Human Activity Recognition: Machine Learning Prediction Assignment

Rohit Padebettu

11/11/2016

Synopsis

In this project, we propose a machine learning based Stochastic Gradient Boosting classifier to classify the activities of 6 human subjects performing 5 different activities wearing accelerometers mounted on their waist, left thigh, right arm, and right ankle. The classifier uses a Gradient Boosted Algorithm of 950 decision trees with an interaction depth of 4 to get to an accuracy of 99.91%.

In this experiment, six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

More information about this can be found at Human Activity Recognition.

Data Source

The data used to build this machine learning classifier and the data for the prediction has been obtainined from the sources below:

- Training Data Set
- Testing Data Set

The source for both these datasets is again Human Activity Recognition

Loading the Data

We begin our analysis by first loading a set of R libraries which aid us in various steps of our analysis.

```
suppressPackageStartupMessages(library(caret))      ## Workhorse for ML setup and algo selection
suppressPackageStartupMessages(library(doParallel)) ## Library to allow parallel multicore processin
suppressPackageStartupMessages(library(gbm))        ## Library for the ML algo used
suppressPackageStartupMessages(library(knitr))     ## Library for producing this document
```

We then download the data from the links above into our working directory and load it into out workspace.

```
trainingurl <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
#download.file(trainingurl,"pml-training.csv",method="curl")

testingurl <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
#download.file(testingurl,"pml-testing.csv",method="curl")

training<-read.csv("pml-training.csv",stringsAsFactors = FALSE)
testing<-read.csv("pml-testing.csv",stringsAsFactors = FALSE)
```

Exploring and Processing the Data

Once the data is loaded into the system, we run some basic data exploration commands to get an idea about the size of the dataset and its structure.

```
    dim(training)
```

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

```
    dim(testing)
```

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

We see that the training set has `19622` observations and `160` attributes whereas the testing set has `20` observations and `160` attributes. The final attribute in the training data set is `classe` which gives the classification of the activity being performed by the subject identified by `user_name`. We will use this attribute to build and train our classifier.

The testing dataset doesn't have the `classe` attribute, but includes the `problem_id` attribute in its place. We will run our trained classifier on this dataset to predict the `classe` attribute for each of the 20 observations using the data from the other attributes for each `problem_id`

Cleaning the data

We run the `str()` command on both these data sets (we don't print the results here to keep the document brief) and observe that a lot of the `160` attributes have missing `NA's` or `blanks`. We proceed to clean the training dataset to remove these attributes so as to commence our work on a relatively clean data set.

```
# Converting the classe column to a factor in the training data set
training$classe=as.factor(training$classe)

# Removing cols with mostly NA's
NACols<-apply(training,MARGIN = 2,FUN = function(X) sum(is.na(X)))
training_sub<-training[,names(NACols[NACols<1000])]

# Removing cols with mostly blank characters
BlnkCols<-apply(training,MARGIN = 2,FUN = function(X) sum(X==""))
BCols<-BlnkCols[!is.na(BlnkCols)]
training_set<-training_sub[,names(BCols[BCols<1000])]
```

Since we do not want our classifer to make its predictions based on the subject name or the time of the activity, we also remove the identifying attributes for subject and the time attributes. After this we end up with a set of about `54` attributes on which we can start building our model.

```
# Removing raw timestamp and name columns
remCols<-c("X","user_name","raw_timestamp_part_1","raw_timestamp_part_2",
           "cvtd_timestamp","new_window")
training_set<-training_set[,!(names(training_set) %in% remCols)]
```

We further make the same transformations on our testing dataset so as to be able to make predictions using a consistent structure.

```
# Removing same cols from testing set
testing_set<-subset(testing,select = c(names(training_set[,-c(54)]),"problem_id"))
```

Building the Classifier

Prepare training data

We prepare our `training_set` data for modelling by splitting it into two datasets:

- `p_training`
- `p_testing`

The `p_training` data set will be used to train and build our classifier, while the `p_testing` dataset will be used as on Out of Sample Test for the classifier **before** we test the classifier on the original `testing_set` which has no classification information.

```
set.seed(1234)
inTrain = createDataPartition(training_set$classe, p = 3/4)[[1]]
p_training = training_set[ inTrain,]
p_testing = training_set[-inTrain,]
```

Setting Parallel Processing

Since we intend to run a large number of model iterations to be able to select the best performing model,it is quite possible for the training process to take hours to run if we ran it on a single core. We therefore make use of the `doParallel` package and parallel processing capabilities of `caret` package to run our simulations on multiple cores on our machine.

```
cl <- makeCluster(detectCores()-1)
registerDoParallel(cl)
```

Gradient Boosting Model

We intend to use the Stochastic Gradient Boosting Model to build our classifier. *Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do. For more information please check out* Wikipedia

Setting Tuning Parameters for GBM

- The `caret` package also allows us to `preprocess` the training data, which we use to `center` and `scale` the data in this case.
- The `caret` package allows us to set tuning parameters via `tuneGrid` to build multiple models with various iterations of the tuning parameters.
- The `caret` package also allows us to control the training and perform `Repeated Cross Validations` via the `trainControl()` function. We perform 5-fold Cross Validation and repeat it 3 times each in this case.

