Machine Learning Course Project - Prediction Assignment

J-Besna

March 29, 2020

Part 1: Project Overview

To run this analysis, download the csv files in this repository. Set your working directory to the folder with the csv file and you will be able to run this analysis.The data for this project come from this source: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

The data consists of accelerometer data worn on the belt, arm, forearm, and dumbell of 6 participants. The participants were asked to perform barbell lifts correctly and incorrectly in five different ways. This analysis predicts the how the exercise performed (the "classe" field) based on independent variables. In the "classe" field, A represents an exercise performed correctly while B-E each represent a specific incorrect performance of the exercise.

The objective is to evaluate several machine learning models using the training data and use the best model to predict values for the 20 observations in the testing data. This exercise will evaluate 3 types of machine learning models: Decision Tree, Random Forest, and Gradient Boosting.

Part 2: Data Import and Initial Data Cleaning

Taking a quick overview of the data using str() function, we can see there are 160 variables over 19,622 observations. We will need to remove as many insignicant variables as possible to be able to run some classification algorithms. Immediately we can remove the following columns that add no modeling value: X, user_name, kurtosis_yaw_belt, skewness_yaw_belt and all the timestamp columns.

Part 3: Eliminating Unncessary Variables

We still have 153 columns with over 19,000 observations, which is too many to run ML algorithms. To run the model we will check the column contents to see if any can be removed using the summary() function, which reveals that many columns contain nearly all NA values. These can all be eliminated. Additionally, we check to see which variables have high correlation and eliminate these variables as well since they will not add modeling value.

```
# run a summary to get a feel for the distribution of values             within the variables
summary(dat.train)
# we see a good number of columns that have 19,216 NA values,            adding little value an
drop_cols <- which(colSums(dat.train=="" | is.na(dat.train))>0.95*dim(dat.train)[1])
str(drop_cols)
# we end up deleting 31 columns
dat.train <- dat.train[ ,-drop_cols]
# this leaves 122 columns
ncol(dat.train)
# next we eliminate redunant variables by checking for variables
# that are correlated with one another by at least 75% (source: https://machinelearningmastery.
# get numeric columns
dat.train.nums <- select_if(dat.train,is.numeric)
# calculate correlation matrix
correlationMatrix <- cor(dat.train.nums)
# summarize the correlation matrix
```

```
        print(correlationMatrix)
        # find attributes that are highly corrected (ideally >0.75)
        highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75)
        # remove highly correlated variables
        dat.train <- dat.train[,-highlyCorrelated]
        nrow(dat.train)
```

Part 4: Partition the data into training data and testing data using a 50/50 split. While a 70/30 split is more common, we will use 50/50 to conserve runtime and accomodate local processing limitations. We then verify that the distribution of classe dependent variable is similar across the training and testing data.

```
        # partition data into train/test. Normally would use 70/30 split,          but going with 50/50
        inTrain <- createDataPartition(y=dat.train$classe,
                                       p=0.5, list=FALSE)
        training <- dat.train[inTrain,]
        testing <- dat.train[-inTrain,]
        # next we to verify that the distribution of the classe variable          is similar in training
        table(training$classe)
```

```
##
##    A    B    C    D    E
## 2790 1899 1711 1608 1804
```

```
        table(testing$classe)
```

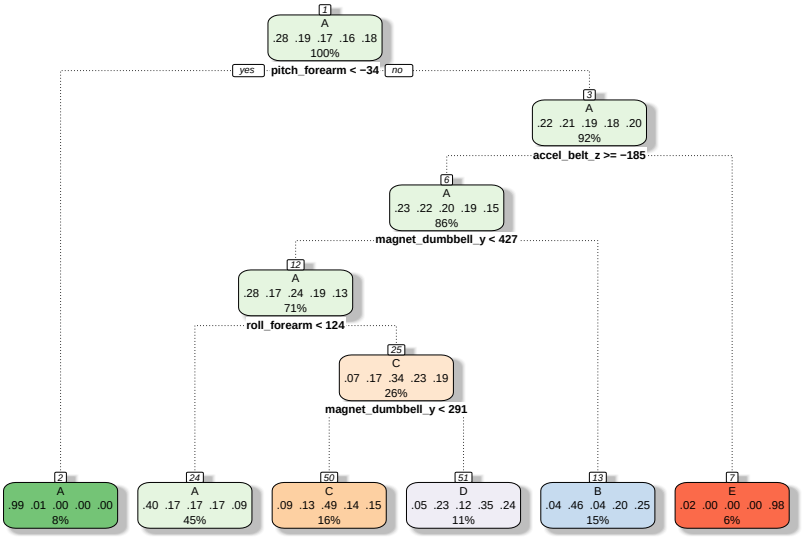
```
##
##    A    B    C    D    E
## 2790 1898 1711 1608 1803
```

