Using Machine Learning to Determine Exercise Type

Paul Cappa

January 18, 2016

1. Synopsis

The objective of this study is to predict which exercise is being performed based on given historic information. We will explore the application of three different models and determine if single model or a voting combination of the models should be used for or final predictions. Based on our findings, we will apply our model(s) to dataset where we do not know the actual exercise performed.

1.1 Background Information

1.1.1 Practical Machine Learning Project Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har. [1]

1.1.2 Human Activity Recognition

Human Activity Recognition - **HAR** - has emerged as a key research area in the last years and is gaining increasing attention by the pervasive computing research community (see picture below, that illustrates the increasing number of publications in HAR with wearable accelerometers), especially for the development of context-aware systems. There are many potential applications for HAR, like: elderly monitoring, life log systems for monitoring energy expenditure and for supporting weight-loss programs, and digital assistants for weight lifting exercises. [2]

2. Preparing the Data for Analysis

Before we can explore the data, we must do some housekeeping, such as loading the necessary libraries, loading the training and testing data, eliminating unnecessary variables and observations. We must also split the training data into training and validation sets. We'll use 75% of the for training, 25% for validation.

Let's begin by loading the necessary libraries.

```
library(ggplot2)
library(lattice)
library(caret)
library(AppliedPredictiveModeling)
library(pgmm)
library(survival)
```

```
library(randomForest)
library(rpart)
library(gbm)
```

Next, we set the seed.

```
set.seed(9807)
```

Now we establish the names of the dataset and cache files.

```
training.file <- ".\\pml-training.csv"      # Training data filename.
testing.file <- ".\\pml-testing.csv"        # Testing data filename.
rf.RDS.file <- "rf.Fit.RDS"                 # Random forest cache filename.
rpart.RDS.file <- "rpart.Fit.RDS"           # Recursive partition cache filename.
gbm.RDS.file <- "gbm.Fit.RDS"               # Teneralized boost cache filename.
```

Next, we load the training and testing data.

```
masterTraining <- read.csv(training.file, na.strings = c("NA", ""))
testing <- read.csv(testing.file, na.strings = c("NA", ""))
```

Now we filter out any observations with "new_window" set to "yes". This class of observations do not appear in the testing set and are therefore an unnecessary distraction to the models.

```
new_window <- which(masterTraining$new_window == "yes")
masterTraining <- masterTraining[-new_window,]
```

Time to split the training data into training and validation. We'll use 75% of the data for training and 25% for validation.

```
inTrain = createDataPartition(masterTraining$classe, p = 3/4)[[1]]
training <- masterTraining[inTrain,]
validation <- masterTraining[-inTrain,]
```

Next, we find all of the variables with very little data. We are using a threshold of 90% of the variable are NA's. At the same time, we are eliminating the first six variables, as they do not contain useful information.

```
r <- nrow(training) * 0.90
keep <- vector(mode="numeric", length=0)

for (c in 7:ncol(training)) {
    if (sum(is.na(training[,c])) <= r) {
        keep <- c(keep, c)
    }
}
```

Now that we've identified the variables to eliminate, we remove them from all three datasets (training, validation, and testing).

```
training <- training[,keep]
validation <- validation[,keep]
testing <- testing[,keep]
```

3. Exploring the Different Models

The models under consideration are *Random Forest*, *Recursive Partitioning*, and *Generalized Boosting*. Ideally, we'd like to use a voting algorithm based on the effectiveness of all three models. Because the models take time to process, we will used cached results if they are available. If not, then we will generate the model and create the cache.

