Machine Learning Project

by Toby Popenfoose

July 22, 2015

Introduction

This project used the data from http://groupware.les.inf.puc-rio.br/har to develop a predictive model to predict the "classe" from weight lifting data sets in Human Activity Recognition (HAR).

Executive Summary

I used 5 different types of classification models to evaluate the accuracy of predicting the correct "classe". The C5.0 model had the highest accuracy and the lowest Out Of Sample Error (OOSE) rate. It had an OOSE of 0.8%.

Load the Required R Packages

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

Get the Data

```
dfTrain <- "./data/pml-training.csv"
dfTest  <- "./data/pml-testing.csv"

if (!file.exists("./data")) {
  dir.create("./data")
}

if (!file.exists(dfTrain)) {
  download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
                destfile=dfTrain, method="curl")
}

if (!file.exists(dfTest)) {
  download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
                destfile=dfTest, method="curl")
}
```

Load the Data

```
dfTrainRaw <- read.csv(dfTrain)
dim(dfTrainRaw)
```

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

```
dfTestRaw <- read.csv(dfTest)
dim(dfTestRaw)
```

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

Clean the Data

First get rid of the "X" id, windows and timestamp columns.

```
dfTrainRaw <- dfTrainRaw[ , -(1:7)] # get rid of X id, windows and timestamps columns
```

Now, get rid of the columns with over half the values are missing.

```
dfTrainRaw <- dfTrainRaw[, colSums(is.na(dfTrainRaw)) <= nrow(dfTrainRaw) / 2]
    # get rid of columns with over half values missing
```

Now remove the remaining factor columns since the majority of their rows are empty.

```
                                        # now remove factor columns
# from http://www.markhneedham.com/blog/2014/09/29/r-filtering-data-frames-by-column-type-x-must-be-n
dfTrainClean <- dfTrainRaw[sapply(dfTrainRaw, function(x) !is.factor(x))]
```

And last but not least, put back in the classe column.

```
dfTrainClean$classe <- dfTrainRaw$classe
```

One last check of the cleaned data.

```
summary(dfTrainClean)
```

Preprocess the Data

I will create two set of data. One to train my models on, the other to test the out of sample error and accuracy. I have used a 60% - 40% split between training set and testing set.

```
set.seed(123) # for reproducibility
inTrain <- createDataPartition(dfTrainClean$classe, p=0.60, list=F)
dfTrainData <- dfTrainClean[inTrain, ]
dfTestData <- dfTrainClean[-inTrain, ]
```

Build the Models

To speed up the models building, I will use the doParallel package. It achieves approximately a speedup factor of 2 using 4 cores on my laptop.

```
library(doParallel)
```

```
## Loading required package: foreach
## Loading required package: iterators
## Loading required package: parallel
```

```
registerDoParallel(cores=4)
```

Now I train 5 different model types. Random Forest (rf), Partial Least Square (pls), Classification Tree (rpart), Stochastic Gradient Boosting (gbm) and C5.0 (Decision Trees and Rule-Based Models). I use 10 repeats of 10 fold cross validated models to reduce overfitting.

```
set.seed(123) # for reproducibility
rfFit <- train(classe ~ ., data=dfTrainData, method="rf",
                trControl=trainControl(method="cv", number=10, repeats=10, allowParallel=TRUE),
               ntree=500)
```

```
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(123) # for reproducibility
plsFit <- train(classe ~ ., data=dfTrainData, method="pls",
                 trControl=trainControl(method="cv", number=10, repeats=10, allowParallel=TRUE))
```

```
## Loading required package: pls
##
## Attaching package: 'pls'
##
## The following object is masked from 'package:caret':
##
##     R2
##
## The following object is masked from 'package:stats':
##
##     loadings
```

```
set.seed(123) # for reproducibility
rpartFit <- train(classe ~ ., data=dfTrainData, method="rpart",
                  trControl=trainControl(method="cv", number=10, repeats=10, allowParallel=TRUE))
```

```
## Loading required package: rpart
```

```
set.seed(123) # for reproducibility
gbmFit <- train(classe ~ ., data=dfTrainData, method="gbm",
                  trControl=trainControl(method="cv", number=10, repeats=10, allowParallel=TRUE))
```

