Practical Machine Learning Final Project

Vineet Pillai

10/21/2019

Overview

The goal of this project is to predict the manner in which the exercise was performed, meaning predict the "classe" variable in the training set by using any of the other variables to make the prediction. This report describes how the model was built, how cross validation was used, the expected output of sample error, and why certain choices were made. This report also uses the prediction model to predict 20 different test cases.

```
library(caret)
library(rpart)
library(rpart.plot)
library(rattle)
library(ggplot2)
library(ggpubr)
library(gbm)
theme_set(theme_pubr())
library(vcd)
library(corrplot)
```

Getting and Cleaning the Data

```
trainData <- read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"),heade
dim(trainData)
```

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

```
testData <- read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"),header=
dim(testData)
```

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

We can see that training data has 19622 observations and 160 variables. Likewise, test data has 20 observations and 160 variables.

That is a lot of variables. We can reduce the variable count by removing any variables from the training and test data that are mostly NA values.

```
NATrain <- sapply(trainData, function(x) mean(is.na(x))) > 0.95
trainClean <- trainData[, NATrain==F]
dim(trainClean)
```

```
## [1] 19622    60
```

```
NATest <- sapply(testData, function(x) mean(is.na(x))) > 0.95
testClean <- testData[, NATest==F]
dim(testClean)
```

```
## [1] 20 60
```

This gets us down to 60 variables. We can also remove any Variables that have hardly any variation.

```
nzv <- nearZeroVar(trainClean, saveMetrics=TRUE)
trainClean <- trainClean[,nzv$nzv==FALSE]
dim(trainClean)
```

```
## [1] 19622    59
```

```
nzv2 <- nearZeroVar(testClean, saveMetrics=TRUE)
testClean <- testClean[,nzv2$nzv==FALSE]
dim(testClean)
```

```
## [1] 20 59
```

Lastly, we remove the first 5 identification variables that do not make sense to use for predictors

```
trainClean <- trainClean[, -(1:5)]
testClean  <- testClean[, -(1:5)]
dim(trainClean)
```

```
## [1] 19622    54
```

```
dim(testClean)
```

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

We have greatly reduced the number of variables. Now we are ready for building and testing our models.

Data Analysis for Model Selection

I already have Training and Test Data but I will need to have data for validation too, so I will split my training data into two sets, one for training and one for validation.

```
inTrain <- createDataPartition(y=trainClean$classe, p=0.7, list=FALSE)
train <- trainClean[inTrain, ]
valid <- trainClean[-inTrain, ]
dim(train)
```

```
## [1] 13737    54
```

```
dim(valid)
```

