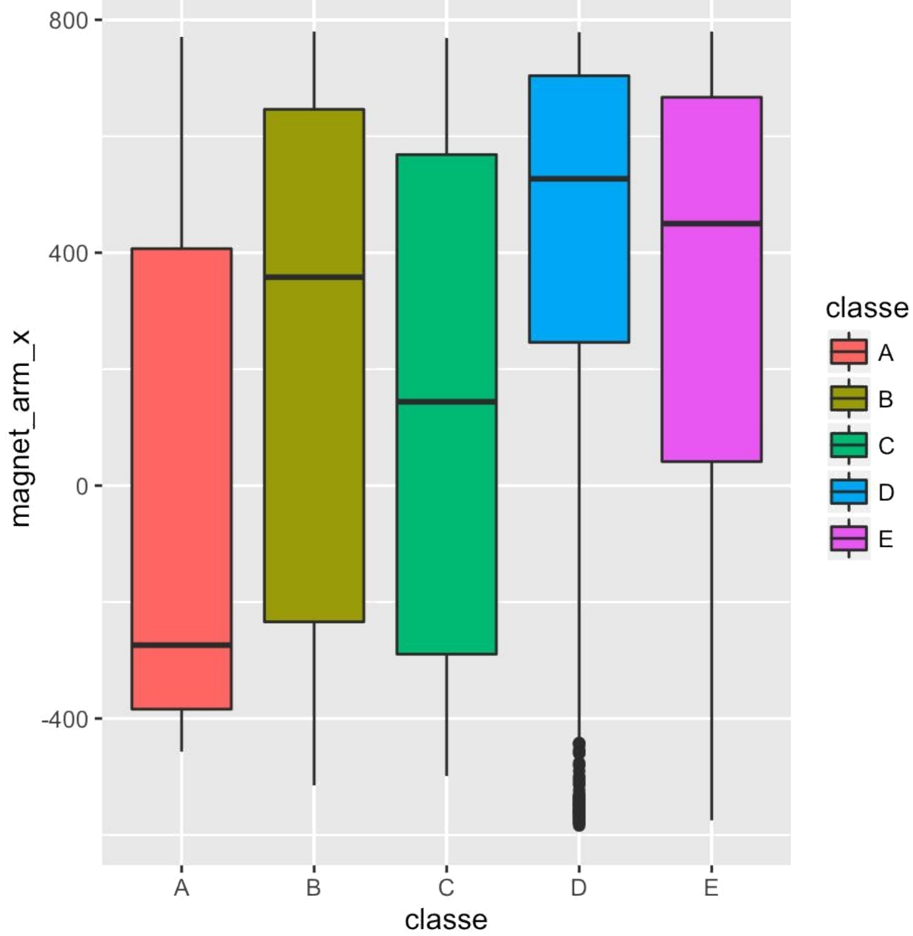
```
correlations <- cor(training.subSetTrain[, -classeIndex], as.numeric(training.subSetTrain$classe))
bestCorrelations <- subset(as.data.frame(as.table(correlations)), abs(Freq)>0.3)
bestCorrelations
```

```
##             Var1 Var2       Freq
## 27  magnet_arm_x    A  0.3023806
## 44 pitch_forearm    A  0.3475548
```

Even the best correlations with classe are hardly above 0.3 Let’s check visually if there is indeed hard to use these 2 as possible simple linear predictors.

```
library(RmiSc)
library(ggplot2)

p1 <- ggplot(training.subSetTrain, aes(classe,pitch_forearm))+
    geom_boxplot(aes(fill=classe))

p2 <- ggplot(training.subSetTrain, aes(classe, magnet_arm_x))+
geom_boxplot(aes(fill=classe))

multiplot(p1,p2,cols=2)
```