For more information on tuning options for the `caret` package please visit Max Kuhn Caret Package Tutorial

We prepare the setup for the model as shown in the code below.

```
set.seed(1234)

# Scaling and Centering the Data
preprocess<-c("center", "scale")

# Model Iterations
myGbm <- expand.grid(n.trees = seq(250,1000,100),
                     interaction.depth = 2:4,
                     shrinkage = 0.05,
                     n.minobsinnode=20)

# Cross Validation and Training Control
trainctrlg <- trainControl(method="repeatedcv",
                           number=5,
                           repeats = 3)
```

Training the Classifier

We train the `gbm` classifier using the `train()` function in `caret` package as below. We pass the Pre-Processing, Training Control and Tuning Grid options into the function itself.

Given the number of iterations of the model we are requesting and the number of cross validations to perform, this takes about `25 mins` to train and select the best model on our system. The metric we use to select the best model is `Overall Accuracy`.

```
    system.time(
      model.gbm<-train(form = classe ~ .,
                       method = "gbm",
                       data = p_training,
                       metric="Accuracy",
                       preProcess = preprocess,
                       trControl = trainctrlg,
                       tuneGrid = myGbm
      )
  )
  stopCluster(cl)
  saveRDS(model.gbm,file="gbmmodel.rds")
```

Classifier Statistics

Once the model is trained using the `p_training` dataset we can print the model to see the accuracy of the various iterations. The output also shows us that the `train` function has automatically selected the best model for us and what the characteristics of such model are.

