MachineLearningProject

Jerad Acosta

January 22, 2015

Goal

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set.

Strategy

- We shall being by acquiring and loading our data.
- Then a quick overview of the data with reflection to our primary goal should allow us to focus on proper feature selection or creation.
- Then, with a proper idea as to the the size and scope of our data with respect to our goal, we can begin the process of removing variables or observations which appear to be corrupted with the purpose of fitting our model to as close to real-world measurements and applicable resources.
- With out data set reduced, managing and manipulating it should be a much easier task, as will interpreting the meaning of our model.
- While this report will only show the final and most accurate and understandable model, there will have been many attempts from basics of plotting potential predictors, to creating new predictors through Principle Component analysis; additionally models from basic linear regression to trees and random forests all attempting methods of bagging and boosting will be tried. However, for the sake of brevity, only the final model and its production will be shown here with perhaps a few nods to why a different attempt either failed or was not selected.

Acquiring, summarizing and compressing the data set

```
setwd('/Users/irJERAD/Courses/Coursera Repo/(C)Practical Machine Learning/PracticalMachineLearning')
```

Download training and testing datasets creating any necessary directories

```
# check if data directory exists
if(!file.exists("data")) {
        # create directory for datasets
        dir.create("data")

        # download testing and training data sets
        trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
        testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

        download.file(trainURL, destfile = "./data/trainingSet.csv", method = "curl")
        download.file(testURL, destfile = "./data/testingSet.csv", method = "curl")
}
```

```
library(caret)
```

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

```
# since we will use the "rf" method we can save time by preloading dependencies
library(randomForest)
```

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

```
# to allow us to compute multiple groups of trees on seperate cores
# also known as Parallel computing, we use the foreach and doParallel libraries
library(foreach)
library(doParallel)
```

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

```
set.seed(1111)
```

load data into memory

```
trainSet <- read.csv("./data/trainingSet.csv", na.strings=c("#DIV/0!"))
#testSet <- read.csv("./data/testingSet.csv", na.strings=c("#DIV/0!"))
```

Quickly review the data as such

```
# summary immediately shows many NA values
# and the first column X appears to be an index column
summar(trainSet)
# This shows our training contains 19622 observations and 160 variables
dim(trainSet)
# when we check how many columns are completely full we find 93
sum(colSums(is.na(trainSet)) == 0)
```

With more than half the variables completely filled it makes sense to stick to these as our predictors and thus remove any incompleted columns to avoid a few observations heavily affecting the weight it holds on a particular classification. Additionally, common sense confirms that columns such as X, an index column, user_name, and a few others provide us with absolutely no predictive value and could only lead toward miss classification and incorrectly fitting to inappropriate variables

```
# coerce remaining values into numeric for ease of computing and comparing model
for(i in c(1:(ncol(trainSet)-1))) {
        trainSet[,i] = as.numeric(as.character(trainSet[,i]))
        }
#for(i in c(1:ncol(testSet)-1)) {
#        testSet[,i] = as.numeric(as.character(testSet[,i]))
#        }
#to remove user name, repetitive index, and other non informative columns
trainSet <- trainSet[ ,-c(1:7)]
#testSet <- testSet[ , -c(1:7)]
keep <- colnames(trainSet[colSums(is.na(trainSet)) == 0])
```

```
#to remove variables that are not completed
trainSet <- trainSet[keep]
# to make sure we dont remove columns from one set and not the other
#we use the trainSet index
#testSet <- testSet[keep]
```

partition a probe set for cross validation Our new Training data will retain 70% of the training set and 30% will be set aside for a probe test set, after which we can update the model prior to using the testing set

```
inTrain <- createDataPartition(y = trainSet$classe, p = 0.7, list = FALSE)
train <- trainSet[inTrain, ]
test <- trainSet[-inTrain, ]
```

Train Random Forests models

With such a large combination of variables and factors and because random forests become so powerful in reducing the variability of their predictions with the use of more trees and forests, it only makes sense to do such a cumbersome and computationally heavy tasks across multiple cores. The foreach package as described by Steve Weston went a long way in helping me harness the grammar of executing functions for parallel computation please see this page for more information on how I learned to use this package The doParallel package is a "parallel backend" for the foreach package - as described by author Steve Weston and Rich Calaway. If you would like to know more about the doParallel package please check here for a simple introduction to doParallel and how it works with foreach

```
# this command will allocate half the computers total number of cores
registerDoParallel()
x <- subset(train, select = -classe)
y <- train$classe
rfModel <- foreach(ntree = rep(200, 6), .combine = combine, .multicombine = TRUE,
              .packages = 'randomForest') %dopar% {
                      randomForest(x, y, ntree=ntree)
              }
```

Create and evaluate Predictions for training and test sets

```
trainPred <- predict(rfModel, newdata=train)
confusionMatrix(trainPred, train$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3906    0    0    0    0
##          B    0 2658    0    0    0
##          C    0    0 2396    0    0
##          D    0    0    0 2252    0
##          E    0    0    0    0 2525
##
## Overall Statistics
##
```

```
##                Accuracy : 1
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

```
testPred <- predict(rfModel, newdata=test)
confusionMatrix(testPred,test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    4    0    0    0
##          B    2 1133    6    0    0
##          C    0    2 1020   11    0
##          D    0    0    0  953    2
##          E    0    0    0    0 1080
##
## Overall Statistics
##
##                Accuracy : 0.9954
##                  95% CI : (0.9933, 0.997)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9942
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9947   0.9942   0.9886   0.9982
## Specificity            0.9991   0.9983   0.9973   0.9996   1.0000
## Pos Pred Value         0.9976   0.9930   0.9874   0.9979   1.0000
## Neg Pred Value         0.9995   0.9987   0.9988   0.9978   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1925   0.1733   0.1619   0.1835
## Detection Prevalence   0.2848   0.1939   0.1755   0.1623   0.1835
## Balanced Accuracy      0.9989   0.9965   0.9957   0.9941   0.9991
```

Conclusion

With 99.59% accuracy, the random forest model has shown its great predictive power. The biggest caveat being the time it takes the model and then combine these 200 trees in each of the 6 random forests. I believe, however, that by using the doParallel library this cost has been significantly reduced. Prior to parallel computing, creating the **rfModel** took over 20 minutes. With the addition of doParallel running with foreach and particularly once I found the multicombine functionality, this time was able to reduce to below 2 minutes.