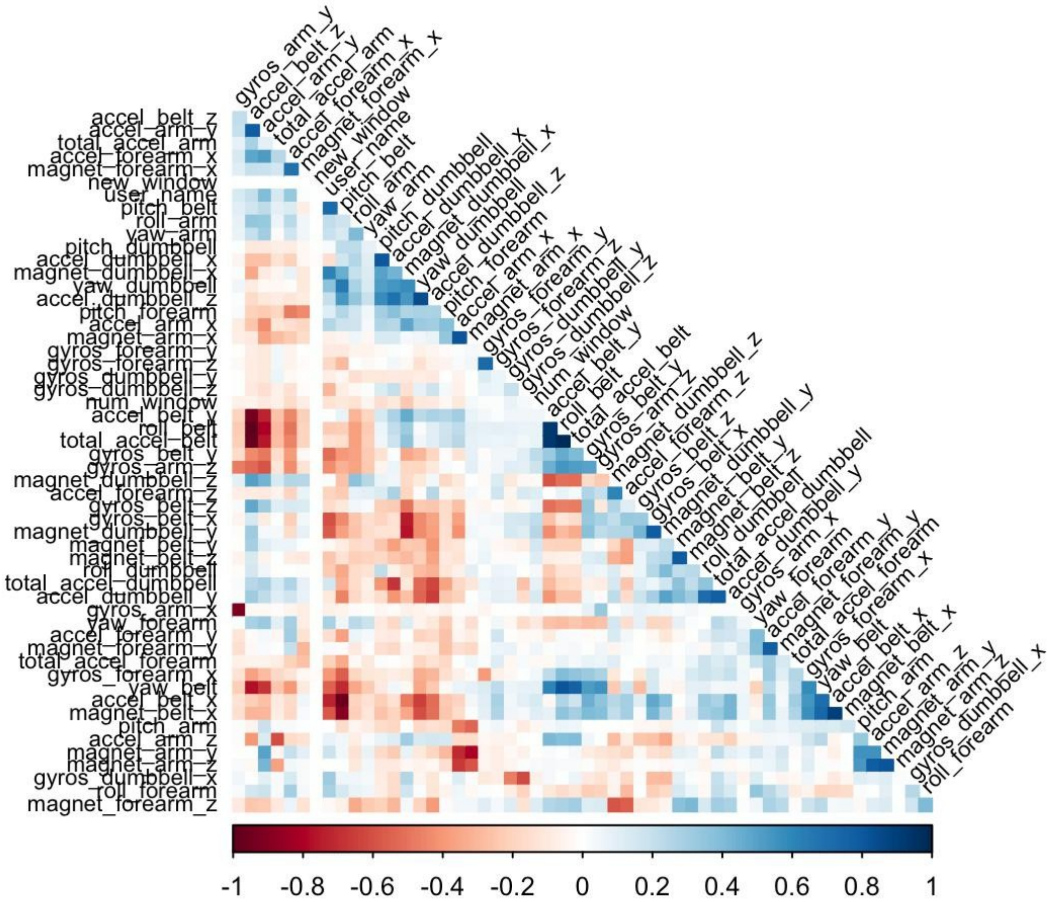


Clearly there is no hard seperation of classes possible using only these ‘highly’ correlated features. Let’s train some models to get closer to a way of predicting these classe’s

Model selection

Let’s identify variables with high correlations amongst each other in our set, so we can possibly exclude them from the pca or training. We will check afterwards if these modifications to the dataset make the model more accurate (and perhaps even faster)

```
library(corrplot)
correlationMatrix <- cor(training.subSetTrain[, -classeIndex])
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.9, exact=TRUE)
excludeColumns <- c(highlyCorrelated, classeIndex)
corrplot(correlationMatrix, method="color", type="lower", order="hclust", tl.cex=0.70, tl.col="black", tl.srt=45, diag=FALSE)
```



We see that there are some features that aree quite correlated with each other. We will have a model with these excluded. Also we'll try and reduce the features by running PCA on all and the excluded subset of the features

```
pcaPreProcess.all <- preProcess(training.subSetTrain[,-classeIndex], method = "pca", thresh = 0.99)
training.subSetTrain.pca.all <- predict(pcaPreProcess.all, training.subSetTrain[,-classeIndex])
training.subSetTest.pca.all <- predict(pcaPreProcess.all, training.subSetTest[,-classeIndex])
testing.pca.all <- predict(pcaPreProcess.all, testing.cleaned[,-classeIndex])


pcaPreProcess.subset <- preProcess(training.subSetTrain[,-excludeColumns], method = "pca", thresh = 0.99)
training.subSetTrain.pca.subset <- predict(pcaPreProcess.subset, training.subSetTrain[,-excludeColumns])
training.subSetTest.pca.subset <- predict(pcaPreProcess.subset, training.subSetTest[,-excludeColumns])
testing.pca.subset <- predict(pcaPreProcess.subset, testing.cleaned[,-classeIndex])
```