3.1 Generate the Random Forest Model

Check to see if the cache file exists. If it does, use that data. If not, then run *Random Forest* model. Once we have fit the model, then run the prediction on the validation data and create a confusion matrix.

```
if (file.exists(rf.RDS.file)) {
    rf.Fit <- readRDS(rf.RDS.file)
} else {
    rf.Fit <- train(classe ~ ., data=training, method="rf")
    saveRDS(rf.Fit, rf.RDS.file)
}

val.rf.Pred <- predict(rf.Fit, validation)
val.rf.Matrix <- confusionMatrix(validation$classe, val.rf.Pred)
val.rf.Matrix$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1367    0    0    0    0
##          B    0  928    1    0    0
##          C    0    0  838    0    0
##          D    0    0    6  779    1
##          E    0    0    0    0  882
```

3.2 Generate the Recursive Partition Model

Check to see if the cache file exists. If it does, use that data. If not, then run *Recursive Partition* model. Once we have fit the model, then run the prediction on the validation data and create a confusion matrix.

```
if (file.exists(rpart.RDS.file)) {
    rpart.Fit <- readRDS(rpart.RDS.file)
} else {
    rpart.Fit <- train(classe ~ ., method="rpart", data=training)
    saveRDS(rpart.Fit, rpart.RDS.file)
}

val.rpart.Pred <- predict(rpart.Fit, validation)
val.rpart.Matrix <- confusionMatrix(validation$classe, val.rpart.Pred)
val.rpart.Matrix$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1237   27  100    0    3
##          B  379  316  234    0    0
##          C  371   31  436    0    0
##          D  372  142  272    0    0
##          E  119  120  235    0  408
```

3.3 Generate the Generalized Boost Model

Check to see if the cache file exists. If it does, use that data. If not, then run *Generalized Boost* model.

The Generalized Boosting Model requires some additional steps not performed when processing the other models. The model returns a table which must be converted into a data frame. We then must compute the prediction, by converting the data into A, B, C, D, E's. This is a bit of an awkward process, but in the end, we will have a vector we can use.

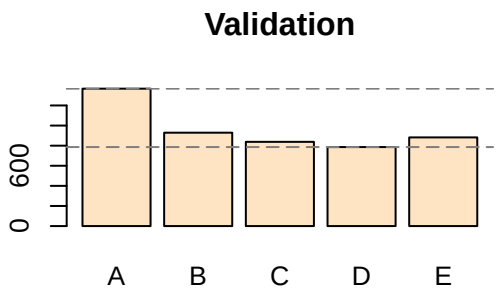
Once we have fit the model, then run the prediction on the validation data and create a confusion matrix.

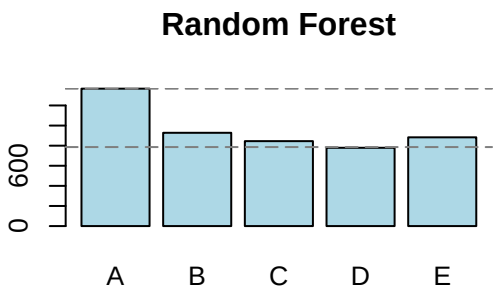
```
if (file.exists(gbm.RDS.file)) {
    gbm.Fit <- readRDS(gbm.RDS.file)
} else {
    gbm.Fit <- gbm(classe ~ ., data=training, shrinkage=0.01)
    saveRDS(gbm.Fit, gbm.RDS.file)
}

val.gbm.Pred <- predict(gbm.Fit, validation, n.trees=100)
gbm.df <- as.data.frame(val.gbm.Pred)
names(gbm.df) <- c("A", "B", "C", "D", "E")

result <- vector(mode="character", length=0)
for (i in 1:nrow(gbm.df)) {
    j <- which(gbm.df[i,]==max(gbm.df[i,]))
    result <- c(result, names(gbm.df)[j])
}

val.gbm.Pred <- factor(result)
val.gbm.Matrix <- confusionMatrix(validation$classe, val.gbm.Pred)
val.gbm.Matrix$table
```

