Practical Machine Learning Course Project

Eddo W. Hintoso

September 22, 2015

Contents

Processing Data

First and foremost, download the files and load necessary packages:

```
# load package if not yet loaded
if(!("caret" %in% loadedNamespaces())) {
    library(caret)
}
# download data
if(!file.exists("pml-training.csv")){
    download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
        destfile = "pml-training.csv")
}
if(!file.exists("pml-testing.csv")){
    download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
        destfile = "pml-testing.csv")
}
# load data
train <- read.csv('pml-training.csv', header = TRUE, na.strings = c("", "NA", "#DIV/0!"))
test <- read.csv('pml-testing.csv', header = TRUE, na.strings = c("", "NA", "#DIV/0!"))
```

Note that I have treated some string values as `NA`. This is because upon primary inspection of the data there were many missing data that best be uniformly expressed as `NA`. The below R code is shown on how the author came to detect missing values - however it is only for display and will not be evaluated due to the length of the output exceeding the degree of readability.

```
# check if there any NA in any column variables of train
TRUE %in% sapply(train, function(col) {NA %in% col})
# individually inspect odd outputs by iterating over factor and character variables
# helps to see the factor levels
for (col in 1:ncol(train)) {
    if(class(train[, col]) == "factor" |
       class(train[, col]) == "character") {
        print(unique(train[, col]))
    }
}
```

In order to run the machine learning algorithms, the features used cannot contain any `NA` values. Let us quickly inspect how many variables are complete:

```
completionStatus <- sapply(train, function(col) {!(NA %in% col)})
c("Complete Variables" = sum(completionStatus),
  "Total Variables" = length(completionStatus),
  "Percent Completion" = 100 * sum(completionStatus) / length(completionStatus))
```

```
## Complete Variables    Total Variables Percent Completion
##               60.0              160.0               37.5
```

We now know that only 60 variables have complete data - so we will only use these column variables for our predictor models, since imputing data carries a considerable risk and may affect the accuracy of the predictor model we're going to fit. Think of this as taking a measure to potentially overfit a predictor model due to too many column variables. Now all that is left is filtering the training and testing data sets from the incomplete data columns:

```r
# initialize index to empty vector
complete_index = c()
# iterate from second column since first column is just row index
for (col in 2:length(completionStatus)) {
    if (completionStatus[[col]] == TRUE) {
        complete_index = c(complete_index, col)
    }
}
# filter training and testing dataset into just complete data columns
train <- train[, complete_index]
test <- test[, complete_index]
```

Both the training and testing data should now only have 59 column variables, and this is confirmed below:

```r
c(ncol(train), ncol(test))
```

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

However, this isn't the last step to preparing a useful training data set. Upon further inspection, some column variables can be argued to not being contributable to a predicting model.

```r
# infer from variable names
names(train)
```

```
##  [1] "user_name"            "raw_timestamp_part_1" "raw_timestamp_part_2"
##  [4] "cvtd_timestamp"       "new_window"           "num_window"
##  [7] "roll_belt"            "pitch_belt"           "yaw_belt"
## [10] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
## [13] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [16] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [19] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [22] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [25] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [28] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [31] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [34] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [37] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [40] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [43] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [46] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [49] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [52] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [55] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [58] "magnet_forearm_z"     "classe"
```

```r
# examine structure
str(train[, 1:10])
```

```
## 'data.frame':    19622 obs. of  10 variables:
##  $ user_name           : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1: int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 13230
##  $ raw_timestamp_part_2: int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484434
##  $ cvtd_timestamp      : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window          : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window          : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt           : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt    : int  3 3 3 3 3 3 3 3 3 3 ...
```

It can be inferred that the first six variables `user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp, new_window, num_window` are administrative, integer/factor variables, unlike the other numeric variables that serve to contribute to building a good predictive model. Thus, more variable elimination is required, bringing it down to 53 variables.

```
# eliminate first 6 columns
train <- train[, -(1:6)]
test <- test[, -(1:6)]
# check dimensions
c(ncol(train), ncol(test))
```

```
## [1] 53 53
```

Cross Validation

We set `test` set aside and split the `train` data into two sections for cross validation. We will allocate 70% of the data to train the model and 30% to validate it.

We expect that the **out-of-bag (OOB)** error rates returned by the models should be good estimate for the out of sample error rate. We will get actual estimates of error rates from the **accuracies** achieved by the models.

```
# set seed
set.seed(3433)
# split train data set
inTrain <- createDataPartition(train$classe, p = 0.7, list = FALSE)
trainData <- train[inTrain, ]
validation <- train[-inTrain, ]
# print out dimensions of each data sets
rbind(trainData = dim(trainData), validation = dim(validation), test = dim(test))
```

```
##             [,1] [,2]
## trainData  13737   53
## validation  5885   53
## test          20   53
```

Comparing Models

For this project, I choose to predict the `classe` variable with all the other variables using a **random forest (`"rf"`)** and **boosted trees (`"gbm"`)**. Finally, I will stack the predictions together using random forests ("rf") for a combined model.