Now we'll do some actual Random Forest training. We'll use 200 trees, because I've already seen that the error rate doesn't decline a lot after say 50 trees, but we still want to be thorough. Also we will time each of the 4 random forest models to see if when all else is equal one pops out as the faster one.

```
library(randomForest)

ntree <- 200 #This is enough for great accuracy (trust me, I'm an engineer).

start <- proc.time()
rfMod.cleaned <- randomForest(
  x=training.subSetTrain[,-classeIndex],
  y=training.subSetTrain$classe,
  xtest=training.subSetTest[,-classeIndex],
  ytest=training.subSetTest$classe,
  ntree=ntree,
  keep.forest=TRUE,
  proximity=TRUE) #do.trace=TRUE
proc.time() - start
```

```
##    user  system elapsed
## 123.664   6.228 132.399
```

```
start <- proc.time()
rfMod.exclude <- randomForest(
  x=training.subsetTrain[, -excludeColumns],
  y=training.subsetTrain$classe,
  xtest=training.subsetTest[, -excludeColumns],
  ytest=training.subsetTest$classe,
  ntree=ntree,
  keep.forest=TRUE,
  proximity=TRUE) #do.trace=TRUE
proc.time() - start
```

```
##    user  system elapsed
## 116.568   4.162 121.669
```

```
start <- proc.time()
rfMod.pca.all <- randomForest(
  x=training.subsetTrain.pca.all,
  y=training.subsetTrain$classe,
  xtest=training.subsetTest.pca.all,
  ytest=training.subsetTest$classe,
  ntree=ntree,
  keep.forest=TRUE,
  proximity=TRUE) #do.trace=TRUE
proc.time() - start
```

```
##    user  system elapsed
## 111.267   4.004 115.881
```

```
start <- proc.time()
rfMod.pca.subset <- randomForest(
  x = training.subsetTrain.pca.subset,
  y = training.subsetTrain$classe,
  xtest = training.subsetTest.pca.subset,
  ytest = training.subsetTest$classe,
  ntree = ntree,
  keep.forest = TRUE,
  proximity = TRUE) #do.trace=TRUE
proc.time() - start
```

```
##    user  system elapsed
## 120.846   7.504 138.42
```

