Practical Machine Learning

Daniel Rimdzius

1/15/2020

Overview

This is the final course project for Johns Hopkins Practical Machine Learning Course. It uses accelerometer data from six participants who were asked to perform barbell lifts correct and incorrectly in five different ways. I will download and clean the data, and train a few models to find the most accurate model, which turns out to be the Random Forest model. That will then be used on the test data and the answers submitted as part of the final quiz.

Data Processing and Exploratory Analysis

I start with loading the data into R, and doing some preprocessing and exploratory analysis

```
require(dplyr)
require(caret)
require(ggplot2)
require(rapportools)
require(rattle)
require(rpart)
training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
testing <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
dim(training)
```

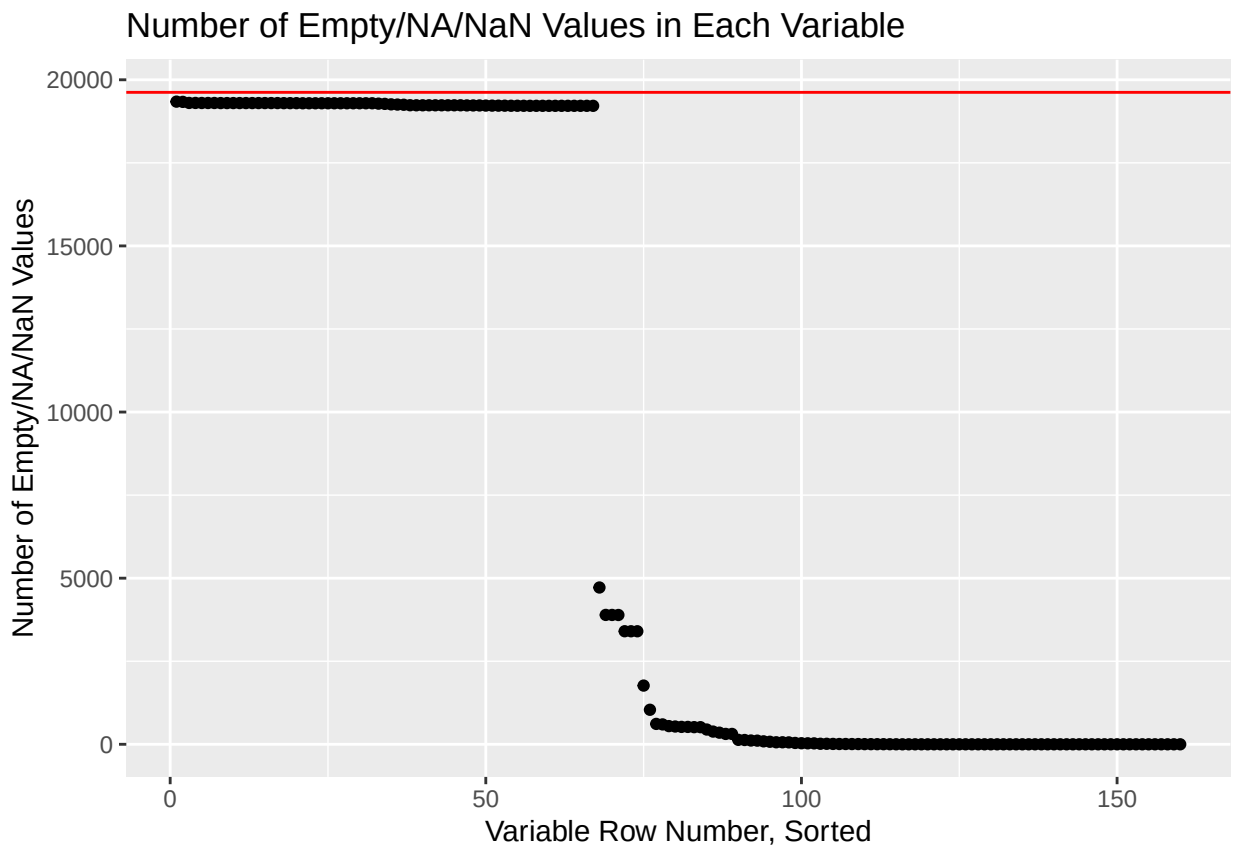
```
## [1] 19622   160
```

The training set is 160 observations of 19622 variables. There's a bit to learn just looking at the str() and summary() of the data. These have been omitted for now, as the 19622 variables causes quite a lot of text.

```
table <- data.frame("variable" = names(training),
                    "empty" = colSums(apply(is.empty(training), 2, as.numeric)),
                    row.names = NULL)
table %>% arrange(desc(empty)) %>% head()
```

```
##                  variable empty
## 1          var_roll_belt 19342
## 2         var_pitch_belt 19335
## 3    stddev_roll_forearm 19303
## 4       var_roll_forearm 19303
## 5 amplitude_pitch_forearm 19301
## 6     stddev_yaw_forearm 19301
```

```
table %>% arrange(desc(empty)) %>%
      ggplot(aes(x=1:160, y=empty)) +
      geom_point() +
      geom_hline(yintercept = nrow(training), col = "red") +
