Machine Learning Assignment

Ed

10/12/2020

Summary

This assignment looks at data from wearable fitness devices and aims to train a model to detect how well a user is performing an exercise.

Explore the Data and Transform ready for Training

```r
pmltrain<-read.csv("pml-training.csv")
pmlval<-read.csv("pml-testing.csv")
dim(pmltrain)
```

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

```r
pmltrain<-subset(pmltrain,new_window=="no")
dim(pmltrain)
```

```
## [1] 19216   160
```

```r
dim(pmlval)
```

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

There are 19,622 observations of 160 variables in the training data and 20 observations of 160 variables in the testing data. Included in the variables are the classe of the exercise (A representing the correct performance and B, C, D an E representing various incorrect performances of it), a user name (6 users), time stamp information, 'window' which I haven't found any information about, and the remainder are measurements taken from 4 devices (represented by "_belt“,”_arm“,”_dumbbell" or "_forearm" being included in the name of the variable).

It is noticeable that there are a number of variables which only have any information in them when the variable "new_window" equals Yes. Since "new_window" equals No for all the observations in the testing set, it will not help to include these rows and they are removed above, reducing the number of observations to 19,216.

Now we only want to train on those variables which are not empty or NA so we subset the data to exclude these data variables. We do the same to the test data. Also, apart possibly from the user, the first 7 columns of indices, dates and windows are not expected to be useful predictors so we will remove them. Then, for the training to work, classe needs to be a factor. These transformations are processed below.

```r
library(caret)
pmltrain<-subset(pmltrain,new_window=="no")
dim(pmltrain)
```

```
## [1] 19216   160
```

```r
no_na_empty<-function(x){r<-TRUE
    for (y in x){if (y=="" | is.na(y)){r<-FALSE}}
    r
}
pmltrain<-pmltrain[,apply(pmltrain,2,no_na_empty)]
pmltrain<-pmltrain[,c(2,8:dim(pmltrain)[2])]
dim(pmltrain)
```

```
## [1] 19216    54
```

```
pmlval<-pmlval[,apply(pmlval,2,no_na_empty)]
pmlval<-pmlval[,c(2,8:dim(pmlval)[2])]
dim(pmlval)
```

```
## [1] 20 54
```

```
pmltrain$classe<-factor(pmltrain$classe)
```

Before we use our model to predict the classe for the final test data, we will need to test it so the training data needs to be split into 2 sets of data, one to train the model on and one to test it. One obvious initial way to split this would be to use 4 of the users in the first set and 2 in the second set (pmltrain_user and pmltest_user). However, it will be interesting to see if including who the user is helps us to make predictions as if this works it would imply that the model needs to be trained on each user rather than being able to be used directly on users who were not part of the study. So we also split based on a random 70% in the first set and 30% in the second set (pmltrain_rand and pmltest_rand).

```
pmltrain_user<-pmltrain[pmltrain$user_name %in% names(table(pmltrain$user_name)[1:4]),]
pmltrain_user<-pmltrain_user[,-1]
dim(pmltrain_user)
```

```
## [1] 13336    53
```

```
pmltest_user<-pmltrain[pmltrain$user_name %in% names(table(pmltrain$user_name)[5:6]),]
pmltest_user<-pmltest_user[,-1]
dim(pmltest_user)
```

```
## [1] 5880   53
```

```
set.seed(12345)
inTrain<-createDataPartition(pmltrain$classe,p=0.7,list=FALSE)
pmltrain_rand<-pmltrain[inTrain,]
dim(pmltrain_rand)
```

```
## [1] 13453    54
```

```
pmltest_rand<-pmltrain[-inTrain,]
dim(pmltest_rand)
```

```
## [1] 5763   54
```