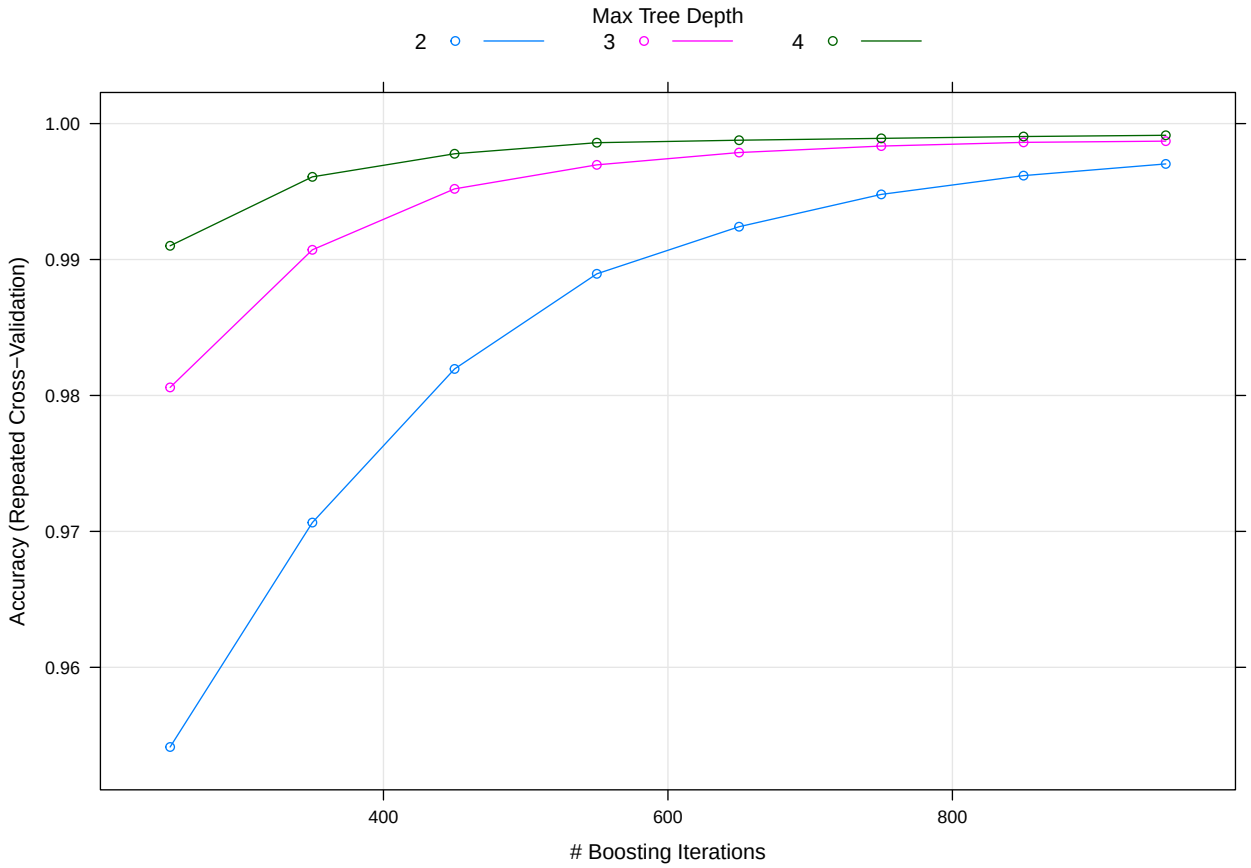
Note: *To save time, we saved the model we generated using the above code to a file locally and on Github. We load it again here to perform the other downstream analysis*

```
  model.gbm<-readRDS(file = "gbmmodel.rds")
  print(model.gbm)
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (53), scaled (53)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 11776, 11774, 11774, 11774, 11774, 11776, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   2                  250      0.9541386  0.9419739
##   2                  350      0.9706487  0.9628641
##   2                  450      0.9819502  0.9771658
##   2                  550      0.9889483  0.9860202
##   2                  650      0.9924132  0.9904033
##   2                  750      0.9947911  0.9934113
##   2                  850      0.9961726  0.9951589
##   2                  950      0.9970332  0.9962475
##   3                  250      0.9805913  0.9754469
##   3                  350      0.9907147  0.9882545
##   3                  450      0.9951986  0.9939268
##   3                  550      0.9969651  0.9961612
##   3                  650      0.9978710  0.9973072
##   3                  750      0.9983466  0.9979088
##   3                  850      0.9986184  0.9982525
##   3                  950      0.9987090  0.9983671
##   4                  250      0.9910090  0.9886268
##   4                  350      0.9960818  0.9950439
##   4                  450      0.9977804  0.9971925
##   4                  550      0.9985958  0.9982238
##   4                  650      0.9987769  0.9984530
##   4                  750      0.9989129  0.9986250
##   4                  850      0.9990488  0.9987968
##   4                  950      0.9991393  0.9989113
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.05
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 20
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 950,
##  interaction.depth = 4, shrinkage = 0.05 and n.minobsinnode = 20.
```

We can also plot the model iterations and see how the accuracies of each model vary with the various tuning parameters.

```
    plot(model.gbm)
```