      labs(x = "Variable Row Number, Sorted",
           y = "Number of Empty/NA/NaN Values",
           title = "Number of Empty/NA/NaN Values in Each Variable")
```



Using is.empty() from rapportools package, I see that there is a lot of empty/NA/NaN values in these data. I've created a table showing the number of "NA", "NaN", empty vectors, etc. in each variable. We can also clearly see in this plot that a little under half of the variables have very little data stored in them, accounting for only 98.57% of the data. I will ignore/remove this data from the model, as with such a low population, there's no certainty on how it will predict a value.

```
training_sub <- training %>% select(as.character(table[table$empty < 19000,]$variable)) %>% select(-c(1
near_zero_variance <- nearZeroVar(training_sub)
training_clean <- training_sub %>% select(-c(near_zero_variance))

partition <- createDataPartition(training_clean$classe, p=0.75, list=FALSE)
train_set <- training_clean[partition,]
validation_set <- training_clean[-partition,]
test_set <- testing %>% select(as.character(table[table$empty < 19000 & table$variable != "classe",]$va
            select(-c(names(training_sub[,near_zero_variance]))))
data.frame("Training" = dim(train_set),"Validation" = dim(validation_set),"Testing" = dim(test_set), ro
```

```
##              Training Validation Testing
## Observations    14718       4904      20
## Variables          53         53      59
```

In the code above, I filtered out the empty variables, and then ran a near zero variance on the result to determine if I could further remove any additional variables with excess information. A training set and validation set were created from the original training data, and the size of each set is displayed above.

