Practical Machine Learning Project- Predictions with the Weight Lifting Exercises Dataset

Elek Dobos

January 26, 2017

Summary Description

A dataset was provided by http://groupware.les.inf.puc-rio.br/har.
A prediction model will attempt to predict which exercise was performed using a dataset with 159 features. The algorithm will consist of the following steps. First the data will be read in and processed. Then there will be some exploration and focus on the data attributes of interest. In addition, there will be selection of a model, followed by testing the model. Finally, we will give it a try and attempt to answer data questions and use the model for predictions.

Data was previously downloaded to our default directory. Read in our data sets from our default directory.

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

Exploratory data analyses Look at the dimensions and header of the dataset to get an idea of what the data set looks like.

```
dim(training.raw)
```

[1] 19622 160

Header was commented for final submission because it looks long and messy, and trying to keep it short since we have a word limit on the report.

Need to clean up the data and remove the NA values.

```
maxNAPerc = 20
maxNACount <- nrow(training.raw) / 100 * maxNAPerc
removeColumns <- which(colSums(is.na(training.raw) | training.raw=="") >
maxNACount)
training.cleaned1 <- training.raw[,-removeColumns]
testing.cleaned1 <- testing.raw[,-removeColumns]
```

Strip the time off of the data.

```
removeColumns <- grep("timestamp", names(training.cleaned01))
training.cleaned2 <- training.cleaned1[,-c(1, removeColumns )]
testing.cleaned2 <- testing.cleaned1[,-c(1, removeColumns )]
classeLevels <- levels(training.cleaned01$classe)
training.cleaned3 <- data.frame(data.matrix(training.cleaned02))
training.cleaned3$classe <- factor(training.cleaned3$classe,
labels=classeLevels)
testing.cleaned3 <- data.frame(data.matrix(testing.cleaned2))
```

Set up the dataset to be explored

```
training.cleaned <- training.cleaned3
testing.cleaned <- testing.cleaned3
```

Split up the current training in a test set and train set in order to prepare for additonal manipulations.

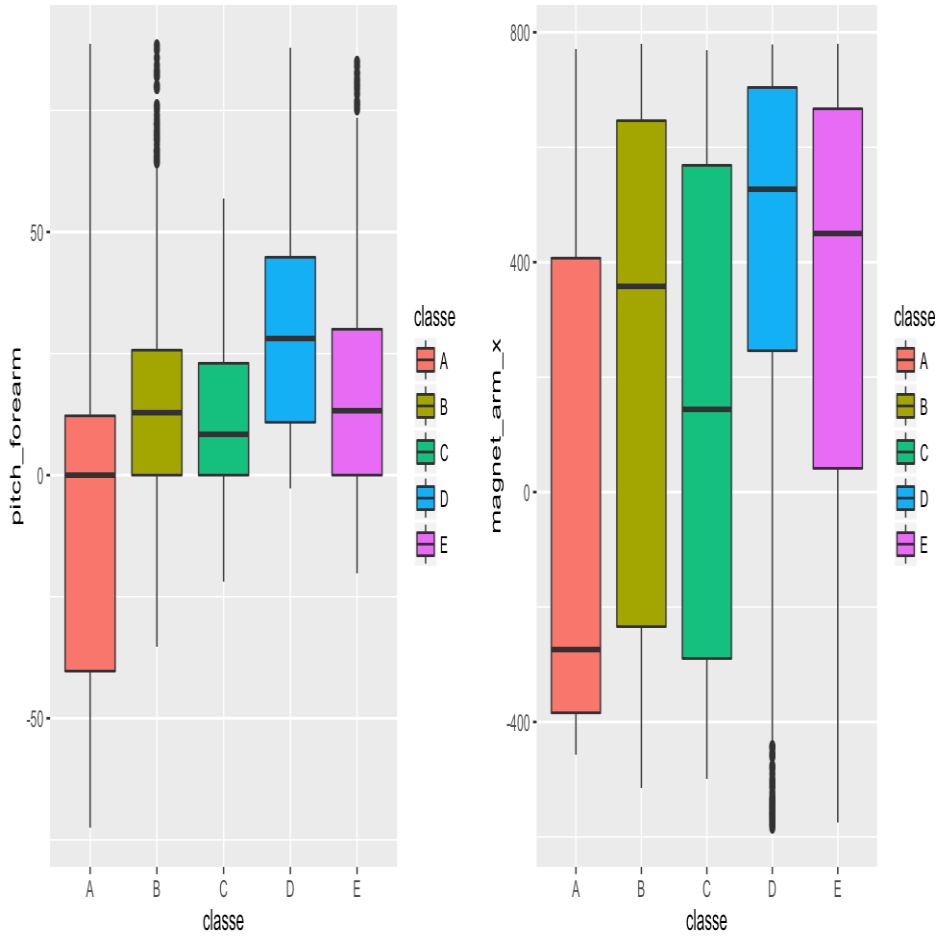
```
set.seed(19791108)
library(caret)

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, ]
```

