Machine Learning Project

Kyle

July 22, 2016

Step 1: Reading the Data

Download and read the data into R.

```
library(caret); library(ggplot2); library(rattle); library(parallel); library(doParallel)
```

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

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

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
#reading in the data
training_url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
test_url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
if(!file.exists("test_data.csv")){
  download.file(test_url, destfile = "test_data.csv")}
if(!file.exists("training_data.csv")){
  download.file(training_url, destfile = "training_data.csv")}
training <- read.csv("training_data.csv")
testing <- read.csv("test_data.csv")
```

Step 2: Clean the Data

Remove the columns that are completely filled with NA's. Take out the columns that have low variances. Take out the first six columns that have little to do with aiding prediction.

```
#exploratory data analysis
#there are a ton of columns with na's, time to shrink this data set
training <- training[, colSums(is.na(training)) == 0]
testing <- testing[, colSums(is.na(testing)) == 0]
#cleaning the training set
zerovar <- nearZeroVar(training, saveMetrics = FALSE)
training <- training[ , -zerovar]
training <- training[, -c(1,2,3,4,5,6)]
```

Step 3: Split the training set into 60% training, 40% validation.

```
#split the training data into a validation set and training set at the 0.6 level
inTrain <- createDataPartition(training$classe, p = 0.6, list = FALSE)
training <- training[inTrain, ]
checker <- training[-inTrain, ]
```

Step 4: Make clusters for Parallel Processing

Utitlizing parallel processing will speed up the process of using random forests. Additionally, making k-fold cross validation will help with determining the accuracy of our prediction and reduction in variability of our prediction,

```
#making clusters for parallel processing
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)
#configuring trainControl object
fitControl <- trainControl(method = "cv", number = 5, allowParallel = TRUE)
```

Step 5a: Build an LDA model

```
model_lda <- train(classe ~. , data = training, method = "lda")
```

```
## Loading required package: MASS
```

Step 5b: Build the Random Forest Model.

Builing the random forest model, although painstakingly slow, yields the best accuracy.

```
#model build
model_rf <- train(classe ~ ., data = training, trControl = fitControl)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
stopCluster(cluster)
```

Step 6: Check the Random Forest Model on the Checker validation Set.

Now that we have a model, we utilize that model to check out validation set 'checker'.

```r
predict_lda_checker <- predict(model_lda, checker)
predict_rf_checker <- predict(model_rf, checker)
print(c(confusionMatrix(predict_rf_checker, checker$classe),
        confusionMatrix(predict_lda_checker, checker$classe)))
```

```
## $positive
## NULL
## 
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 1312    0    0    0    0
##          B    0  942    0    0    0
##          C    0    0  800    0    0
##          D    0    0    0  798    0
##          E    0    0    0    0  844
## 
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull 
##      1.0000000      1.0000000      0.9992148      1.0000000      0.2793867 
## AccuracyPValue  McnemarPValue 
##      0.0000000            NaN 
## 
## $byClass
##          Sensitivity Specificity Pos Pred Value Neg Pred Value Prevalence
## Class: A           1           1              1              1  0.2793867
## Class: B           1           1              1              1  0.2005963
## Class: C           1           1              1              1  0.1703578
## Class: D           1           1              1              1  0.1699319
## Class: E           1           1              1              1  0.1797274
##          Detection Rate Detection Prevalence Balanced Accuracy
## Class: A      0.2793867            0.2793867                 1
## Class: B      0.2005963            0.2005963                 1
## Class: C      0.1703578            0.1703578                 1
## Class: D      0.1699319            0.1699319                 1
## Class: E      0.1797274            0.1797274                 1
## 
## $dots
## list()
## 
## $positive
## NULL
## 
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 1040  133   74   38   30
##          B   31  602   83   38  145
##          C  131  120  533   92   80
##          D  106   34   94  600   73
##          E    4   53   16   30  516
##
## $overall
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.008092e-01   6.223148e-01   6.874842e-01   7.138841e-01   2.793867e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   3.108492e-46
##
## $byClass
##          Sensitivity Specificity Pos Pred Value Neg Pred Value Prevalence
## Class: A   0.7926829   0.9187352      0.7908745      0.9195504  0.2793867
## Class: B   0.6390658   0.9208844      0.6696329      0.9104556  0.2005963
## Class: C   0.6662500   0.8914271      0.5575314      0.9286096  0.1703578
## Class: D   0.7518797   0.9212417      0.6615215      0.9477435  0.1699319
## Class: E   0.6113744   0.9732606      0.8336026      0.9195487  0.1797274
##          Detection Rate Detection Prevalence Balanced Accuracy
## Class: A      0.2214651            0.2800256         0.8557091
## Class: B      0.1281942            0.1914395         0.7799751
## Class: C      0.1135009            0.2035775         0.7788386
## Class: D      0.1277683            0.1931431         0.8365607
## Class: E      0.1098807            0.1318143         0.7923175
##
## $dots
## list()
```

Noting a strong finish, we use the model on the 20 data points located in the test set.

Step 7: Use the Random Forest Model to predict on the Testing Set since it has the higher accuracy.

```r
print(predict(model_rf, testing))
```

```
##  [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
```