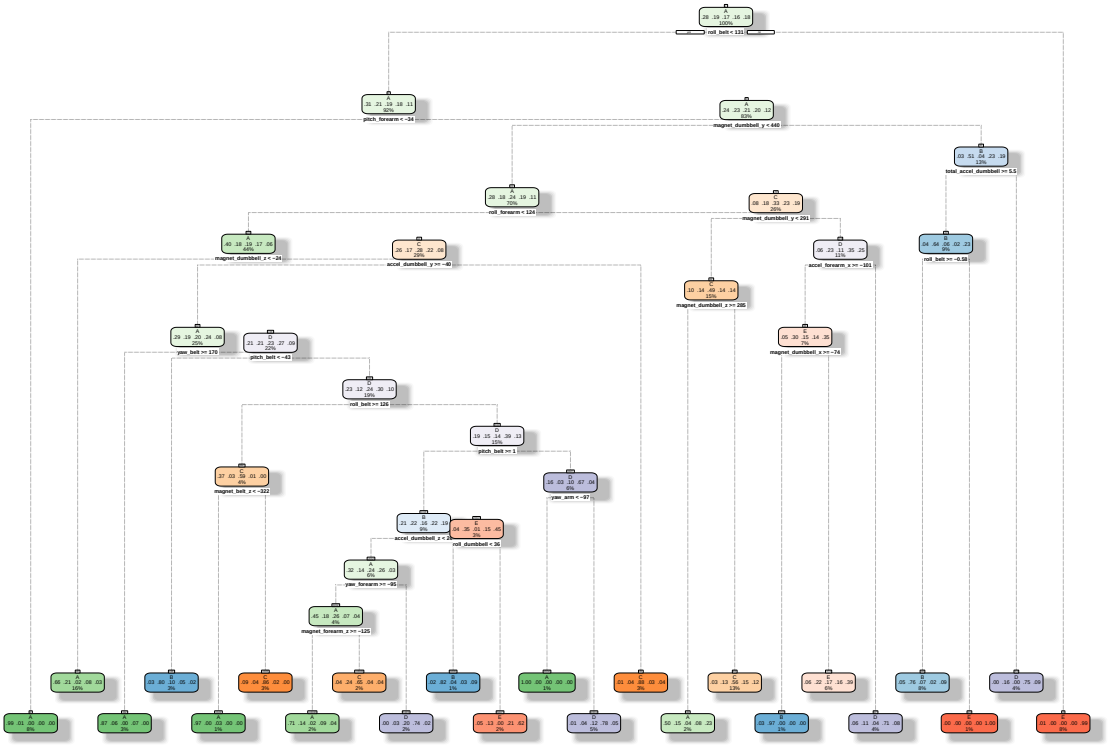
Modeling

I will use two different models, classification tree and random forest, as well as a generalized boosted model.

Classification Tree

I will start with the classification tree model. The classification tree dendrogram is shown below.

```
model_CT <- rpart(classe ~ ., train_set, method = "class")
fancyRpartPlot(model_CT)
```



I can test the validation set against this model:

```
predict_CT <- predict(model_CT, validation_set, type = "class")
matrix_CT <- confusionMatrix(predict_CT, validation_set$classe)
matrix_CT$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.375612e-01   6.659164e-01   7.250065e-01   7.498324e-01   2.844617e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   8.655587e-74
```

We see that the accuracy is roughly 73.8%, which is better than just randomly guessing, but doesn't offer a lot more than that.