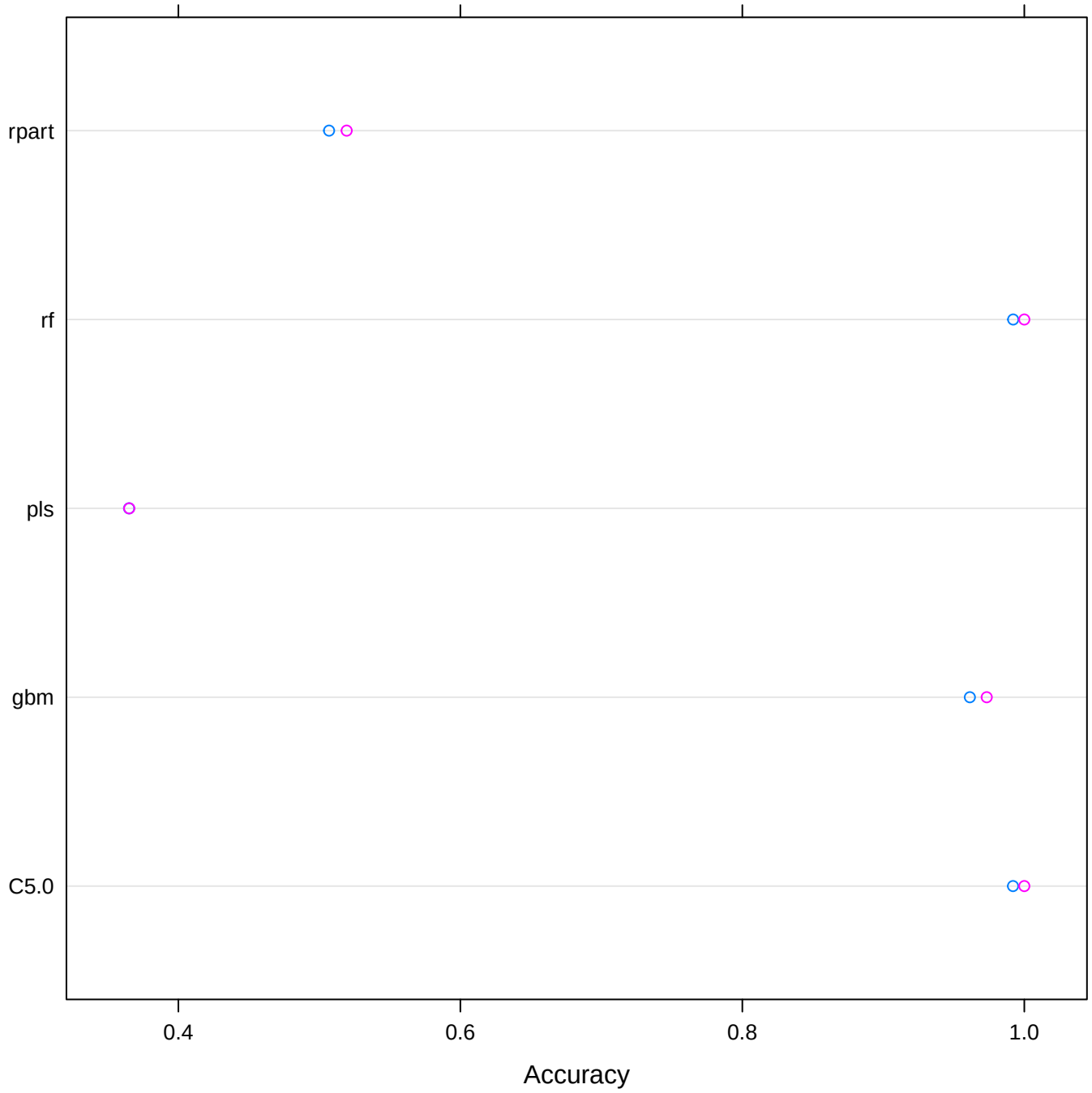
```
## Loading required package: gbm
## Loading required package: survival
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: splines
## Loaded gbm 2.1.1
## Loading required package: plyr
```

```
set.seed(123) # for reproducibility
c50Fit <- train(classe ~ ., data=dfTrainData, method="C5.0",
                    trControl=trainControl(method="cv", number=10, repeats=10, allowParallel=TRUE))
```