Part 5: Run a decision tree model.

```
        # decision tree example
        mod_rpart <- train(classe ~ .,method="rpart",data=training)
        print(mod_rpart$finalModel)
```

```
## n= 9812
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 9812 7022 A (0.28 0.19 0.17 0.16 0.18)
##    2) pitch_forearm< -33.95 802    8 A (0.99 0.01 0 0 0) *
##    3) pitch_forearm>=-33.95 9010 7014 A (0.22 0.21 0.19 0.18 0.2)
##      6) accel_belt_z>=-185.5 8457 6472 A (0.23 0.22 0.2 0.19 0.15)
##       12) magnet_dumbbell_y< 426.5 6968 5044 A (0.28 0.17 0.24 0.19 0.13)
##         24) roll_forearm< 123.5 4379 2646 A (0.4 0.17 0.17 0.17 0.09) *
##         25) roll_forearm>=123.5 2589 1706 C (0.074 0.17 0.34 0.23 0.19)
##           50) magnet_dumbbell_y< 290.5 1524  771 C (0.087 0.13 0.49 0.14 0.15) *
##           51) magnet_dumbbell_y>=290.5 1065  692 D (0.054 0.23 0.12 0.35 0.24) *
##       13) magnet_dumbbell_y>=426.5 1489  801 B (0.041 0.46 0.044 0.2 0.25) *
##      7) accel_belt_z< -185.5 553   11 E (0.02 0 0 0 0.98) *
```

```
fancyRpartPlot(mod_rpart$finalModel)
```



Rattle 2020−Mar−30 18:19:58 johnmara

```
# Predict and create confusion matrix
pred_cm <- predict(mod_rpart,newdata=testing)
cm_rpart <- confusionMatrix(as.factor(testing$classe),pred_cm)
cm_rpart
```

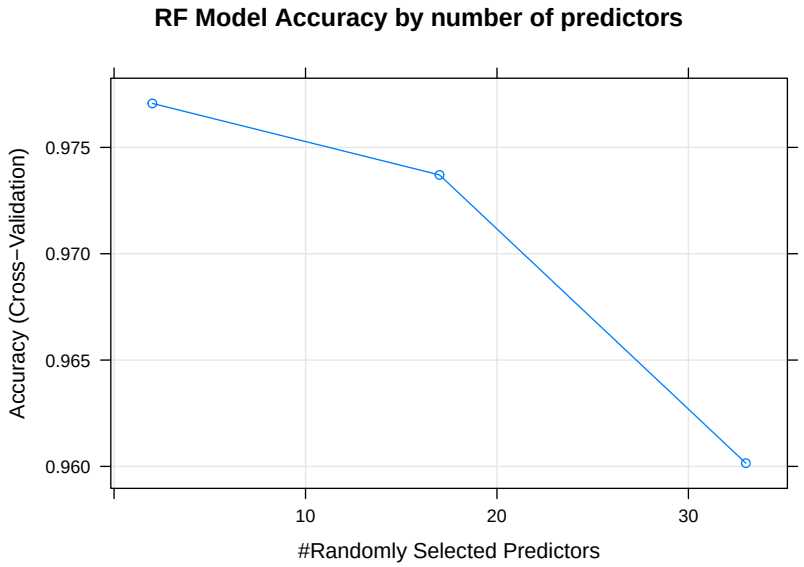
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2501   62  149   56   22
##          B  762  660  214  261    1
##          C  807   63  725  116    0
##          D  698  318  239  353    0
##          E  403  388  236  260  516
##
## Overall Statistics
##
##                Accuracy : 0.4847
##                  95% CI : (0.4748, 0.4947)
##     No Information Rate : 0.5271
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3267
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4837  0.44266   0.4639  0.33748  0.95733
## Specificity            0.9377  0.85118   0.8804  0.85680  0.86118
## Pos Pred Value         0.8964  0.34773   0.4237  0.21953  0.28619
## Neg Pred Value         0.6197  0.89497   0.8965  0.91551  0.99713
## Prevalence             0.5271  0.15199   0.1593  0.10663  0.05494
```

```
## Detection Rate         0.2549  0.06728   0.0739  0.03598  0.05260
## Detection Prevalence   0.2844  0.19348   0.1744  0.16391  0.18379
## Balanced Accuracy      0.7107  0.64692   0.6721  0.59714  0.90925
```

```
        rpart_accuracy <- cm_rpart$overall['Accuracy']
```

Part 6: Run a random forest model.