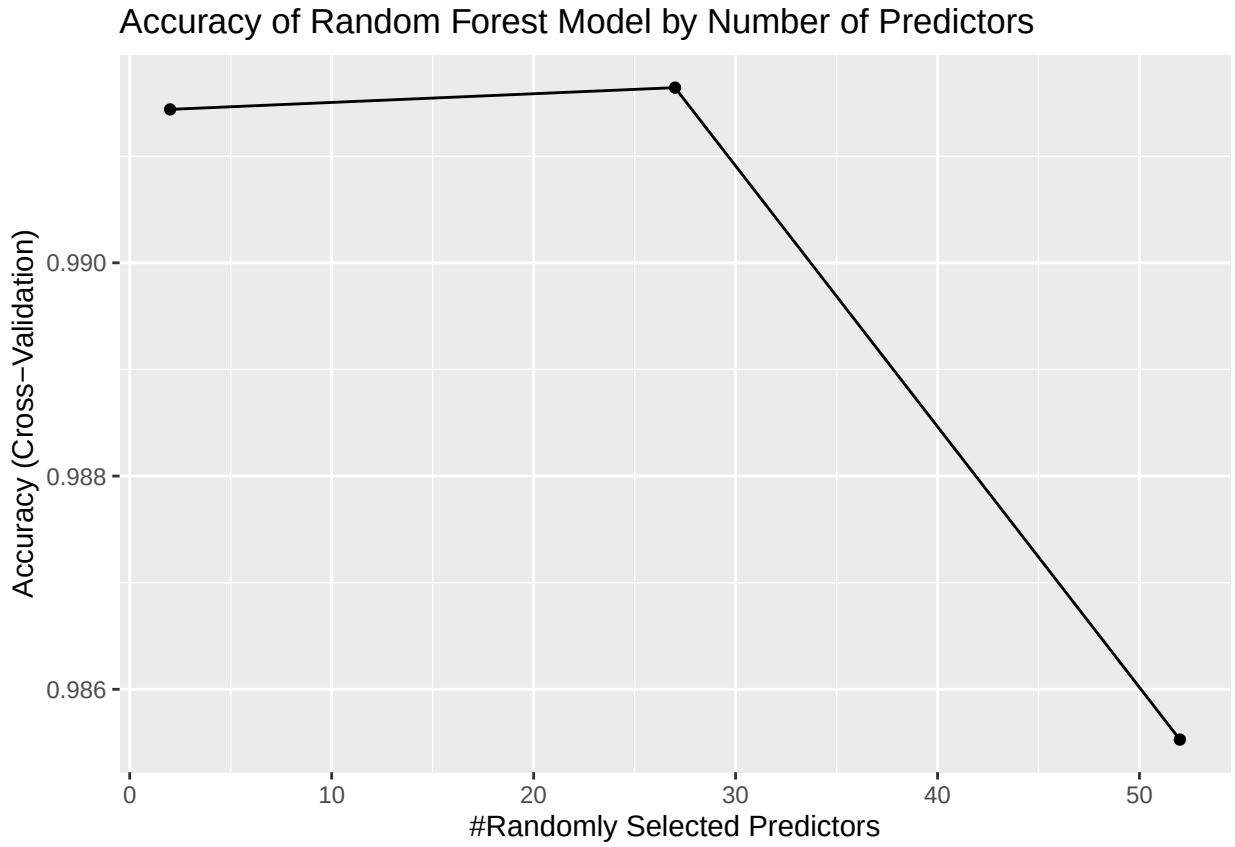
Random Forest

The next model will be utilizing random forest. I create the model below and show the summary.

```
model_RF <- train(classe ~ ., train_set, method = "rf", trControl = trainControl(method = "cv", number =
model_RF$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.69%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4179    4    1    0    1 0.001433692
## B   17 2820   10    1    0 0.009831461
## C    0   10 2548    9    0 0.007401636
## D    0    2   31 2377    2 0.014510779
## E    0    2    4    7 2693 0.004804139
```

```
ggplot(model_RF) + labs(title = "Accuracy of Random Forest Model by Number of Predictors")
```



As before, I then use the validation set against this model to test its accuracy:

```
predict_RF <- predict(model_RF, validation_set)
matrix_RF <- confusionMatrix(predict_RF, validation_set$classe)
matrix_RF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    4    0    0    0
##          B    0  944    7    0    0
##          C    0    1  845    7    3
##          D    0    0    3  797    3
##          E    0    0    0    0  895
##
## Overall Statistics
##
##                Accuracy : 0.9943
##                  95% CI : (0.9918, 0.9962)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9928
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9947   0.9883   0.9913   0.9933
## Specificity            0.9989   0.9982   0.9973   0.9985   1.0000
## Pos Pred Value         0.9971   0.9926   0.9871   0.9925   1.0000
## Neg Pred Value         1.0000   0.9987   0.9975   0.9983   0.9985
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1925   0.1723   0.1625   0.1825
## Detection Prevalence   0.2853   0.1939   0.1746   0.1637   0.1825
## Balanced Accuracy      0.9994   0.9965   0.9928   0.9949   0.9967
```

We can see we are at a 99.4290375% accuracy level with this test, with a very tight 95% confidence interval. Per the graph above, we can see that the max number of predictors one would want to use is around 27, as the accuracy of the model begins to fall after that, likely due to dependencies in the variables.

Since I am using a **separate validation set** for testing, I can simply calculate the **out-of-sample error rate** as (1-0.9942904) * 100 = 0.5709625%.