Model examination

Now that we have 4 trained models, we will check the accuracies of each. (There probably is a better way, but this still works good)

```
rfMod.cleaned
```

```
##
## Call:
##  randomForest(x = training.subSetTrain[, -classeIndex], y = training.subSetTrain$classe,      xtest = training.subSetTest[, -classeIndex], ytest = training.subSetTest$classe,      ntree = ntree, proximity = TRUE, keep.forest = TRUE)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.27%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4185    0    0    0    0 0.000000000
## B    7 2839    2    0    0 0.003160112
## C    0    8 2558    1    0 0.003506038
## D    0    0   16 2396    0 0.006633499
## E    0    0    0    6 2700 0.002217295
##                 Test set error rate: 0.29%
## Confusion matrix:
##      A   B   C   D   E  class.error
## A 1394   0   0   0   1 0.0007168459
## B    0 949   0   0   0 0.0000000000
## C    0   8 847   0   0 0.0093567251
## D    0   0   2 801   1 0.0037313433
## E    0   0   0   2 899 0.0022197558
```

```
rfMod.cleaned.training.acc <- round(1-sum(rfMod.cleaned$confusion[, 'class.error']),3)
paste0("Accuracy on training: ",rfMod.cleaned.training.acc)
```

```
## [1] "Accuracy on training: 0.984"
```

```
rfMod.cleaned.testing.acc <- round(1-sum(rfMod.cleaned$test$confusion[, 'class.error']),3)
paste0("Accuracy on testing: ",rfMod.cleaned.testing.acc)
```

```
## [1] "Accuracy on testing: 0.984"
```

```
rfMod.exclude
```

```
##
## Call:
##  randomForest(x = training.subSetTrain[, -excludeColumns], y = training.subSetTrain$classe, xtest = training.subSetTest[,-excludeColumns], ytest = training.subSetTest$classe,     ntree = ntree, proximity = TRUE, keep.forest = TRUE)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.23%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4185    0    0    0    0 0.000000000
## B    4 2842    2    0    0 0.002106742
## C    0    9 2558    0    0 0.003506038
## D    0    0   13 2398    1 0.005804312
## E    0    0    0    5 2701 0.001847746
##                 Test set error rate: 0.29%
## Confusion matrix:
##      A   B   C   D   E  class.error
## A 1394   0   0   0   1 0.0007168459
## B    1 948   0   0   0 0.0010537408
## C    0   8 847   0   0 0.0093567251
## D    0   0   0 803   1 0.0012437811
## E    0   0   0   3 898 0.0033296337
```

```
rfMod.exclude.training.acc <- round(1-sum(rfMod.exclude$confusion[, 'class.error']),3)
paste0("Accuracy on training: ",rfMod.exclude.training.acc)
```

```
## [1] "Accuracy on training: 0.987"
```

```
rfMod.exclude.testing.acc <- round(1-sum(rfMod.exclude$test$confusion[, 'class.error']),3)
paste0("Accuracy on testing: ",rfMod.exclude.testing.acc)
```

```
## [1] "Accuracy on testing: 0.984"
```

```
rfMod.pca.all
```

```
##
## Call:
##  randomForest(x = training.subSetTrain.pca.all, y = training.subSetTrain$classe,      xtest = training.subSetTest.pca.all, ytest = training.subSetTest$classe,      ntree = ntree, proximity = TRUE, keep.forest = TRUE)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 2.13%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4169    5    1    9    1 0.003823178
## B   57 2761   27    2    1 0.030547753
## C    2   35 2505   18    7 0.024152707
## D    2    2   89 2311    8 0.041873964
## E    4   15   12   16 2659 0.017368810
##                 Test set error rate: 1.96%
## Confusion matrix:
##      A   B   C   D   E class.error
## A 1389   1   1   3   1 0.004301075
## B   13 922  12   1   1 0.028451001
## C    1  15 833   5   1 0.025730994
## D    2   0  24 775   3 0.036069652
## E    0   4   3   5 889 0.013318535
```

```
rfMod.pca.all.training.acc <- round(1-sum(rfMod.pca.all$confusion[, 'class.error']),3)
paste0("Accuracy on training: ",rfMod.pca.all.training.acc)
```

```
## [1] "Accuracy on training: 0.882"
```

```
rfMod.pca.all.testing.acc <- round(1-sum(rfMod.pca.all$test$confusion[, 'class.error']),3)
paste0("Accuracy on testing: ",rfMod.pca.all.testing.acc)
```

```
## [1] "Accuracy on testing: 0.892"
```

```
rfMod.pca.subset
```

```
## 
## Call:
##  randomForest(x = training.subSetTrain.pca.subset, y = training.subSetTrain$classe,      xtest = training.subSetTest.pca.subset, ytest = training.subSetTest$classe,      ntree = ntree, proximity = TRUE, keep.forest = TRUE) 
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 6
## 
##         OOB estimate of  error rate: 2.34%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4159    9    6   10    1 0.006212664
## B   60 2758   26    3    1 0.031601124
## C    7   31 2507   19    3 0.023373588
## D    8    1   97 2302    4 0.045605307
## E    7   16   20   15 2648 0.021433851
##                 Test set error rate: 2.28%
## Confusion matrix:
##      A   B   C   D   E class.error
## A 1382   5   5   2   1 0.009318996
## B   13 924  11   0   1 0.026343519
## C    2  19 825   8   1 0.035087719
## D    4   0  26 770   4 0.042288557
## E    0   3   0   7 891 0.011098779
```

```
rfMod.pca.subset.training.acc <- round(1-sum(rfMod.pca.subset$confusion[, 'class.error']),3)
paste0("Accuracy on training: ",rfMod.pca.subset.training.acc)
```

```
## [1] "Accuracy on training: 0.872"
```

```
rfMod.pca.subset.testing.acc <- round(1-sum(rfMod.pca.subset$test$confusion[, 'class.error']),3)
paste0("Accuracy on testing: ",rfMod.pca.subset.testing.acc)
```

```
## [1] "Accuracy on testing: 0.876"
```