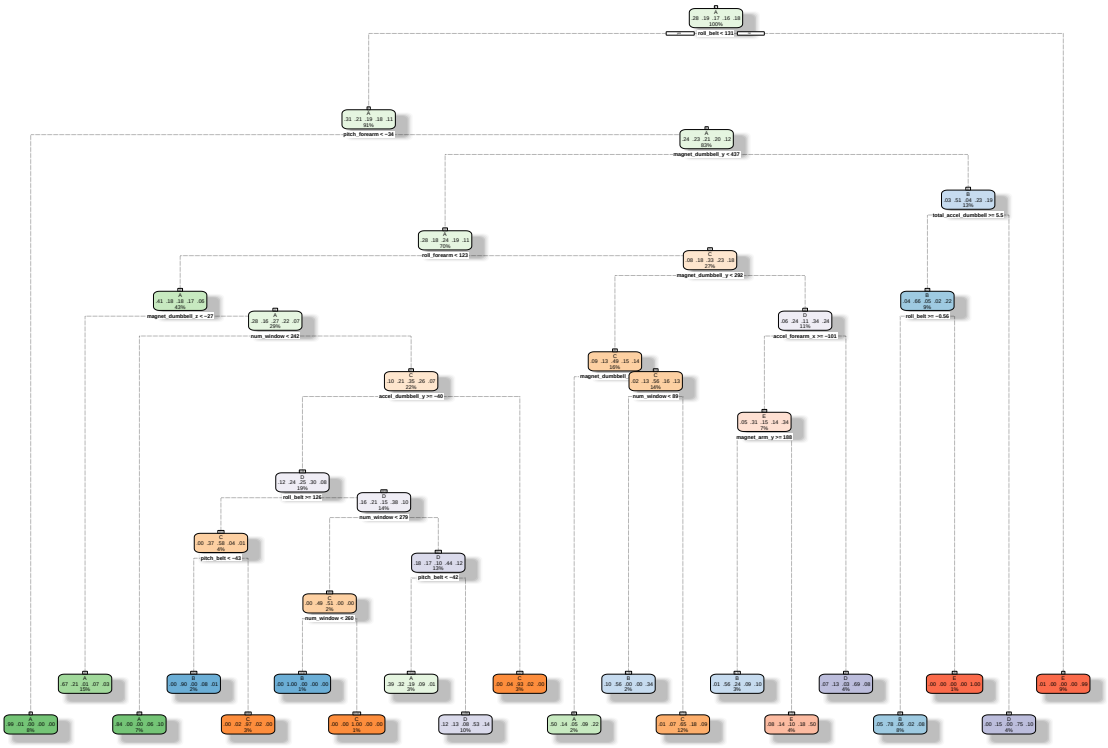
```
## [1] 5885   54
```

Decision Tree Model (DT)

Now I can building the first model. I will start by training and fitting a Model using a simple Decision Tree.

```
set.seed(33333)
trainDT <- rpart(classe ~ ., data=train, method="class")
fancyRpartPlot(trainDT)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2019−Oct−21 04:02:45 Mahe

Now we will run the DT model Predictions on the validation data

```
predictDT <- predict(trainDT, newdata=valid, type="class")
confMatrixDT <- confusionMatrix(predictDT, valid$classe)
confMatrixDT
```

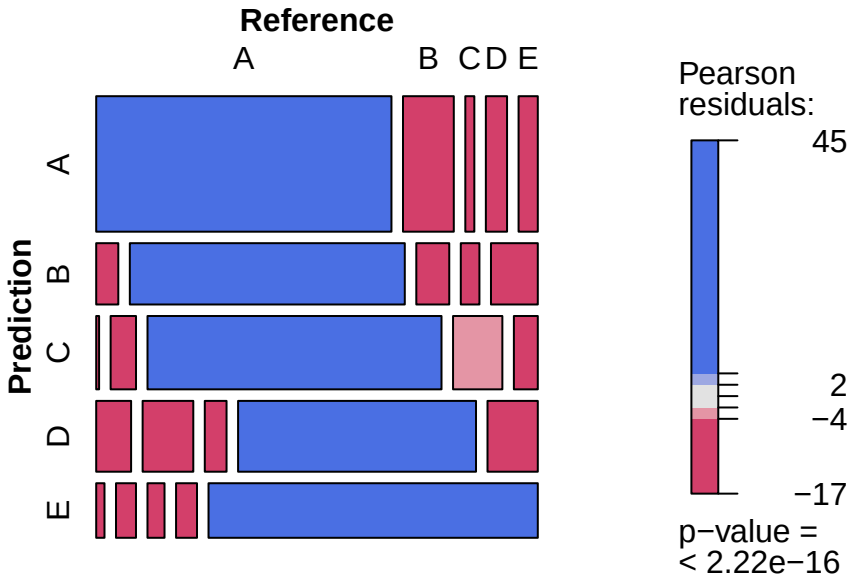
```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 1505  259   46  108   98
##          B   51  634   76   43  108
##          C    8   70  811  136   66
##          D   93  135   58  634  134
##          E   17   41   35   43  676
##
## Overall Statistics
##
##                Accuracy : 0.7239
##                  95% CI : (0.7123, 0.7353)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6485
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8990   0.5566   0.7904   0.6577   0.6248
## Specificity            0.8787   0.9414   0.9424   0.9147   0.9717
## Pos Pred Value         0.7465   0.6952   0.7434   0.6015   0.8325
## Neg Pred Value         0.9563   0.8985   0.9552   0.9317   0.9200
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2557   0.1077   0.1378   0.1077   0.1149
## Detection Prevalence   0.3426   0.1550   0.1854   0.1791   0.1380
## Balanced Accuracy      0.8888   0.7490   0.8664   0.7862   0.7982
```

Next we will plot the DT Model results

```
mosaic(confMatrixDT$table, shade = TRUE, legend = TRUE,
                           main = paste("Decision Trees: Accuracy =",
                                        round(confMatrixDT$overall['Accuracy'], 4)))
```

Decision Trees: Accuracy = 0.7239



The accuracy of the DT model (0.7239) was not as good as I expected. Also, decision trees are easy to interpret but one drawback is that the results can vary greatly across samples. I will try to increase the accuracy of the model by using a boosting method.

Generalized Boosted Model (GBM)

The GBM method builds upon the use of trees by splitting the sample into multiple copies and fitting a separate tree to each copy, and then applying the results of the current tree to the next tree thus training the model slowly.

For this model I will use a training control to specify repeated cross validation as the method to use for re-sampling. Using a training control will give me some control over how robust the cross validation will be.