Stochastic Gradient Boosting

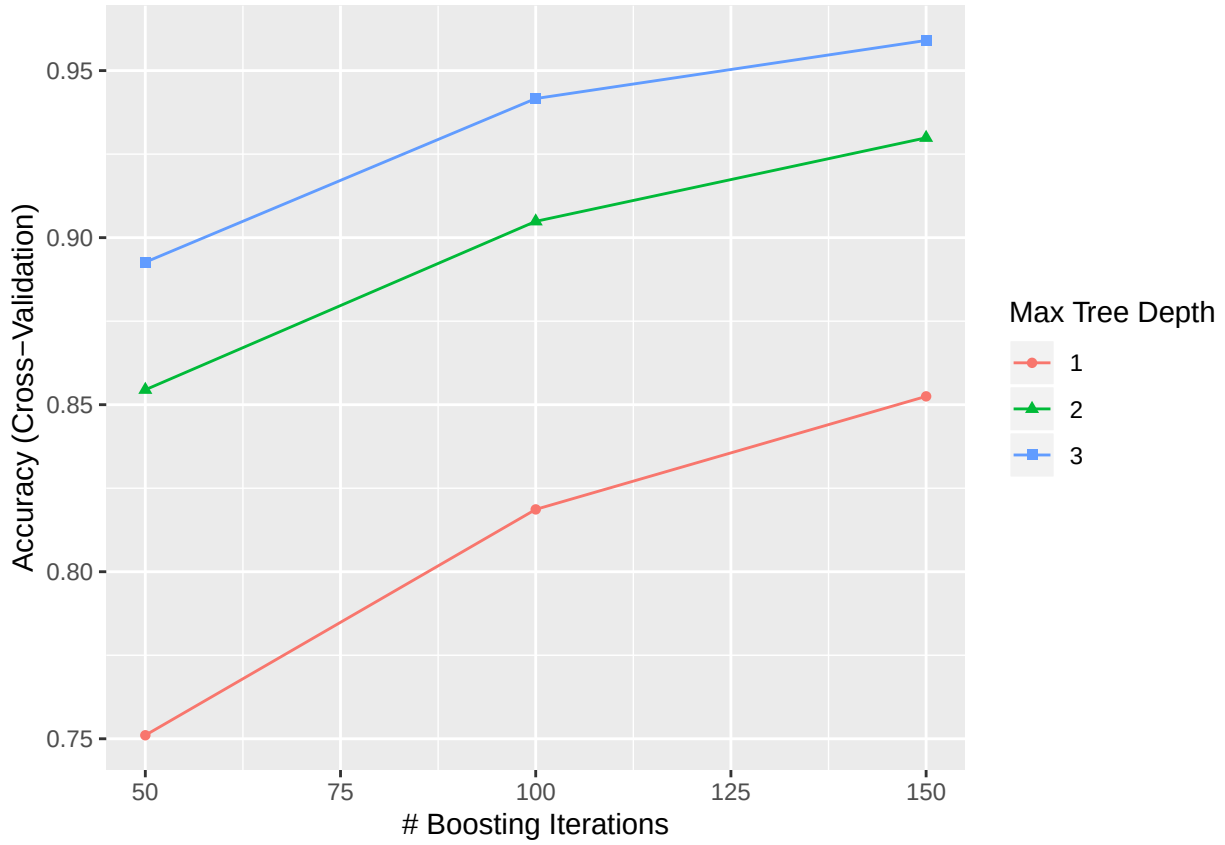
Finally, I will look at a boosted model for this data.

```
model_GBM <- train(classe ~ ., train_set, method = "gbm", trControl = trainControl(method = "cv", numbe
model_GBM
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 11774, 11775, 11774, 11774, 11775
```

```
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7510526  0.6845305
##   1                  100      0.8186562  0.7705685
##   1                  150      0.8524927  0.8133796
##   2                   50      0.8544633  0.8156042
##   2                  100      0.9048782  0.8796250
##   2                  150      0.9298816  0.9112603
##   3                   50      0.8926479  0.8640816
##   3                  100      0.9416362  0.9261426
##   3                  150      0.9590301  0.9481655
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
ggplot(model_GBM)
```




```
predict_GBM <- predict(model_GBM, validation_set)
matrix_GBM <- confusionMatrix(predict_GBM, validation_set$classe)
matrix_GBM
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1376   15    0    2    0
##          B   13  912   29    2   13
##          C    3   20  818   24   12
##          D    3    0    7  772   16
##          E    0    2    1    4  860
##
## Overall Statistics
##
##                Accuracy : 0.9662
##                  95% CI : (0.9607, 0.971)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9572
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9864   0.9610   0.9567   0.9602   0.9545
## Specificity            0.9952   0.9856   0.9854   0.9937   0.9983
## Pos Pred Value         0.9878   0.9412   0.9327   0.9674   0.9919
## Neg Pred Value         0.9946   0.9906   0.9908   0.9922   0.9898
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2806   0.1860   0.1668   0.1574   0.1754
## Detection Prevalence   0.2841   0.1976   0.1788   0.1627   0.1768
## Balanced Accuracy      0.9908   0.9733   0.9711   0.9769   0.9764
```

Again, running the validation set through this model, we see around a 96.6150082% accuracy, which is not quite as high as the random forest model.

Conclusion

I've shown that the Random Forest model is the best predictor out of these three models for this data set, with a 99.4290375% accuracy level. For the final test data, the predictions are given below.

```
predict(model_RF, test_set)
```

```
##  [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
```