```
      # Try random forest
        fitControl <- trainControl(method='cv', number = 3)
        mod_rf <- train(classe~., data=training, method="rf", trControl=fitControl, verbose=FALSE)
        # Plot
        plot(mod_rf,main="RF Model Accuracy by number of predictors")
```



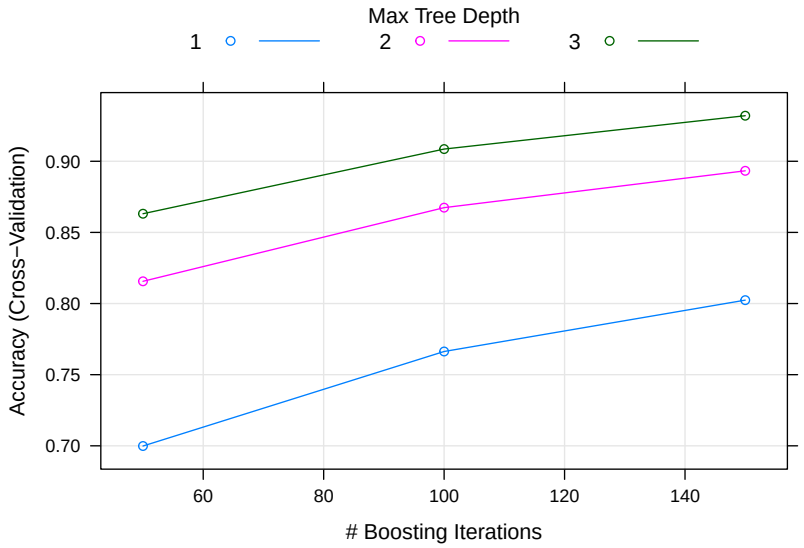

```
        # Testing the model
        pred_rf <- predict(mod_rf,newdata=testing)
        cm_rf <- confusionMatrix(as.factor(testing$classe),pred_rf)
        rf_accuracy <- cm_rf$overall['Accuracy']
        # Display confusion matrix and model accuracy
        cm_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2788    2    0    0    0
##          B   38 1833   26    0    1
##          C    0   29 1679    3    0
##          D    0    0   50 1554    4
##          E    0    4    5    9 1785
##
## Overall Statistics
##
##                Accuracy : 0.9826
##                  95% CI : (0.9798, 0.9851)
##     No Information Rate : 0.2881
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                  Kappa : 0.9779
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9866   0.9813   0.9540   0.9923   0.9972
## Specificity            0.9997   0.9918   0.9960   0.9934   0.9978
## Pos Pred Value         0.9993   0.9658   0.9813   0.9664   0.9900
## Neg Pred Value         0.9946   0.9956   0.9900   0.9985   0.9994
## Prevalence             0.2881   0.1904   0.1794   0.1596   0.1825
## Detection Rate         0.2842   0.1869   0.1712   0.1584   0.1820
## Detection Prevalence   0.2844   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9931   0.9865   0.9750   0.9929   0.9975
```

Part 7: Run a gradient boosting model (GBM).

```
# try gradient boosting model
mod_gbm <- train(classe~., data=training, method="gbm", trControl=fitControl, verbose=FALSE)
#  Plot
plot(mod_gbm)
```




```
# Testing the model
pred_gbm <- predict(mod_gbm,newdata=testing)
cm_gbm <- confusionMatrix(as.factor(testing$classe),pred_gbm)

