Machine Learning Course Project
How Well Are You Exercising?: (Tracking Physical Movement To Determine if Exercise Is Being Performed Effectively)

Executive Summary

Using data from accelerometers, the physical activity of 6 participants were analyzed to determine whether they performed certain physical activities in the correct manner. To answer this question, various models were fitted on data from the Human Activity Recognition (HAR) dataset.

The data was first split into a training set and a testing set. The training data was further split into two parts. The data were cleaned to remove variables that were of no significance to the model and those that provided little prediction value in terms of variability. After cleaning the data, the training dataset (and the testing set) consisted of 52 predictor variables.

A decision tree and a random forest model were tested on the training data. The decision tree yielded an accuracy reading of 0.49, while the random forest yielded an accuracy reading of 0.99. Because the random forest yielded a higher accuracy reading, it was selected for the final model to run the testing set on. Results from the final model are below.

Response Variable Key

A = exactly according to the specification

B = throwing the elbows to the front

c = lifting the dumbbell only halfway

D = lowering the dumbbells only halfway

E = throwing the hips to the front

```
wd<-setwd("C:/Users/c ford/Documents/Machine Learning/finalproject")
setwd("C:/Users/c ford/Documents/Machine Learning/finalproject")

library(corrplot)
library(caret)
library(ggplot2)
library(rattle)
library(e1071)
library(randomForest)
```

Data Download and Cleaning

First, I downloaded the training and testing sets from their respective websites, and I read the CSV datasets into R.

```
#training
if(!file.exists("training.csv"))
  { download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv","trai
ning.csv")}

#testing
if(!file.exists("testing.csv"))
{download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv","testing.
csv")}

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

In this piece of code, I removed variables with missing values and those that are meaningless in terms of providing predictive value to the model. I also rename the testing variable from ‘problem_id’ to ‘classe’ so that it matches with the training set variable name.

```
#testing subset
na<-is.na(testing[1,])
testing1<-testing[,!na]

#training subset
training1<-training[training$new_window=="no",]
training1<-training1[,!na]

#Remove variables you don't want in the model (those with meaningless prediction value, at lea
st for this model)
training1<-training1[,-c(1:7)]
testing1<-testing1[,-c(1:7)]

#rename testing variable
testing1$classe<-testing1$problem_id
testing1<-testing1[,-c(53)]
```

I also use the nearzero function to remove variables without any variability as they lack predictive value (none were actually removed but it is important to check for this).