Training Models - 1. Decision Tree

Now we can begin to look at the results achieved with some different algorithms. First with rpart for decision trees, showing the confusion matrix against the training set and against the test set as well as doing both of these for the training/testing data split by user and the random split.

```
library(rpart)
library(rattle)
mod_rpart_user<-train(classe~.,data=pmltrain_user,method="rpart")
confusionMatrix(predict(mod_rpart_user,pmltrain_user),pmltrain_user$classe)$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 2253  149    0   42   19
##          B   72 1331   38  279  761
##          C 1037 1159 2114 1150  347
##          D  313  108   74  712  173
##          E   13    0    0    0 1192
```

```
conf_rpart_user<-confusionMatrix(predict(mod_rpart_user,pmltest_user),pmltest_user$classe)
conf_rpart_user$overall[1];conf_rpart_user$table
```

```
##  Accuracy
## 0.2231293
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A   82  295  102   23   14
##          B  525  188  382  427   59
##          C 1157  487  642  511  566
##          D   19    1    0    3    0
##          E    0    0    0    0  397
```

```
mod_rpart_rand<-train(classe~.,data=pmltrain_rand,method="rpart")
confusionMatrix(predict(mod_rpart_rand,pmltrain_rand),pmltrain_rand$classe)$table
```

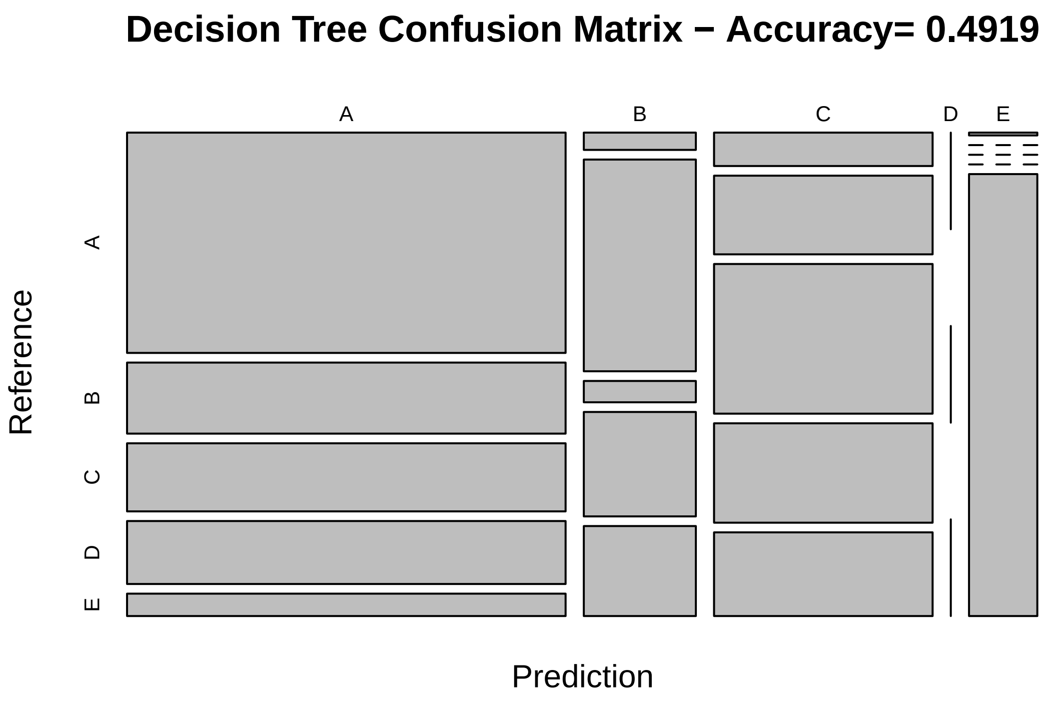
```
##           Reference
## Prediction    A    B    C    D    E
##          A 3482 1063 1087  990  363
##          B   56  899   72  379  324
##          C  282  641 1188  834  661
##          D    0    0    0    0    0
##          E   10    0    0    0 1122
```

```
conf_rpart_rand<-confusionMatrix(predict(mod_rpart_rand,pmltest_rand),pmltest_rand$classe)
conf_rpart_rand$overall[1];conf_rpart_rand$table
```

