Predicting the Quality of Exercise Activities

Vin Lam

The goal of this analysis is to use data generated by accelerometers to predict the quality of how various exercise activities were performed among 6 test subjects. A wide variety of measurements were recorded, as well as the manner in which the activity was performed within the training set (the "classe" column). First, the data will be cleaned by fixing missing values and converting variables into the correct classes. Then, data will be preprocessed using the caret package, first with PCA and then removing predictors with near zero variance. I will experiment with three different models and assess the performance of each model on the validation set before picking the best model to predict on the test set.

First things first, we load the necessary libraries before beginning.

```r
library(caret);library(tidyverse);library(parallel);library(doParallel)
```

Reading in the data locally, then removing unneeded columns such as username and timestamps.

```r
train = read.csv("pml-training.csv"); train = as_tibble(train)
test = read.csv("pml-testing.csv"); test = as_tibble(test)

train = train[,-c(1:7)]; test = test[,-c(1:7)]
```

Looping through each column of the training set and converting factor variables that should be numeric. Storing the classe variable for future use, since it is the only variable that should be a factor.

```r
classe = train$classe

train = train %>% mutate_if(is.factor, as.character)
train = train %>% mutate_if(is.character, as.numeric)
```

Now we need to check for missing values. We can easily check each column and return the sum of na values from each column. Then, if data is missing for that column in more than 90% of observations, that column will be omitted from the dataframe. Afterwards, the classe variable in its proper factor form is re-bound to the dataframe.

```r
head(colSums(is.na(train)),10);head(colSums(is.na(test)),10)
```

```
##            roll_belt           pitch_belt             yaw_belt 
##                    0                    0                    0 
##     total_accel_belt   kurtosis_roll_belt  kurtosis_picth_belt 
##                    0                19226                19248 
##    kurtosis_yaw_belt   skewness_roll_belt skewness_roll_belt.1 
##                19622                19225                19248 
##    skewness_yaw_belt 
##                19622
```

```
##            roll_belt           pitch_belt             yaw_belt 
##                    0                    0                    0 
##     total_accel_belt   kurtosis_roll_belt  kurtosis_picth_belt 
##                    0                   20                   20 
##    kurtosis_yaw_belt   skewness_roll_belt skewness_roll_belt.1 
##                   20                   20                   20 
##    skewness_yaw_belt 
##                   20
```

```
train = train[,which(colMeans(!is.na(train)) > 0.9)]; test = test[,which(colMeans(!is.na(test)) > 0.9)]

train = cbind(train,classe); train = as_tibble(train)
```

We are left with 52 predictors after removing columns with over 90% of values missing. The next step is to split the training set into a training and validation set to build our models with.

```
set.seed(69420)
inTrain = createDataPartition(y = train$classe,p=0.7,list=FALSE)
training = train[inTrain,]
testing = train[-inTrain,]
```

The first model I would like to train is a simple random forest model on only the principal components of the training data. The reason for this is that the crossvalidation method I want to use is repeated k-folds cross validation. This is a very lengthy process, especially with many variables, but it boosts model performance quite significantly. Luckily, we can take advantage of dimension reduction via principal components, which will make training a random forest model much quicker. The main disadvantage with this approach is reduced accuracy, but it will provide great insight on the performance of random forest before actually investing the time to train a rf model on the entire training set. We preprocess the training and validation data and create new dataframes containing the principal components, but making sure to use the preprocessed object from the training data to form the new validation dataframe.

```
preObj = preProcess(training[,-53],method="pca")
trainingProc = predict(preObj,training[,-53])
testingProc = predict(preObj,testing[,-53])
```