# Display confusion matrix and model accuracy
cm_gbm
```

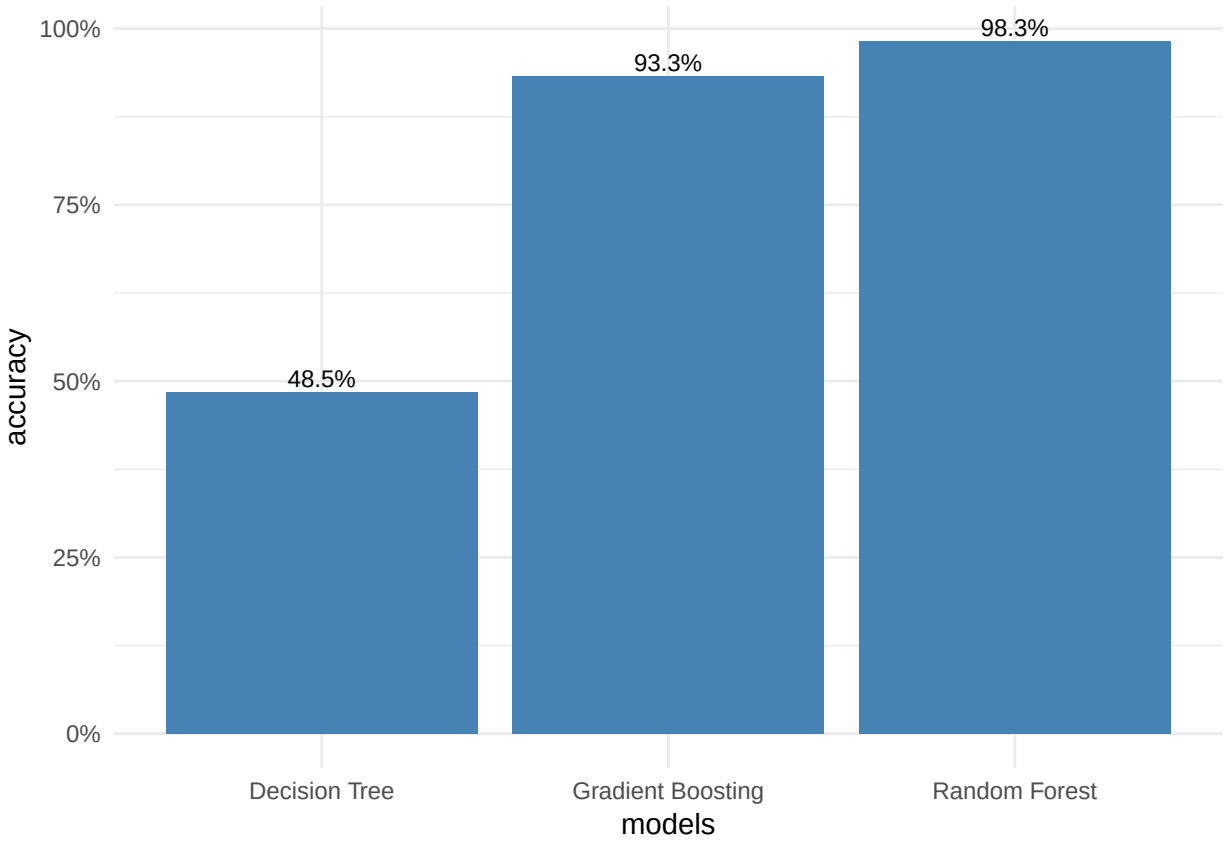
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2724   44    8    7    7
##          B  108 1685   70    3   32
##          C    6   77 1601   24    3
##          D    1    7   96 1484   20
##          E    7   45   63   32 1656
```

```
##
## Overall Statistics
##
##                Accuracy : 0.9327
##                  95% CI : (0.9276, 0.9376)
##     No Information Rate : 0.2901
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9148
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9571   0.9069   0.8711   0.9574   0.9639
## Specificity            0.9905   0.9732   0.9862   0.9850   0.9818
## Pos Pred Value         0.9763   0.8878   0.9357   0.9229   0.9185
## Neg Pred Value         0.9826   0.9781   0.9707   0.9920   0.9923
## Prevalence             0.2901   0.1894   0.1874   0.1580   0.1751
## Detection Rate         0.2777   0.1718   0.1632   0.1513   0.1688
## Detection Prevalence   0.2844   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9738   0.9401   0.9286   0.9712   0.9729
```

```
gbm_accuracy <- cm_gbm$overall['Accuracy']
```

Part 8: Compare the model accuracy across the 3 models (decision tree, random forest, gradient boosting)

```
# Graph acccuracy of models to see which is more accurate
models <- c('Decision Tree','Random Forest','Gradient Boosting')
accuracy <- c(rpart_accuracy,rf_accuracy,gbm_accuracy)
graph.data <- data.frame(models,accuracy)
p<-ggplot(data=graph.data, aes(x=models, y=accuracy)) +
        geom_bar(stat="identity", fill="steelblue")+
        geom_text(aes(label=percent(accuracy)), vjust=-0.3, size=3)+ scale_y_continuous(labels =
        theme_minimal()
p
```



We see that the Random Forest model has the highest accuracy. We then predict the test set of 20 observations using the Random Forest model to get the following predicted results:

```
test_final <- predict(mod_rf,newdata=dat.test)
test_final
```

```
##  [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
```