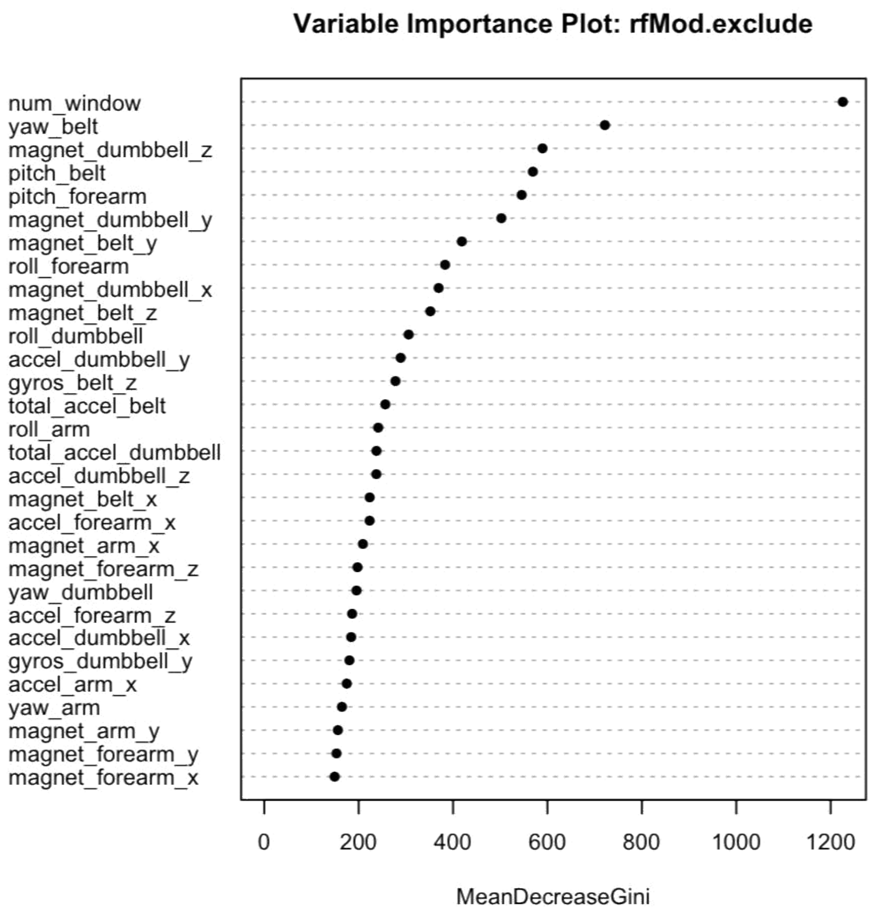
Conclusion

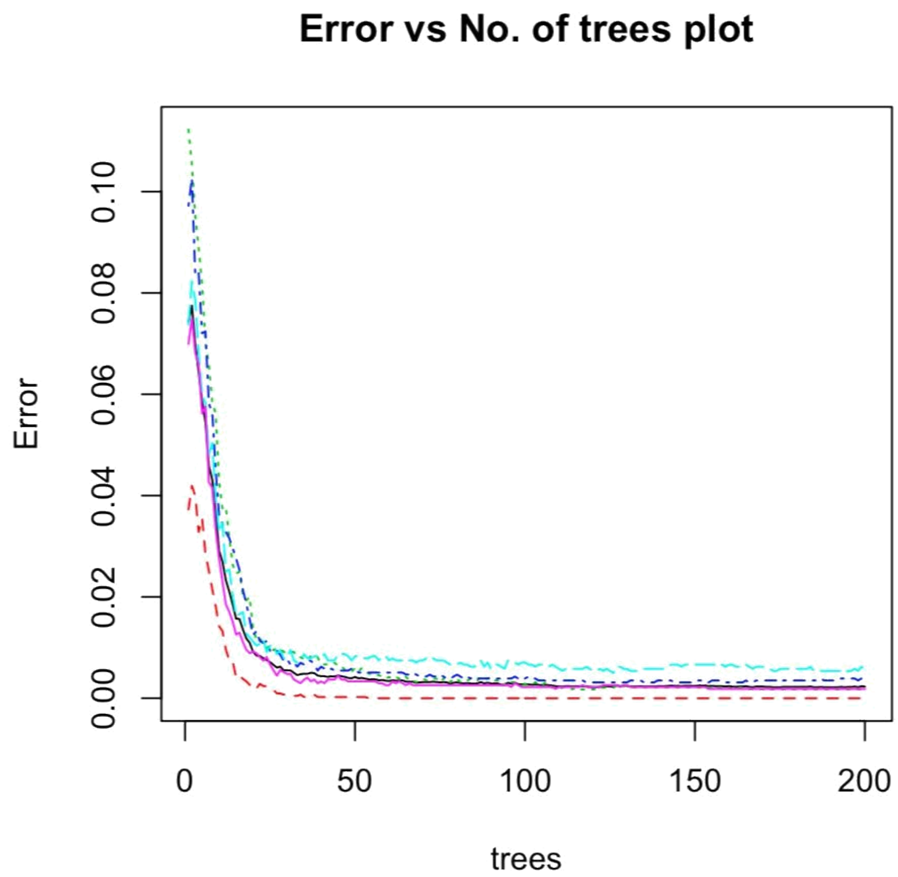
This concludes that nor PCA doesn't have a positive of the accuracy (or the process time for that matter) The `rfMod.exclude` perform's slightly better then the 'rfMod.cleaned'