```
nearzero<-as.data.frame(nearZeroVar(training1,saveMetrics = TRUE))
nearzero<-nearzero[nearzero$nzv=="FALSE",]
nearzero<-rownames(nearzero)
training1<-training1[,nearzero]
testing1<-testing1[,nearzero]
```

Correlation Matrix

I also ran a correlation matrix on the training set data to get a better sense of which variables are linearly correlated with each other. Usually this process can lead to the removal of variables with high collinearity or to the creation of a new variable that captures the variability of multiple collinear variables (principal components analysis), but none of those actions were taken.

Analysis of the correlation matrix shows that a substantial share of the variable pairs - 2532 - are not well correlated (under 0.5), and 44 pairs are highly correlated.

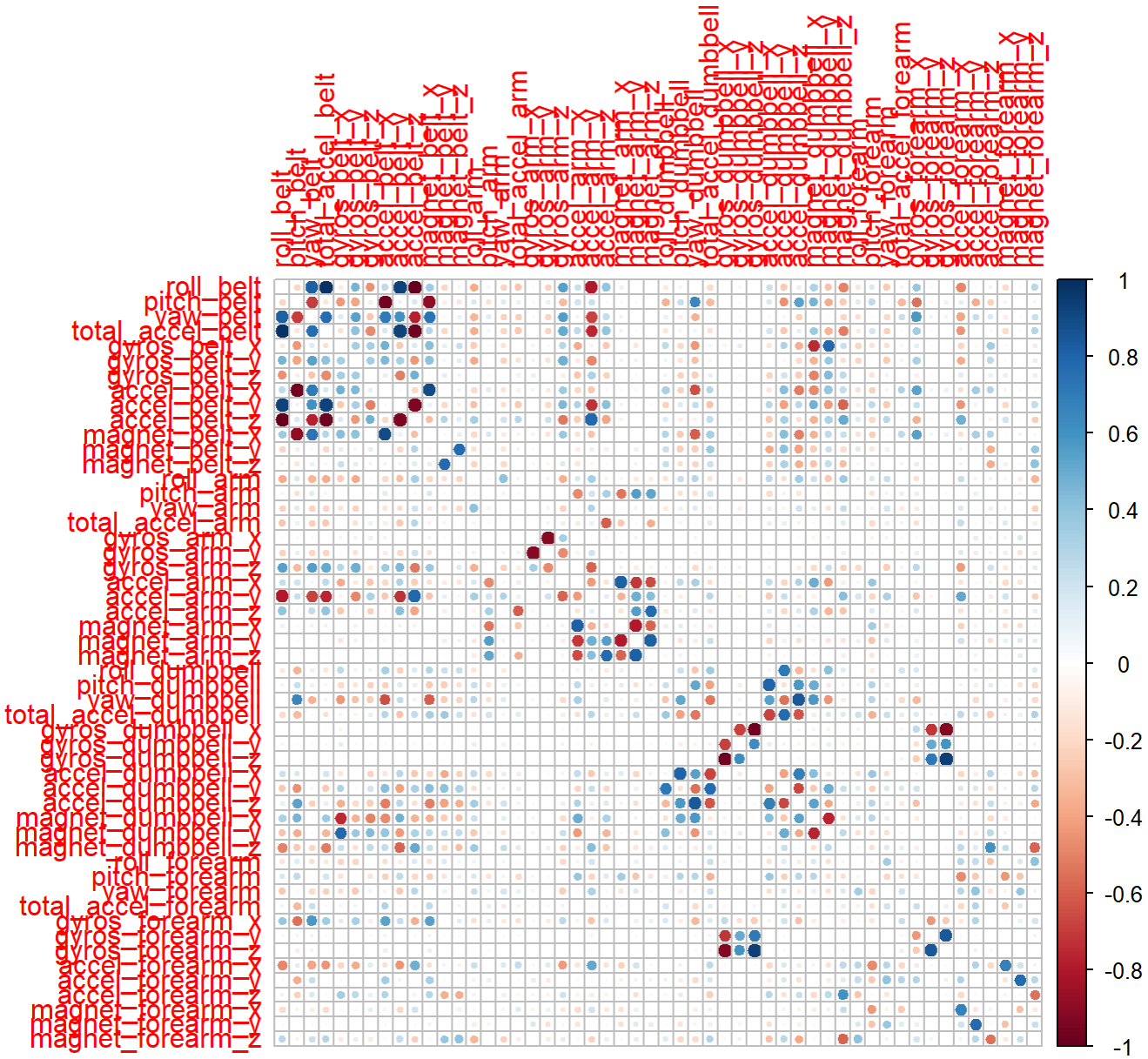
```
m<-cor(training1[,c(-53)])
diag(m)<-0
high<-length(which(m >= 0.800000000001,arr.ind=TRUE))
low<-length(which(m > 0.000000000 & m < 0.5000000000001,arr.ind=TRUE))
high
```

```
## [1] 44
```

```
low
```

```
## [1] 2532
```

```
corrplot(m)
```



Splitting Training Set

Next, I split the training set into two parts to run my models on.