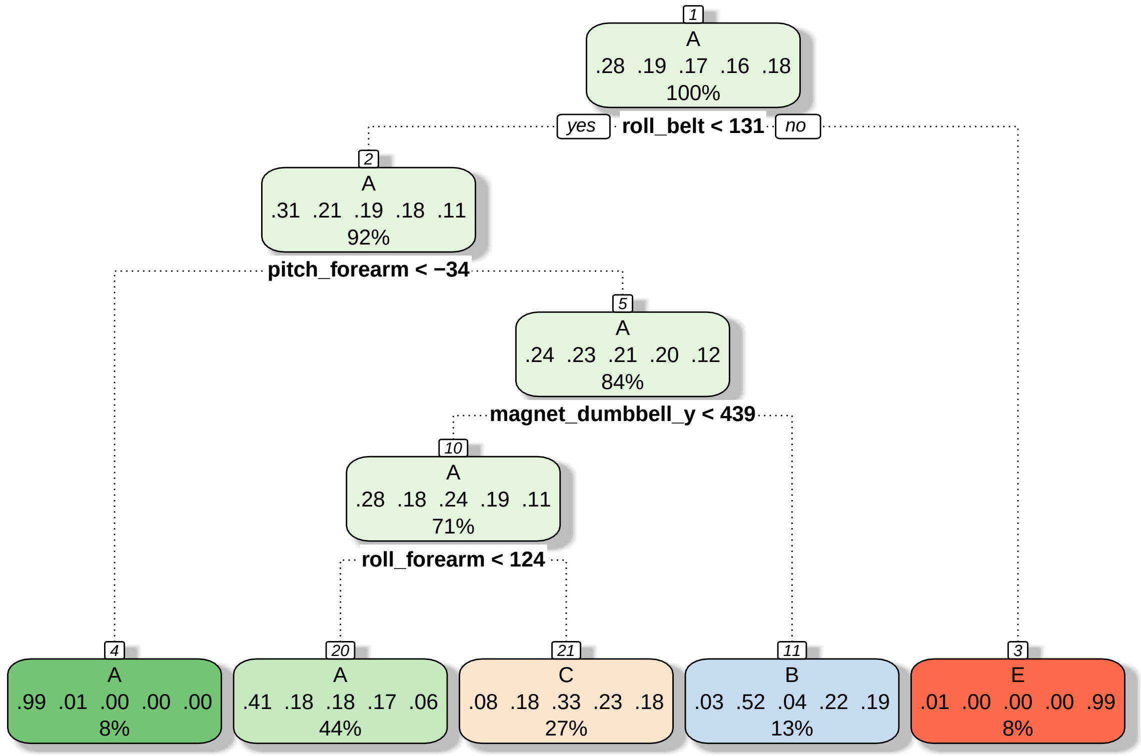
```
##  Accuracy
## 0.4919313
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1495  482  462  427  152
##          B   30  367   37  181  156
##          C  113  266  506  336  283
##          D    0    0    0    0    0
##          E    3    0    0    0  467
```

```
plot(conf_rpart_rand$table,col=conf_rpart_rand$byclass,main=paste("Decision Tree Confusion Matrix - Accu
```



```
fancyRpartPlot(mod_rpart_rand$finalModel)
```



Rattle 2020−Dec−14 13:36:51 xtc_e

This model is not performing well at all.The decision tree for the training set split by user predicts mostly C while the decision tree for the training set determined randomly does not predict D at all.

Training Models - 2. Random Forest

However, it seems possible that moving to a random forest would enable the decisions to be more accurate, at the cost of a more time-consuming training process.

```
mod_rf_user<-train(classe~.,data=pmltrain_user,method="rf")
mod_rf_user$finalModel$confusion
```

```
##      A    B    C    D    E  class.error
## A 3687    1    0    0    0 0.0002711497
## B    7 2738    2    0    0 0.0032763014
## C    0   10 2214    2    0 0.0053908356
## D    0    0   26 2155    2 0.0128263857
## E    0    0    0    2 2490 0.0008025682
```

```
conf_rf_user<-confusionMatrix(predict(mod_rf_user,pmltest_user),pmltest_user$classe)
conf_rf_user$overall[1];conf_rf_user$table
```

```
## Accuracy
## 0.382483
```

```
##           Reference
## Prediction   A   B   C   D   E
##          A 900 134 154   6  22
##          B 403 407 434 438 148
##           C   0  11 111  20  12
##           D   0   0   0   4  27
##           E 480 419 427 496 827
```

```
mod_rf_rand<-train(classe~.,data=pmltrain_rand,method="rf")
mod_rf_rand$finalModel$confusion
```