Now, we can train our first model. I take advantage of the allowParallel parameter in the train function to use more of my PC's computing power. I specify repeated k-folds CV as my method of CV, with 10 folds and 5 repeats. 10 is chosen since it balances the bias versus variance trade-off very well.

```
cluster = makeCluster(detectCores()-1)
registerDoParallel(cluster)

tControl = trainControl(method = "repeatedcv",
                        number = 10,
                        repeats = 5,
                        allowParallel = TRUE)

model1 = train(x = trainingProc, y = training$classe, method = "rf", trControl = tControl)

pred1 = predict(model1, newdata=testingProc)

confusionMatrix(pred1, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1662   17    1    1    0
##          B    1 1100   11    0    8
##          C    8   20  999   44    6
##          D    3    0   12  916    8
##          E    0    2    3    3 1060
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9749
##                  95% CI : (0.9705, 0.9787)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9682
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9928   0.9658   0.9737   0.9502   0.9797
## Specificity            0.9955   0.9958   0.9839   0.9953   0.9983
## Pos Pred Value         0.9887   0.9821   0.9276   0.9755   0.9925
## Neg Pred Value         0.9971   0.9918   0.9944   0.9903   0.9954
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2824   0.1869   0.1698   0.1556   0.1801
## Detection Prevalence   0.2856   0.1903   0.1830   0.1596   0.1815
## Balanced Accuracy      0.9942   0.9808   0.9788   0.9728   0.9890
```

The model does very well despite using only PCA. The accuracy is over 97%, suggesting that random forest is a good candidate for prediction and that it would be worth training a model on every variable in the dataset. With this in mind, gradient boosting might be another good candidate taking into consideration the performance of the previous model. Both are ensemble methods using classification trees so boosting is worth trying out. I will use new preprocessed dataframes, this time eliminating near zero variance predictors from our training set.

```r
preObj2 = preProcess(training[,-53], method = "nzv")
trainingProc2 = predict(preObj2,training[,-53])
testingProc2 = predict(preObj2,testing[,-53])

model2 = train(x = trainingProc2, y = training$classe, method = "gbm", trControl = tControl, verbose = FALS

pred2 = predict(model2, newdata=testingProc2)

confusionMatrix(pred2, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1643   51    0    0    2
##          B   22 1054   24    5   14
##          C    4   32  984   35   10
##          D    3    1   17  920   16
##          E    2    1    1    4 1040
##
## Overall Statistics
##
##                Accuracy : 0.9585
##                  95% CI : (0.9531, 0.9635)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9475
##
```

```
##  Mcnemar's Test P-Value : 4.103e-08
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9815   0.9254   0.9591   0.9544   0.9612
## Specificity            0.9874   0.9863   0.9833   0.9925   0.9983
## Pos Pred Value         0.9687   0.9419   0.9239   0.9613   0.9924
## Neg Pred Value         0.9926   0.9822   0.9913   0.9911   0.9913
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2792   0.1791   0.1672   0.1563   0.1767
## Detection Prevalence   0.2882   0.1901   0.1810   0.1626   0.1781
## Balanced Accuracy      0.9844   0.9558   0.9712   0.9734   0.9798
```

While the model performs well, it is weaker than the random forest model trained on only the principal components. The final model selected is a random forest model trained on all 52 predictors.