Determine fields with a good correlation.

```
correlations <- cor(training.subSetTrain[, -classeIndex],
as.numeric(training.subSetTrain$classe))
goodCorrelation <- subset(as.data.frame(as.table(correlations)),
abs(Freq)>0.3)
goodCorrelation
##             Var1 Var2      Freq
## 27  magnet_arm_x    A 0.3023806
## 44 pitch_forearm    A 0.3475548
```

The correlations with classes are running from 0.302 and 0.347. Need to check if these two possible simple linear predictors.

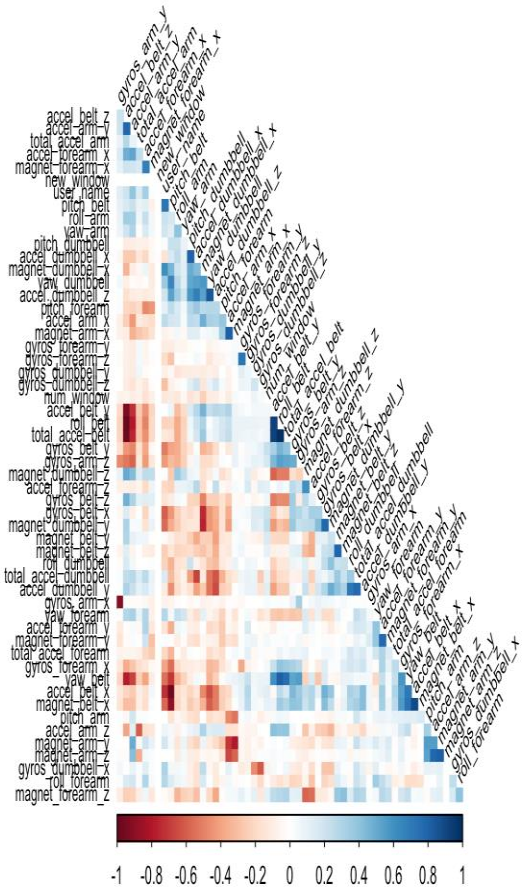
```
library(Rmisc)
library(ggplot2)

p1a <- ggplot(training.subSetTrain, aes(classe,pitch_forearm)) +
  geom_boxplot(aes(fill=classe))

p2b <- ggplot(training.subSetTrain, aes(classe, magnet_arm_x)) +
  geom_boxplot(aes(fill=classe))

multiplot(p1a,p2b,cols=2)
```



Model Evaluation

Identify the variables with good correlation with each other in the data set. Check to determine if the model is more accurate after changes.

```
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)
```



Some of the attributes show a good correlation with each other. There will be a few models examined. We will have a model with these excluded. In addition, try and reduce the features by executing 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])
```

Train utilizing the random forest. Estimate to utilize 200 trees of the random forest. The length of processing time will be examined to see which one of the four models is the quickest.

```
Model 1
library(randomForest)

ntree <- 200

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
Model 2
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
Model 3
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
Model 4
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   7.504 138.427
```

Model Examination

Four models have been trained with the data. Need to check how accurate each model is.