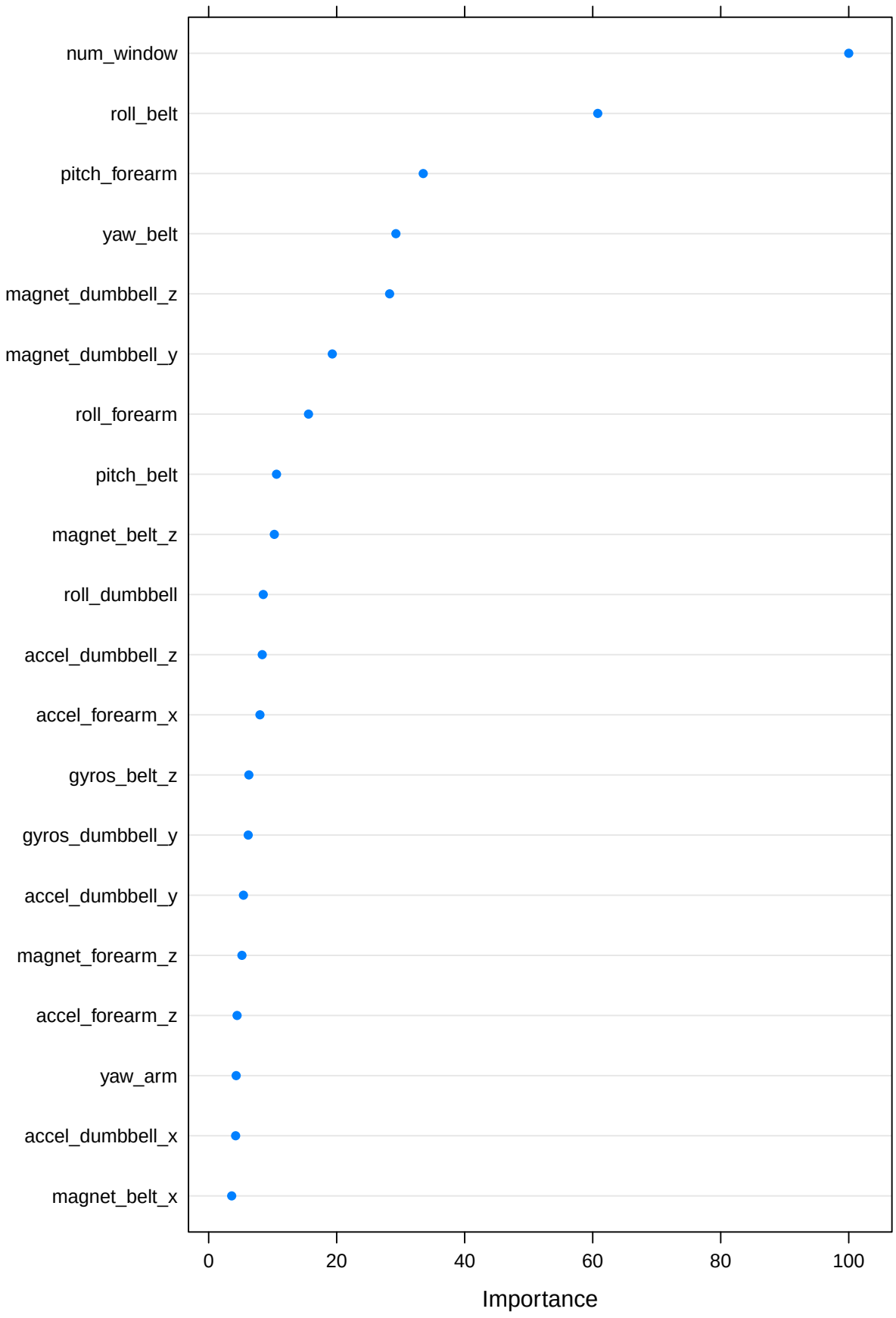


We can see from the above data and plot that the selected model with 950 trees has an `Accuracy` of about **99.91%** with `Kappa` about **0.9989**

Finally we also plot the `Importance` of attributes in the model for the top 20 or so attributes

```
dotPlot(varImp(model.gbm))
```

```
## Loading required package: plyr
```


num_window
roll_belt
pitch_forearm
yaw_belt
magnet_dumbbell_z
magnet_dumbbell_y
roll_forearm
pitch_belt
magnet_belt_z
roll_dumbbell
accel_dumbbell_z
accel_forearm_x
gyros_belt_z
gyros_dumbbell_y
accel_dumbbell_y
magnet_forearm_z
accel_forearm_z
yaw_arm
accel_dumbbell_x
magnet_belt_x
0
20
40
60
80
100
Importance

Out of Sample Accuracy

To calculate the **Out Of Sample** accuracy of the trained model we predict the classifications using the `p_testing` set we created initially and generate the confusion matrix to compare against the actual classifications. **As can be seen, the accuracy of this classifier is very high on the testing set as well!**

```
pred.gbm<-predict(model.gbm,p_testing)
confusionMatrix(p_testing$classe,pred.gbm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  949    0    0    0
##          C    0    0  855    0    0
##          D    0    0    0  804    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9992, 1)
##     No Information Rate : 0.2845
##     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             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Classifying the Testing set provided

Now that we have trained the classifier and also performed an out of sample test, we can now use it to perform the classification on the original unclassified testing set with `20 observations` that was provided.

Note: *We has processed the original testing set to remove the columns with NA's, Blanks, Identifying information as well as time attributes. We called that the* ***`testing_set`***

```
pred.test.gbm<-predict(model.gbm,testing_set)
summary(pred.test.gbm)
```

```
## A B C D E
## 7 8 1 1 3
```

```
    testing_pred<-cbind(testing_set,pred.test.gbm)
    kable(t(testing_pred[,c(54,55)]))
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
problem_id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
pred.test.gbm	B	A	B	A	A	E	D	B	A	A	B	C	B	A	E	E	A	B	B	B

Conclusion

We began by attempting to build and train a classifier to classify 5 different human activities of six subjects. We used the Stochastic Gradient Boosting model to build the classifier. We tested out a number of iterations of the parameters for the model and performed several cross validations to be able to select the best set of parameters and attributes. The resulting model performed had an accuracy of about 99.91% on the training set and had a perfect classification score on the out of sample testing set. The model was later applied to the unclassified testing set provided and scored a perfect 20/20 there as well!