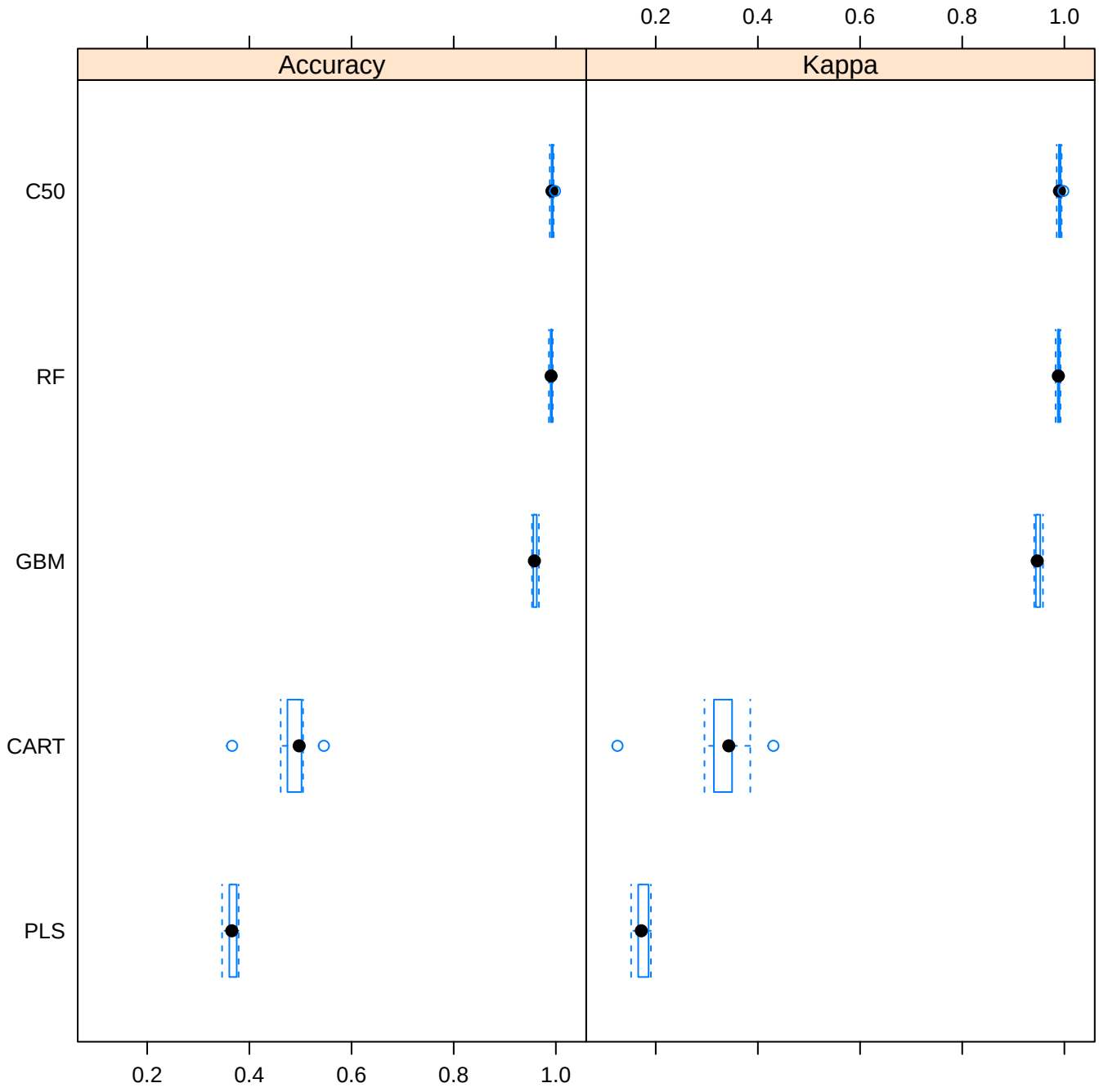
```
## Loading required package: C50
```

I take a graphical look at each models accuracy.