```
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)
```

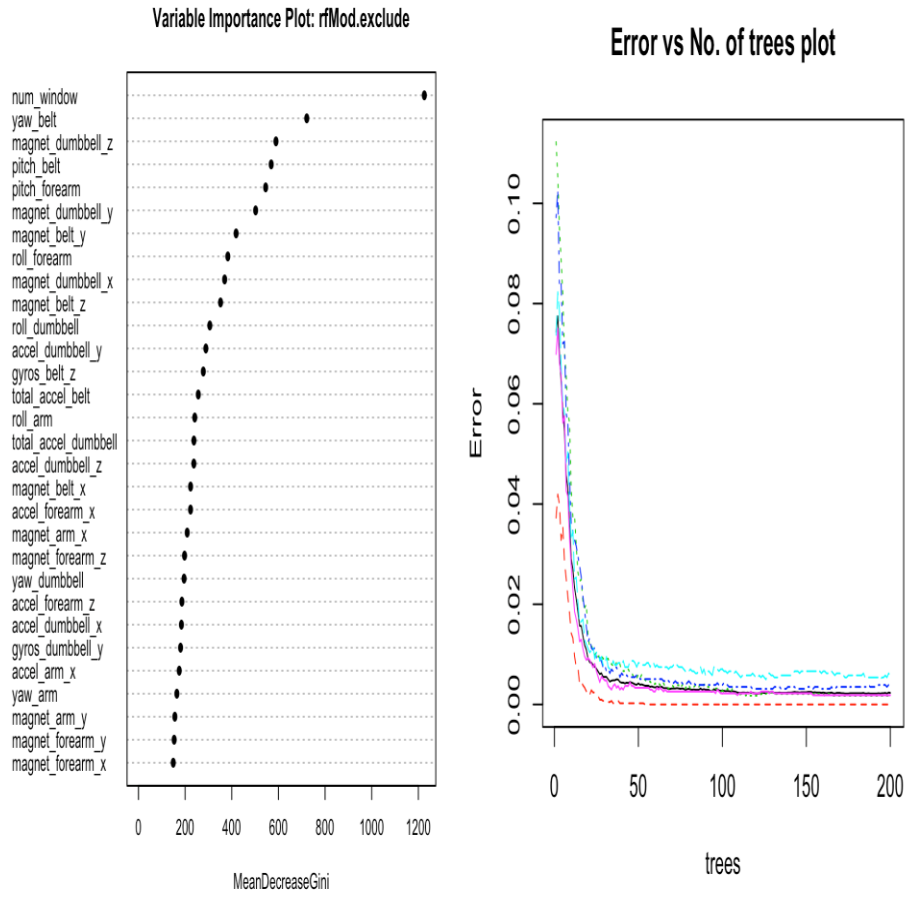
[1] “Accuracy on training: 0.872”

```
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”

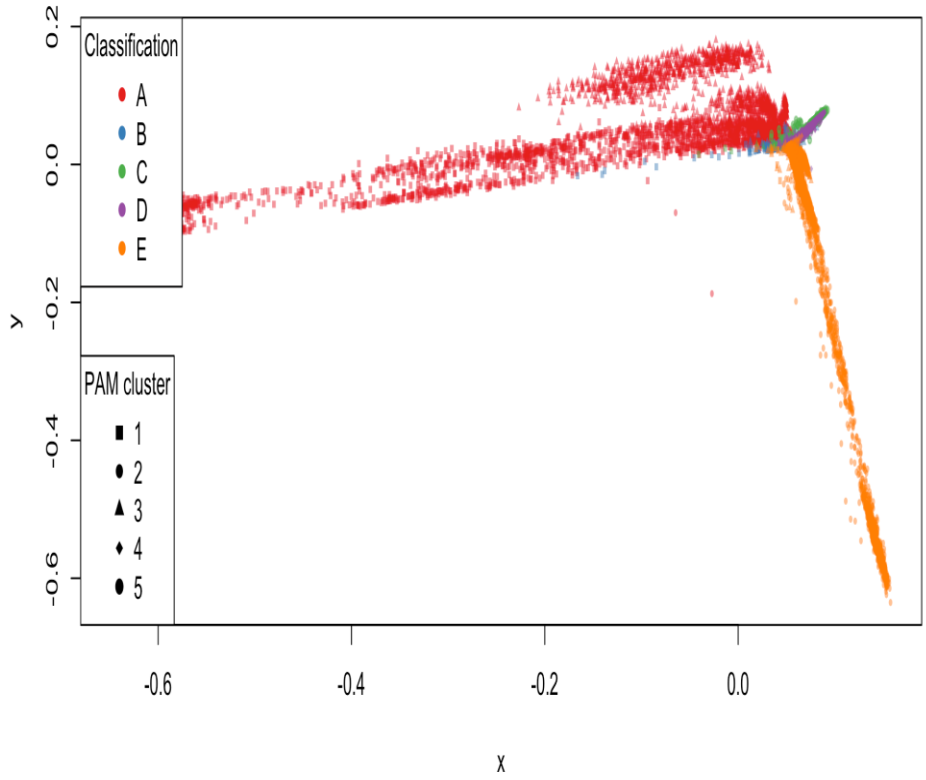
Conclusion

This PCA does not have great accuracy or great processing duration. The rfMod.exclude seems to be a little better than the rfMod.cleaned model. The rfMod.exclude model seems to be the best model to use in our prediction of the test set. The rfMod.exclude model possess an accuracy of 98.7% and an error rate of 0.23%.




```
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))
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
```

Results

The rfMod.exclude is the best model. The other models, cleaned, pcaexclude, and pcaall will also be examined for prediction on the final test set.

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

))

predictions
```

Predictions

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

The predictions have a little bit of variation, but in general did not really change too significantly. rfMod.exclude looks like the best model, and thus we will use those associated values in the prediction for our final answer.