We'll stick with the `rfMod.exclude` model as the best model to use for predicting the test set. Because with an accuracy of 98.7% and an estimated OOB error rate of 0.23% this is the best model.

Before doing the final prediction we will examine the chosen model more in depth using some plots

```
par(mfrow=c(1,2))
varImpPlot(rfMod.exclude, cex=0.7, pch=16, main='Variable Importance Plot: rfMod.exclude')
plot(rfMod.exclude, , cex=0.7, main='Error vs No. Of trees Plot')
```

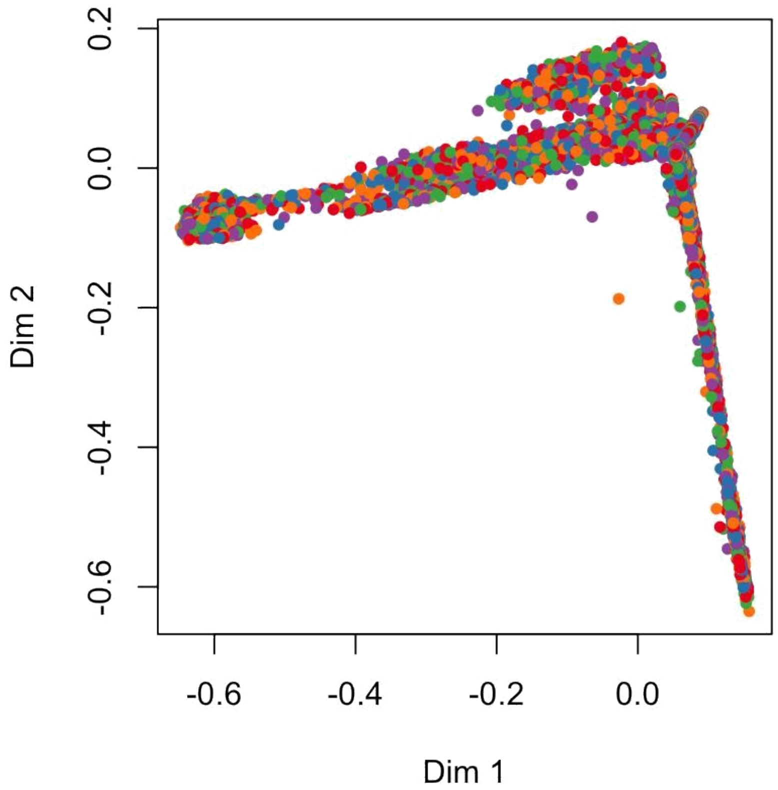





```
par(mfrow= c(1, 1))
```

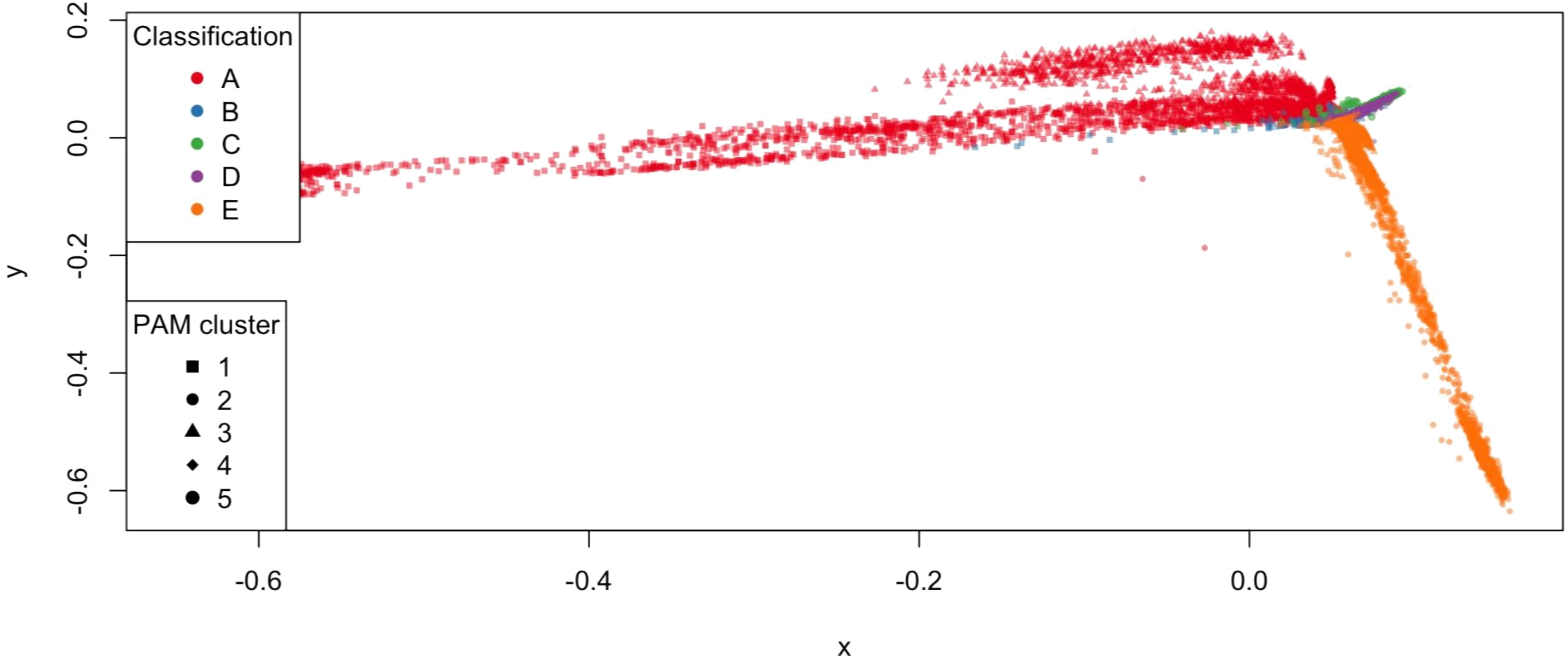
To really look in depth at the distances between predictions we can use MDSplot and cluster predictiosn and results

```
start <- proc.time()

library(RColorBrewer)
palette <- brewer.pal(length(classeLevels), "Set1")
rfMod.mds <- MDSplot(rfMod.exclude, as.factor(classeLevels), k=2, pch=20, palette=palette)
```



0.2
0.0
-0.2
-0.4
-0.6
Dim 2
-0.6
-0.4
-0.2
0.0
Dim 1

```
library(cluster)
rfMod.pam <- pam(1-rfMod.exclude$proximity, k=length(classeLevels), diss=TRUE)

plot(
   rfMod.mds$points[,1],
   rfMod.mds$points[,2],
   pch=rfMod.pam$clustering+14,
   col=alpha(palette[as.numeric(training.subSetTrain$classe)],0.5),
   bg=alpha(palette[as.numeric(training.subSetTrain$classe)],0.2),
   cex=0.5,
   xlab="x",ylab="y")
legend("bottomleft", legend=unique(rfMod.pam$clustering), pch=seq(15,14+length(classeLevels)), title="PAM cluster")
   legend("topleft", legend=classeLevels, pch=16, col=palette, title="Classification")
```




```
proc.time() - start
```

```
##      user   system  elapsed
## 4524.813   39.624 4588.622
```

Test results

Although we've chosen the `rfMod.exclude` it's still nice to see what the other 3 models would predict on the final test set.

Let's look at predictions for all models on the final test set.

```
predictions <- t(cbind(
    exclude=as.data.frame(predict(rfMod.exclude, testing.cleaned[, -excludeColumns]), optional=TRUE),
    cleaned=as.data.frame(predict(rfMod.cleaned, testing.cleaned), optional=TRUE),
    pcaAll=as.data.frame(predict(rfMod.pca.all, testing.pca.all), optional=TRUE),
    pcaExclude=as.data.frame(predict(rfMod.pca.subset, testing.pca.subset), optional=TRUE)
))
predictions
```

```
##            1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20
## exclude    "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"
## cleaned    "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"
## pcaAll     "B" "A" "C" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"
## pcaExclude "B" "A" "C" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"
```

The predictions don't really change a lot with each model, but since we have most faith in the `rfMod.exclude`, we'll keep that as final answer.