```
set.seed(33333)
controlGBM <- trainControl(method = "repeatedcv", number = 3, repeats = 1)
trainGBM  <- train(classe ~ ., data=train, method = "gbm",
                   trControl = controlGBM, verbose = FALSE)
trainGBM$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 53 predictors of which 53 had non-zero influence.
```

Now we will run the GBM model Predictions on the validation data

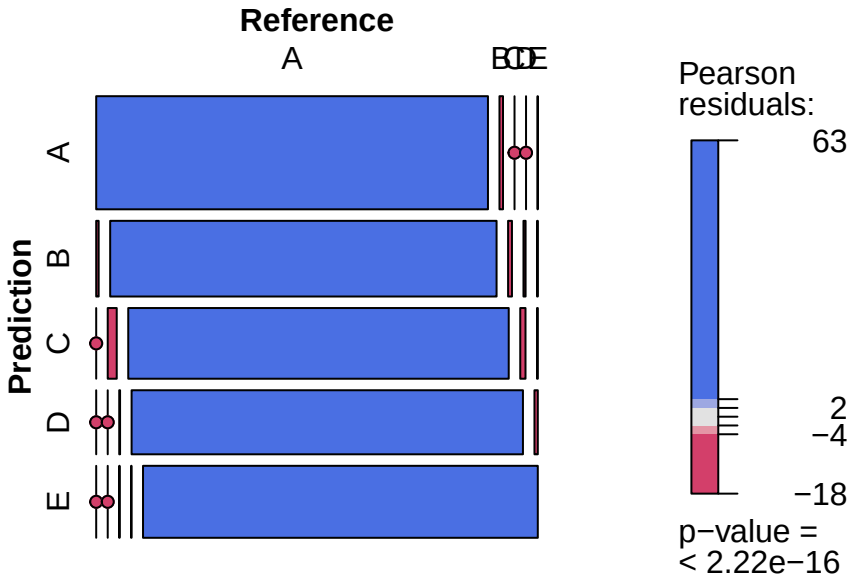
```
predGBM <- predict(trainGBM, newdata=valid)
confMatrixGBM <- confusionMatrix(predGBM, valid$classe)
confMatrixGBM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1667   15    0    0    1
##          B    7 1100   11    6    2
##          C    0   24 1012   14    2
##          D    0    0    2  943    8
##          E    0    0    1    1 1069
##
## Overall Statistics
##
##                Accuracy : 0.984
##                  95% CI : (0.9805, 0.9871)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9798
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9958   0.9658   0.9864   0.9782   0.9880
## Specificity            0.9962   0.9945   0.9918   0.9980   0.9996
## Pos Pred Value         0.9905   0.9769   0.9620   0.9895   0.9981
## Neg Pred Value         0.9983   0.9918   0.9971   0.9957   0.9973
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2833   0.1869   0.1720   0.1602   0.1816
## Detection Prevalence   0.2860   0.1913   0.1788   0.1619   0.1820
## Balanced Accuracy      0.9960   0.9801   0.9891   0.9881   0.9938
```

Next we will plot the GBM Model results

```
mosaic(confMatrixGBM$table, shade = TRUE, legend = TRUE,
                          main = paste("Generalized Boosted Mode: Accuracy =",
                                       round(confMatrixGBM$overall['Accuracy'], 4)))
```

Generalized Boosted Mode: Accuracy = 0.984



The accuracy of the GBM model(0.984) was much better than the DT Model (0.7239). However, I will try one more model to see if I may get even better accuracy.

Random Forest Model

For the last model, I will use the Random Forest method, which re-samples using a random set of predictors to reduce the variance and the error rate. I will again use a train control to specify repeated cross validation as the method for re-sampling.