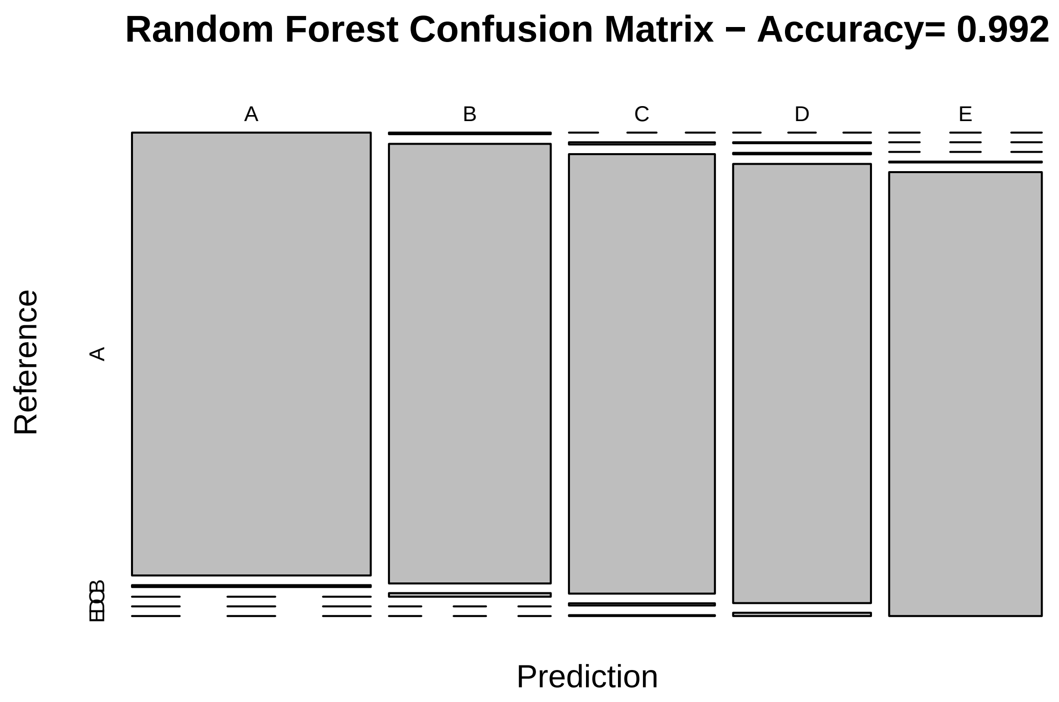
```
##      A    B    C    D    E class.error
## A 3824    4    1    0    1 0.001566580
## B   19 2575    7    1    1 0.010756819
## C    0    9 2328   10    0 0.008095441
## D    0    0   22 2180    1 0.010440309
## E    0    2    5    7 2456 0.005668016
```

```
conf_rf_rand<-confusionMatrix(predict(mod_rf_rand,pmltest_rand),pmltest_rand$classe)
conf_rf_rand$overall[1];conf_rf_rand$table
```

```
## Accuracy
## 0.992018
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1637    7    0    0    0
##          B    4 1101    9    0    0
##          C    0    5  993    5    2
##          D    0    2    3  937    7
##          E    0    0    0    2 1049
```

```
plot(conf_rf_rand$table,col=conf_rf_rand$byclass,main=paste("Random Forest Confusion Matrix - Accuracy=
```



This shows that the random forest method fits very well to the training set. However, on the testing set it performs much better on the random split (99% accuracy to 38%) which implies that there is something about including the individual users in the predictors which helps the training algorithm. It also takes around 40 minutes to run on my computer which is not quick for ‘only’ 13,000 observations in the training set.

Training Models - 2. Random Forest with PCA

Model-based predictions are not helpful, probably because the variables are highly dependent on each other. However, one way of improving the fit while improving the performance might be to use a PCA analysis in the pre-processing. Below, only the results for the training set split randomly are shown.