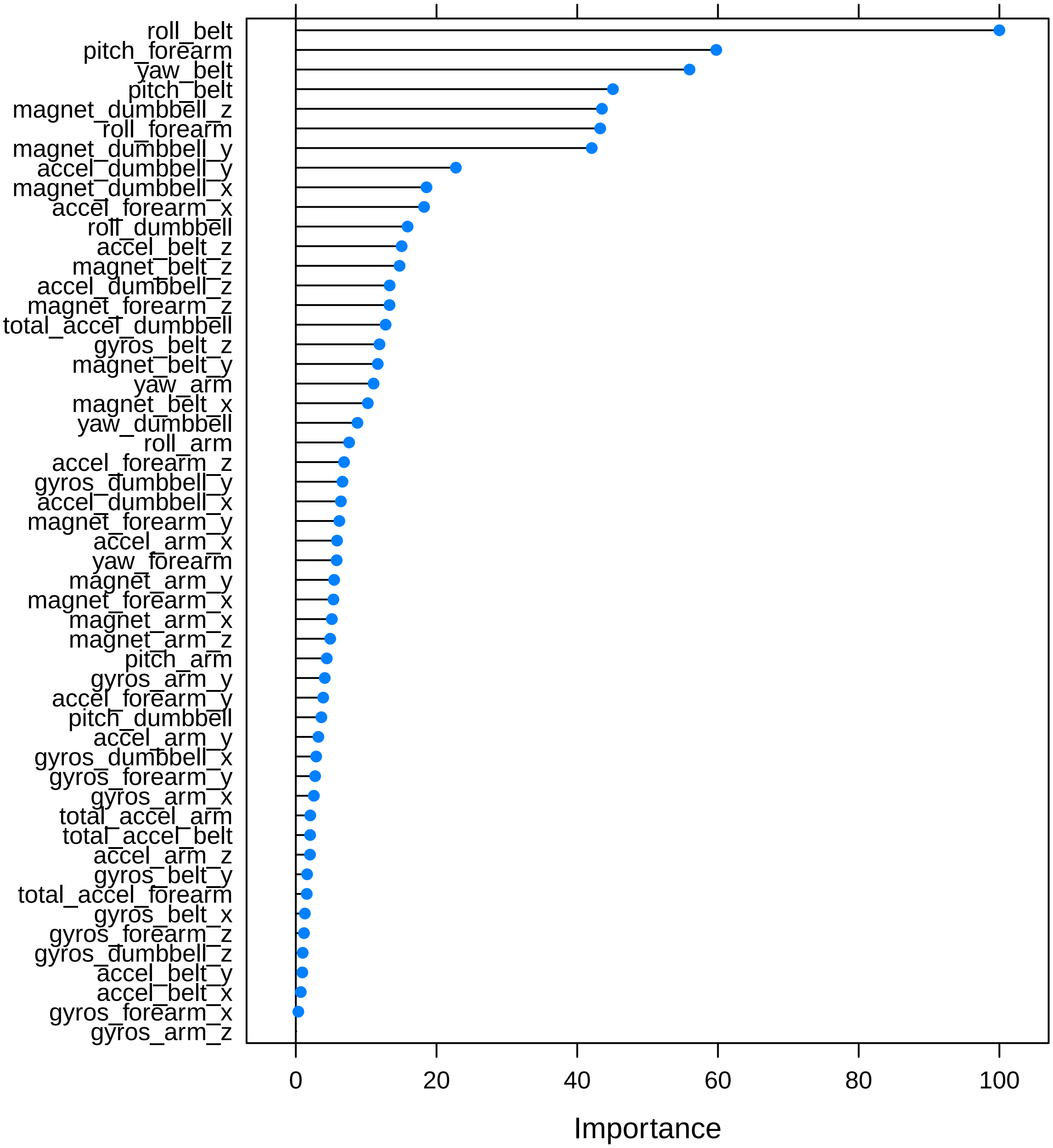
```
model3 = train(x = trainingProc2, y=training$classe, method = "rf", trControl = tControl)

pred3 = predict(model3, newdata=testingProc2)

confusionMatrix(pred3, testing$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1674   14    0    0    0
##          B    0 1122    6    1    1
##          C    0    3 1016   17    3
##          D    0    0    4  944    1
##          E    0    0    0    2 1077
## 
## Overall Statistics
## 
##                Accuracy : 0.9912          
##                  95% CI : (0.9884, 0.9934)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
## 
##                   Kappa : 0.9888          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9851   0.9903   0.9793   0.9954
## Specificity            0.9967   0.9983   0.9953   0.9990   0.9996
## Pos Pred Value         0.9917   0.9929   0.9779   0.9947   0.9981
## Neg Pred Value         1.0000   0.9964   0.9979   0.9959   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1907   0.1726   0.1604   0.1830
## Detection Prevalence   0.2868   0.1920   0.1766   0.1613   0.1833
## Balanced Accuracy      0.9983   0.9917   0.9928   0.9891   0.9975
```

```r
stopCluster(cluster)
registerDoSEQ()
```

The final model has the best in sample accuracy out of the 3 models, boasting over 99%. The importance of each predictor to the algorithm can be seen here.

```r
plot(varImp(model3))
```



With the lower bound of the 95% CI for the accuracy of the third model being ~99%, I would expect the out-of-sample error to be around 1%. At this point, we might try to pursue marginal improvements in the model through other means such as combining predictors, hypertuning parameters, and experimenting with more powerful/computationally demanding algorithms like XGBoost. However, I don't think it is necessary as the random forest model trained on our near-zero-variance preprocessed data performs exceptionally.

Lastly, we can make predictions on the test set.

```
testpred = predict(model3, test)
testpred
```

```
##  [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
```