```
intraining1<-createDataPartition(training1$classe,p=0.5,list=FALSE)
train1<-training1[intraining1,]
train2<-training1[-intraining1,]
```

Model Fit One: Classification Tree

A basic classification tree model using the method="rpart" model is used to fit the data. A decision tree was chosen for its interpretability and for its ability to partition data into many spaces that aren't necessarily linearly related. This is in contrast to a logistic-based model which only allows for one linear decision boundary.

A classification tree was used (rather than a regression tree) because the outcome, 'classe', is discrete, not continuous. The drawbacks of a decision tree is that it may lead to overfitting. But cross validation is also used to help control for overfitting.

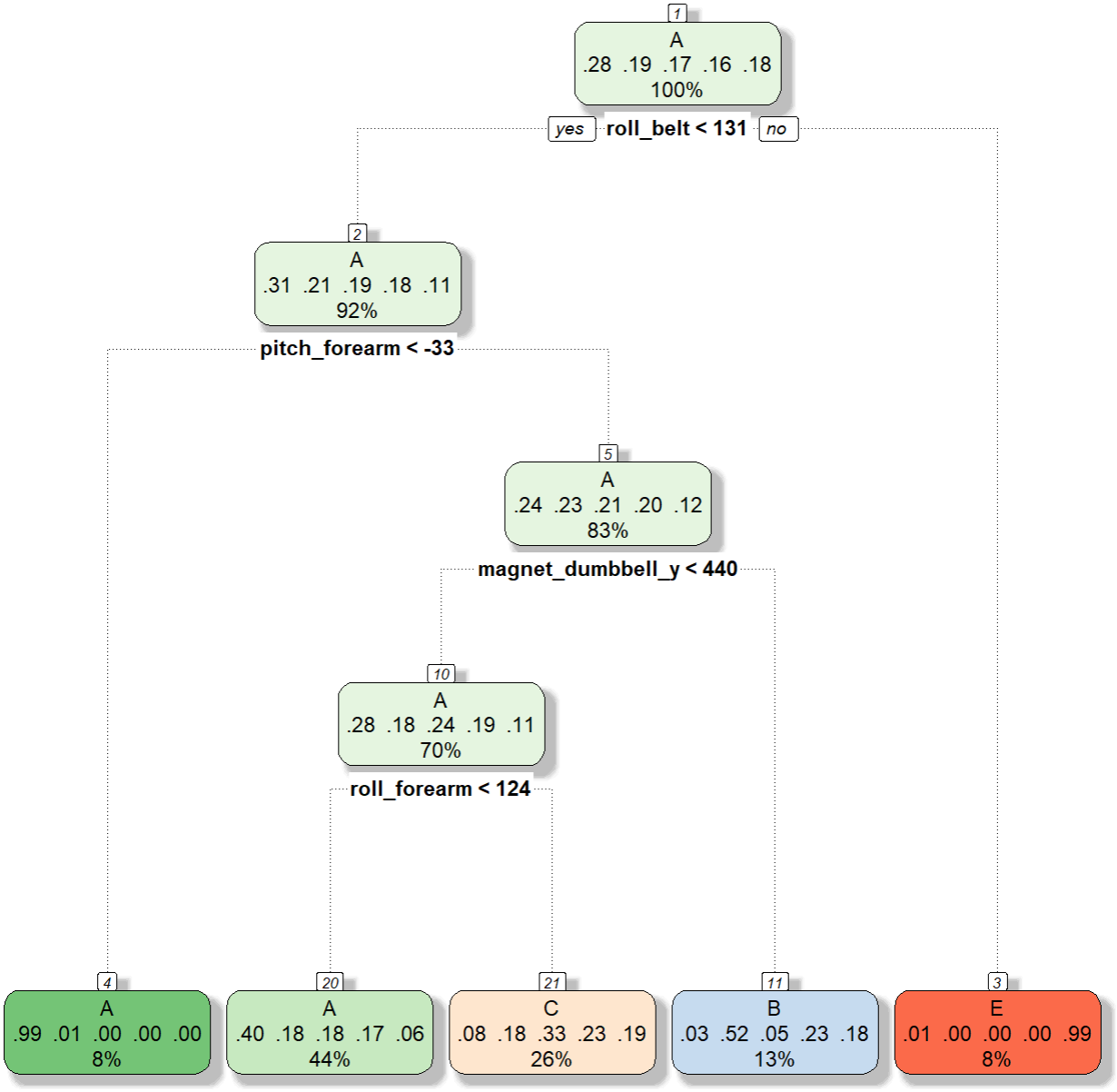
Results for this model show an accuracy rate of 0.49, which isn't vary high. We will explore other models.