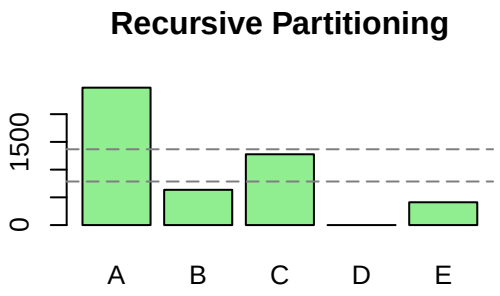
```
##           Reference
## Prediction    A    B    C    D    E
##          A 1071   37   96  145   18
##          B  309  399  153   68    0
##          C  198   47  521   72    0
##          D  222   22   78  464    0
##          E  109  180   81   79  433
```

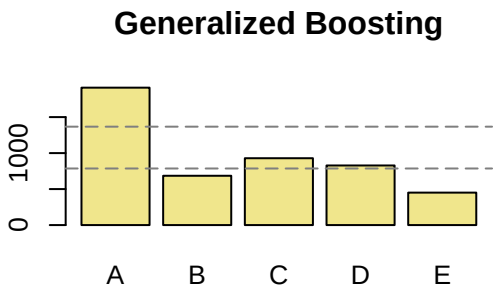
3.4 Comparing Model Performance

Now comes the moment of truth, as we will compare how our three models performed. Let's visually compare the distributions of the results. This should give us a quick peek into the differences (Note, the dashed lines, on all plots, represent the minimum and maximum from the *Validation* distribution).









It appears *Random Forest* is quite similar to the *Validation* data, whereas, *Recursive Partitioning* and *Generalized Boosting* are quite different. Now let's look at the model accuracies.

```
## Accuracies for:
##     Random Forest:          99.83%
##     Recursive Partitioning: 49.92%
##     Generalized Boosting:   60.14%
```

Our numbers confirm our visual interpretation.

3.5 Out of Sample Error Rate

The out of sample error rate for each model is calculated below:

```
## Out of Sample Error Rate for:
##     Random Forest:          0.17%
##     Recursive Partitioning: 50.08%
##     Generalized Boosting:   39.86%
```

4. Shall We Take a Vote?

As we can see, based on the effectiveness on the models, the Random Forest model is, by far, the most accurate. However, we may not be ready to toss out the other two models. From our confusion matrix,

we can see the Random Forest is not 100% accurate. Maybe there's a possibility to combine the *Recursive Partitioning* and *Generalized Boosting* models as votes that can override the *Random Forest* model.

To check this, we will create a data frame with predictions from the three models, along with the the actual validation data. We will then look through each prediction where the *Random Forest* does not match the Validation data. If the combined predictions from *Recursive Partitioning* and *Generalized Boosting* models have the correct answer, then we have a chance for success.

```
df <- data.frame(rf=val.rf.Pred, rpart=val.rpart.Pred, gbm=val.gbm.Pred,
        classe=validation$classe)

t <- vector(mode="numeric", length=0)
for (i in 1:nrow(df)) {
    if ((df[i,1] != df[i,4]) &
        ((df[i,2] == df[i,4]) | (df[i,3] == df[i,4]))) {
            t <- c(t, i)
    }
}
df[t,]
```

```
##      rf rpart gbm classe
## 3172  C     C   D      D
## 3335  E     B   D      D
## 3495  C     C   D      D
## 3496  C     C   D      D
```

We can clearly see there are no instances where the *Recursive Partitioning* and *Generalized Boosting* do not agree when the *Random Forest* prediction is incorrect. Voting is a bust!.

5. Applying Model to Twenty Test Cases

Because a voting algorithm offers us no benefit, we will simply use the model generated with the *Random Forest* to predict on the "testing" dataset containing the twenty test cases.

```
tst.rf.Pred <- predict(rf.Fit, testing)
```

. . . and the results are:

```
tst.rf.Pred
```

```
##  [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
```

6. References

1. Caffo, B.; Leek, J, Peng, R. **Practical Machine Learning, Course Project**, *Course Project Instructions, Background.* Coursera, Johns Hopkins University, Data Science Specialization.

2. Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. **Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements.** Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.