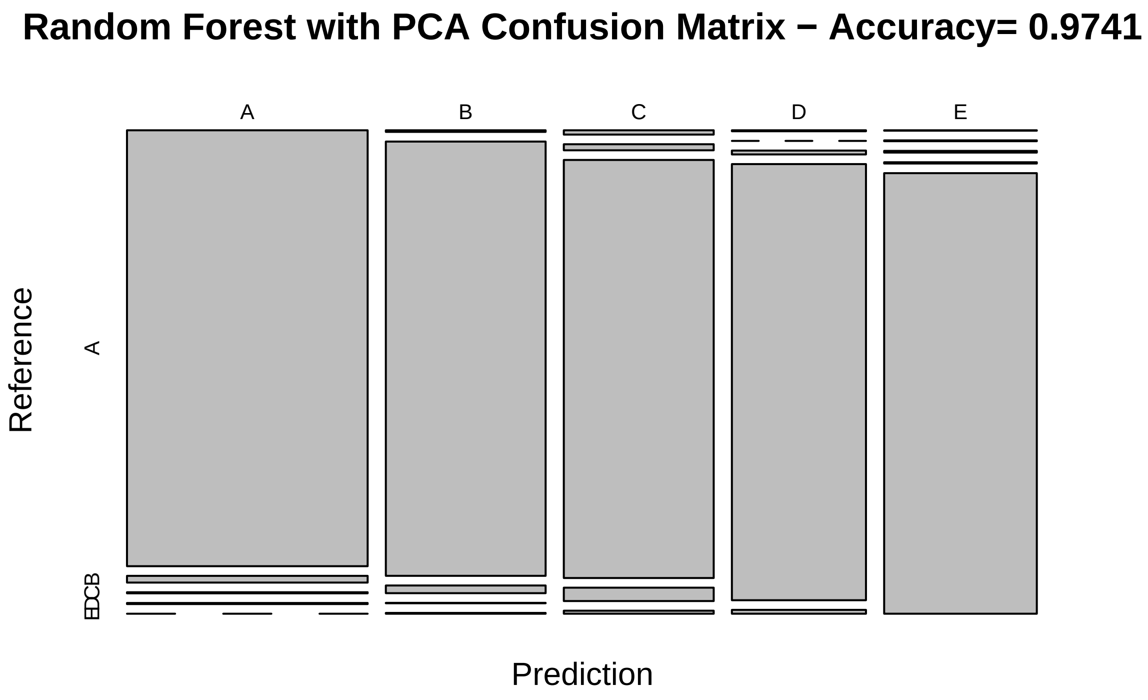
```
library(dplyr)
ctrlRF<-trainControl(method="cv",allowParallel=T)
mod_pca_rand<-train(classe~.,data=pmltrain_rand,method="rf",preProcess="pca",trControl=ctrlRF,tuneGrid=
mod_pca_rand$finalModel$confusion
```

```
##      A    B    C    D    E class.error
## A 3807   11    8    2    2 0.006005222
## B   44 2514   35    2    8 0.034191318
## C    8   33 2283   22    1 0.027268854
## D    1    5   78 2115    4 0.039945529
## E    1   12   12   12 2433 0.014979757
```

```
conf_pca_rand<-confusionMatrix(predict(mod_pca_rand,pmltest_rand),pmltest_rand$classe)
conf_pca_rand$overall[1];conf_pca_rand$table
```

```
##  Accuracy
## 0.9741454
##             Reference
## Prediction    A    B    C    D    E
##          A 1624   25    4    4    0
##          B    4 1074   19    1    2
##          C   10   14  970   31    7
##          D    2    0    8  905    8
##          E    1    2    4    3 1041
```

```
plot(conf_pca_rand$table,col=conf_pca_rand$byclass,main=paste("Random Forest with PCA Confusion Matrix
```



This is a little less accurate at 97% on the random split but is much faster, taking only around 2 minutes to run on my computer.

Conclusion

There are many further tests that could be made such as increasing the threshold in the PCA (from the default of 0.95) to allow it to include more principal components which should increase accuracy but slow the process down or trying Generalised Booster Models.

Based on the results above, it could be better to use the PCA pre-process in situations where the training needs to be done frequently. However, for the validation data we will use the Random Forest model on the training data chosen randomly since it is the most accurate with an expected error rate of just under 1%. The results below will be used to answer the Course Project Prediction Quiz.

```
predict(mod_rf_rand,pmlval)
```

```
##  [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
```