```
set.seed(1965)
```

```
tree_fit<-train(classe ~ . , data=train1, method="rpart", trControl=trainControl(method="cv",n
umber=10))
print(tree_fit)
```

```
## CART
##
## 9609 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8646, 8648, 8650, 8649, 8648, 8648, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03360978  0.5098416  0.35973525
##   0.06101169  0.4543651  0.27199062
##   0.11537902  0.3483105  0.09715556
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03360978.
```

```
fancyRpartPlot(tree_fit$finalModel)
```



Rattle 2020-Sep-05 08:53:17 c ford

```
tree_fit1<-predict(tree_fit, newdata=train2)
confusionMatrix(tree_fit1,as.factor(train2$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2481  789  773  715  253
##          B   49  620   51  272  249
##          C  199  450  852  586  473
##          D    0    0    0    0    0
##          E    6    0    0    0  789
##
## Overall Statistics
##
```

```
##                Accuracy : 0.4936
##                  95% CI : (0.4836, 0.5036)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3379
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9071  0.33351  0.50835   0.0000  0.44728
## Specificity            0.6318  0.91985  0.78464   1.0000  0.99923
## Pos Pred Value         0.4951  0.49960  0.33281      NaN  0.99245
## Neg Pred Value         0.9447  0.85190  0.88307   0.8363  0.88936
## Prevalence             0.2847  0.19350  0.17446   0.1637  0.18362
## Detection Rate         0.2582  0.06454  0.08869   0.0000  0.08213
## Detection Prevalence   0.5216  0.12918  0.26647   0.0000  0.08275
## Balanced Accuracy      0.7695  0.62668  0.64650   0.5000  0.72326
```

Model Two: Random Forest Tree

A random forest model was used on the data. The advantage of a random forest is that it can provide improved accuracy over a basic decision tree or a bagged tree as it bootstraps the data and variables across a large number of trees and averages the results. The drawback often cited with random forest models is the overfitting risk. Cross validation is incorporated in the model to help control for overfitting (k-fold=10).

The random forest model results show a 0.99 accuracy reading. For this reason, the random forest will be used to predict the classe variable in the testing set.

```
rf_fit<-train(classe ~ . , data=train1, method="rf", trControl=trainControl(method="cv",number
=10))
print(rf_fit)
```

```
## Random Forest
##
## 9609 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8647, 8649, 8649, 8649, 8646, 8649, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9837627  0.9794541
##   27    0.9861552  0.9824818
##   52    0.9785572  0.9728647
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
rf_fit1<-predict(rf_fit, newdata=train2)
```

```
confusionMatrix(rf_fit1,as.factor(train2$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2729   22    0    0    0
##          B    5 1825   10    2    0
##          C    1   12 1659   14    0
##          D    0    0    7 1557    6
##          E    0    0    0    0 1758
##
## Overall Statistics
##
##                Accuracy : 0.9918
##                  95% CI : (0.9898, 0.9935)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9896
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9978   0.9817   0.9899   0.9898   0.9966
## Specificity            0.9968   0.9978   0.9966   0.9984   1.0000
## Pos Pred Value         0.9920   0.9908   0.9840   0.9917   1.0000
## Neg Pred Value         0.9991   0.9956   0.9979   0.9980   0.9992
## Prevalence             0.2847   0.1935   0.1745   0.1637   0.1836
## Detection Rate         0.2841   0.1900   0.1727   0.1621   0.1830
## Detection Prevalence   0.2864   0.1917   0.1755   0.1634   0.1830
## Balanced Accuracy      0.9973   0.9898   0.9932   0.9941   0.9983
```

Random Forest Model Prediction on Testing Data

The random forest model is used on the testing data and the “classe” variable results are stored in the variable modelfittest.

```
modelfittest<-predict(rf_fit, newdata=testing1)
modelfittest
```

```
##  [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
```

Concluding Remarks

The classification decision tree produced an accuracy of 0.49, therefore the out-of-sample error for this model is approximately 0.51. The random forest out-of-sample error is much lower at approximately 0.01 (with an accuracy rate of .99). Although the random forest model produces a low out-of-sample error for the training set, it will be higher for the testing set, due to overfitting.