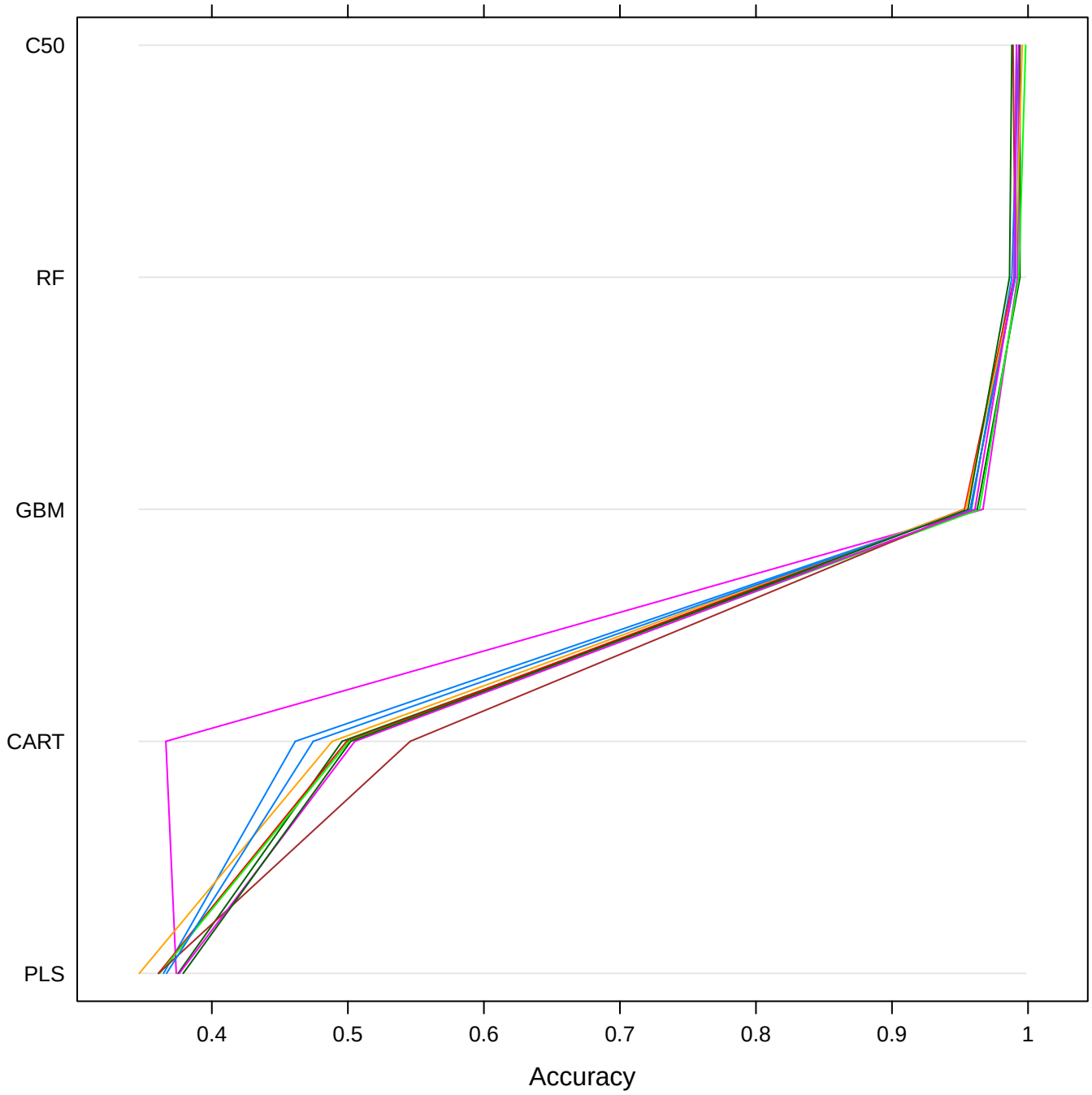
```
predVals <- extractPrediction(list(rfFit, rpartFit, plsFit, gbmFit, c50Fit),
                              testX = dfTestData[ , -53],
                              testY = dfTestData$classe)
plotObsVsPred(predVals)
```




```
resamps <- resamples(list(RF=rfFit, CART=rpartFit, PLS=plsFit, GBM=gbmFit, C50=c50Fit))
bwplot(resamps)
```




```
parallelplot(resamps)
```