```
set.seed(33333)
controlRF <- trainControl(method="repeatedcv", number=3, verboseIter=FALSE)
trainRF <- train(classe ~ ., data=train, method="rf",
                         trControl=controlRF)
trainRF$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: 0.27%
## Confusion matrix:
##      A    B    C    D    E  class.error
```

```
## A 3905    1    0    0    0 0.0002560164
## B    8 2646    4    0    0 0.0045146727
## C    0    6 2390    0    0 0.0025041736
## D    0    0   11 2240    1 0.0053285968
## E    0    1    0    5 2519 0.0023762376
```

Now we will run the Predictions on the validation data

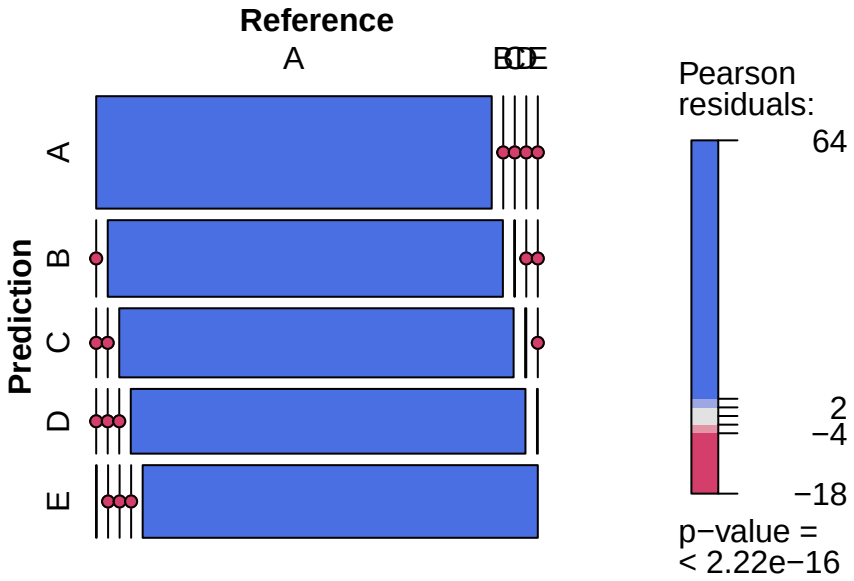
```
# prediction on Test dataset
predRF <- predict(trainRF, newdata=valid)
confMatrixRF <- confusionMatrix(predRF, valid$classe)
confMatrixRF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    0    0    0    0
##          B    0 1139    1    0    0
##          C    0    0 1025    3    0
##          D    0    0    0  961    2
##          E    1    0    0    0 1080
##
## Overall Statistics
##
##                Accuracy : 0.9988
##                  95% CI : (0.9976, 0.9995)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9985
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   1.0000   0.9990   0.9969   0.9982
## Specificity            1.0000   0.9998   0.9994   0.9996   0.9998
## Pos Pred Value         1.0000   0.9991   0.9971   0.9979   0.9991
## Neg Pred Value         0.9998   1.0000   0.9998   0.9994   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1935   0.1742   0.1633   0.1835
## Detection Prevalence   0.2843   0.1937   0.1747   0.1636   0.1837
## Balanced Accuracy      0.9997   0.9999   0.9992   0.9982   0.9990
```

Next we will plot the RF Model results

```
mosaic(confMatrixRF$table, shade = TRUE, legend = TRUE,
                           main = paste("Random Forest: Accuracy =",
                                        round(confMatrixRF$overall['Accuracy'], 4)))
```

Random Forest: Accuracy = 0.9988



#Model selection

Summary of Results
- Decision Trees: Accuracy = 0.7239
- Generalized Boosted Mode (GBM): Accuracy = 0.984
- Random Forest (RF): Accuracy = 0.9988

From the results, it is clear that Random Forest is the best fit for the data so I will fit the RF model to the full set of the training data to get the best prediction against the actual test data.

```
trainFull <- trainClean
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
trainRF_Full <- train(classe ~ ., data=trainFull, method="rf",
                      trControl=controlRF)
trainRF_Full$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: 0.12%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 5578    1    0    0    1 0.0003584229
```

```
## B    5 3790    2    0    0 0.0018435607
## C    0    5 3417    0    0 0.0014611338
## D    0    0    8 3207    1 0.0027985075
## E    0    0    0    1 3606 0.0002772387
```

Getting Quiz Results by Applying the Model

Lastly, I will apply the RF Model to the test data to predict the answers to the 20 Test Cases.

```
testFull <- testClean
predictRF_Full <- predict(trainRF_Full, newdata=testFull)
predictRF_Full
```

```
##  [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
```