First, however, we will use parallel processing capabilities to speed up the training speed, since creating four predictor models is computationally expensive.

```r
# process in parallel
library(doParallel)
registerDoSEQ()
cl <- makeCluster(detectCores(), type='PSOCK')
registerDoParallel(cl)
```

Now we are ready to fit the model predictors, but not without setting a seed first:

```r
# set seed
set.seed(62433)
# load packages
library(randomForest)
# fitting random forest model predictor and record elapsed time, printing out results
elapsedFitRF <- system.time(
    print(
        fitRF <- randomForest(classe ~ ., data=trainData, method="rf")
    )
)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = trainData, method = "rf")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.46%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3905    1    0    0    0 0.0002560164
## B   13 2643    2    0    0 0.0056433409
## C    0   12 2381    3    0 0.0062604341
## D    0    0   21 2229    2 0.0102131439
## E    0    0    4    5 2516 0.0035643564
```

```r
# fitting boosted trees model predictor and record elapsed time, printing out results
library(gbm)
elapsedFitGBM <- system.time(
    print(
        fitGBM <- train(classe ~ ., data=trainData, method="gbm", verbose=FALSE)
    )
)
```

```
## Stochastic Gradient Boosting
```

```
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa      Accuracy SD
##   1                   50      0.7498638  0.6827703  0.005630100
##   1                  100      0.8178820  0.7695040  0.005754893
##   1                  150      0.8501666  0.8104177  0.004950070
##   2                   50      0.8524162  0.8130544  0.005507407
##   2                  100      0.9023231  0.8764130  0.003608319
##   2                  150      0.9276288  0.9084563  0.003433540
##   3                   50      0.8930723  0.8646520  0.005165469
##   3                  100      0.9375999  0.9210675  0.003847560
##   3                  150      0.9571657  0.9458234  0.003154868
##   Kappa SD
##   0.007175652
##   0.007249185
##   0.006241642
##   0.006982172
##   0.004558243
##   0.004356973
##   0.006540842
##   0.004859960
##   0.003996828
##
## 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.
```

After we have trained our models, we predict:

```
# predict using model predictors and record elapsed time
elapsedPredRF <- system.time(
    predRF <- predict(fitRF, newdata=validation)
)
elapsedPredGBM <- system.time(
    predGBM <- predict(fitGBM, newdata=validation)
)

# create new dataframe for stacking predictors
predAll <- data.frame(predRF, predGBM, classe = validation$classe)
elapsedFitStacked <- system.time(
    fitStacked <- randomForest(classe ~ ., data=predAll, method = 'rf')
)
```

```
# predicting with stacked predictors
elapsedPredStacked <- system.time(
    predStacked <- predict(fitStacked, newdata=validation)
)
```

From the above, we can see that **randomForest** is the better performing algorithm with **0.46% out-of-bag (OOB) error rate**, which is what we expect the out of sample error rate to be.

Run-Time Graphical Analysis of Models

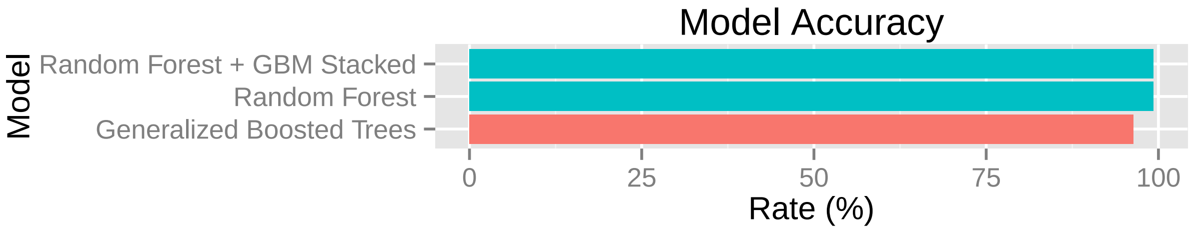
In this section we will attempt to see what is the best model to use, considering trade-offs. First, we need to the confusion matrices containing analysis of the models into variables:

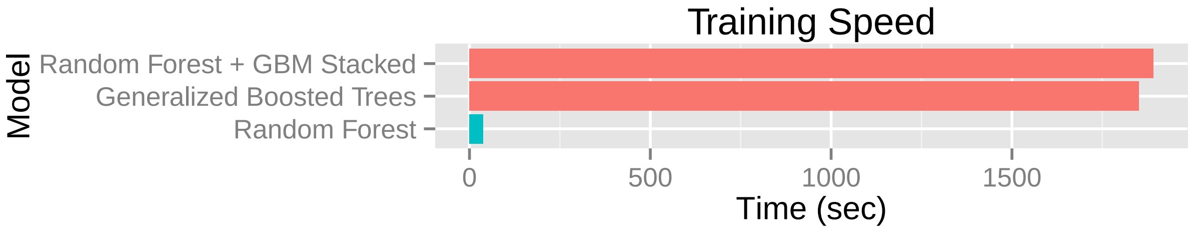
```
# confusion matrices
cmRF <- confusionMatrix(predRF, validation$classe)
cmGBM <- confusionMatrix(predGBM, validation$classe)
cmStacked <- confusionMatrix(predStacked, validation$classe)

# create table
analysis_table <- data.frame("Model" = c("Random Forest",
                                          "Generalized Boosted Trees",
                                          "Random Forest + GBM Stacked"),
                             "Accuracy" = 100 * c(cmRF$overall[[1]],
                                                  cmGBM$overall[[1]],
                                                  cmStacked$overall[[1]]),
                             "Training Speed" = c(elapsedFitRF[['elapsed']],
                                                  elapsedFitGBM[['elapsed']],
                                                  (elapsedFitRF[['elapsed']] +
                                                       elapsedFitGBM[['elapsed']] +
                                                       elapsedFitStacked[['elapsed']])),
                             "Prediction Speed" = c(elapsedPredRF[['elapsed']],
                                                    elapsedPredGBM[['elapsed']],
                                                    (elapsedPredRF[['elapsed']] +
                                                         elapsedPredGBM[['elapsed']] +
                                                         elapsedPredStacked[['elapsed']])))

names(analysis_table) <- c('Model', 'Accuracy', 'Training Speed (sec)', 'Prediction Speed (sec)')

# round numeric columns
analysis_table[, 2:4] <- round(analysis_table[, 2:4], digits = 2)

# display table nicely
kable(analysis_table,
      align = "c")
```

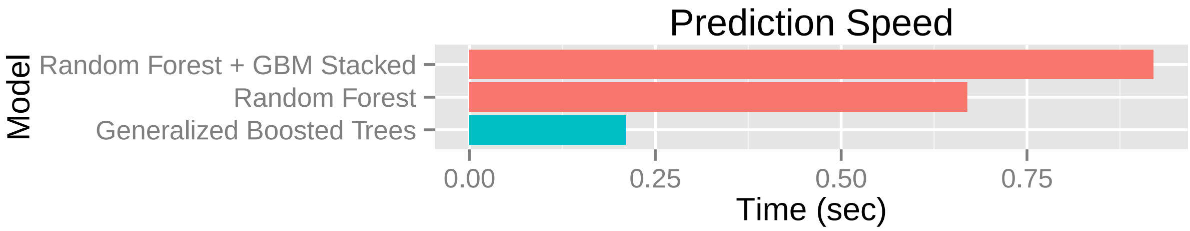
Model	Accuracy	Training Speed (sec)	Prediction Speed (sec)
Random Forest	99.27	37.87	0.67
Generalized Boosted Trees	96.35	1851.62	0.21
Random Forest + GBM Stacked	99.27	1891.17	0.92

To better visualize the run-time results, we can also make a bar graph:

```r
library(ggplot2)
# accuracy comparisons
accuracy_plot <- ggplot(transform(analysis_table,
                                  Model = reorder(Model, Accuracy)),
                        aes(x = Model, y = Accuracy)) +
    geom_bar(stat="identity",
             aes(fill = Accuracy == max(Accuracy)),
             position=position_dodge()) +
    scale_fill_discrete(guide = 'none') +
    labs(x = 'Model',
         y = 'Rate (%)',
         title = 'Model Accuracy') +
    coord_flip()
# training speed comparisons
train_speed_plot <- ggplot(transform(analysis_table,
                                     Model = reorder(Model, analysis_table[, 3])),
                           aes(x = Model, y = analysis_table[, 3])) +
    geom_bar(stat="identity",
             aes(fill = analysis_table[, 3] == min(analysis_table[, 3])),
             position=position_dodge()) +
    labs(x = 'Model',
         y = 'Time (sec)',
         title = 'Training Speed') +
    scale_fill_discrete(guide = 'none') +
    coord_flip()
# prediction speed comparisons
pred_speed_plot <- ggplot(transform(analysis_table,
                                    Model = reorder(Model, analysis_table[, 4])),
                          aes(x = Model, y = analysis_table[, 4])) +
    geom_bar(stat="identity",
             aes(fill = analysis_table[, 4] == min(analysis_table[, 4])),
             position=position_dodge()) +
    labs(x = 'Model',
         y = 'Time (sec)',
         title = 'Prediction Speed') +
    scale_fill_discrete(guide = 'none') +
    coord_flip()
# plot all three at once
library(gridExtra)
grid.arrange(accuracy_plot,
             train_speed_plot,
             pred_speed_plot,
             ncol = 1)
```







As one can see from above, the *best accuracy rate* belongs to **Random Forest and GBM stacked together** and **Random Forest** - both accuracy values are identical.

The *shortest training speed* belongs to **Random Forest**, by a huge margin.

The *shortest prediction speed* belongs to **Generalized Boosted Trees**, but only by a matter of seconds, so the difference is trivial.

Result

Given the analysis presented, there is no doubt that the best accuracy combined with best time efficiency belongs to **Random Forest**. Thus this will be our model of choice in predicting the `test` set.

```r
print(
    test_result <- predict(fitRF, test)
)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

```r
# save results into separate files in appropriate directory
source('./pml_writing_files.R')
pml_write_files(as.character(data.frame(test_result)$test_result))
```