Generate the Confusion Matrix

```
predictRF <- predict(rfFit, dfTestData)
confusionMatrix(dfTestData$classe, predictRF)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2228    3    0    1    0
```

```
##          B   11 1503    4    0    0
##          C    0   13 1355    0    0
##          D    0    0   23 1263    0
##          E    0    0    2    4 1436
##
## Overall Statistics
##
##                Accuracy : 0.992
##                  95% CI : (0.99, 0.994)
##     No Information Rate : 0.285
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.99
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.995    0.989    0.979    0.996    1.000
## Specificity             0.999    0.998    0.998    0.997    0.999
## Pos Pred Value          0.998    0.990    0.990    0.982    0.996
## Neg Pred Value          0.998    0.997    0.996    0.999    1.000
## Prevalence              0.285    0.194    0.176    0.162    0.183
## Detection Rate          0.284    0.192    0.173    0.161    0.183
## Detection Prevalence    0.284    0.193    0.174    0.164    0.184
## Balanced Accuracy       0.997    0.994    0.989    0.996    1.000
```

```
predictC50 <- predict(c50Fit, dfTestData)
confusionMatrix(dfTestData$classe, predictC50)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2222   10    0    0    0
##          B   11 1499    6    1    1
##          C    1    5 1351   11    0
##          D    0    0    7 1276    3
##          E    0    1    1    5 1435
##
## Overall Statistics
##
##                Accuracy : 0.992
##                  95% CI : (0.99, 0.994)
##     No Information Rate : 0.285
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.99
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.995    0.989    0.990    0.987    0.997
## Specificity             0.998    0.997    0.997    0.998    0.999
## Pos Pred Value          0.996    0.987    0.988    0.992    0.995
## Neg Pred Value          0.998    0.997    0.998    0.997    0.999
## Prevalence              0.285    0.193    0.174    0.165    0.183
## Detection Rate          0.283    0.191    0.172    0.163    0.183
## Detection Prevalence    0.284    0.193    0.174    0.164    0.184
## Balanced Accuracy       0.996    0.993    0.994    0.993    0.998
```

Calculate Accuracy and Out of Sample Error for Random Forest

The Out of Sample Error Rate for the Random Forest model is 0.88%.

```
accuracy <- postResample(predictRF, dfTestData$classe)
accuracy
```

```
## Accuracy    Kappa
##   0.9922   0.9902
```

```
outOfSampleError <- 1 - as.numeric(confusionMatrix(dfTestData$classe, predictRF)$overall[1])
outOfSampleError
```

```
## [1] 0.007775
```

Calculate Accuracy and Out of Sample Error for C5.0

The Out of Sample Error Rate for the C5.0 model is 0.8%.

```
accuracy <- postResample(predictC50, dfTestData$classe)
accuracy
```

```
## Accuracy    Kappa
##   0.9920   0.9898
```

```
outOfSampleError <- 1 - as.numeric(confusionMatrix(dfTestData$classe, predictC50)$overall[1])
outOfSampleError
```

```
## [1] 0.00803
```

Generate Submission Files with Raw Test Data

```
rfPredicted <- predict(rfFit, dfTestRaw)
rfPredicted
```

```
##  [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
```

```
c50Predicted <- predict(c50Fit, dfTestRaw)
c50Predicted
```

```
##  [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
```

```
answers <- c50Predicted

pml_write_files <- function(x) {
    n = length(x)
    for(i in 1:n){
        filename = paste0("problemResults/problem_id_", i, ".txt")
        write.table(x[i], file=filename, quote=FALSE, row.names=FALSE, col.names=FALSE)
    }
}

pml_write_files(answers)
```