Practical Machine Learning: Peer Assessment

WR

December 21, 2015

Overview

This prediction algorithm looks at the weight lifting data by Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. and described at http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har). It intends to predict whether a specific data point relates to 1 of 5 different motion pathways (1 representing the correct way and 4 representing common mistakes) that was performed by 6 different participants and with 10 repetitions each.

The properties of the predictor led to pre-processing the data for data compression and focusing on relevant predictors. The prediction algorithm chosen was the Random Forest algorithm, chosen for its accuracy, as in submitting this assignment, the upload of each character answer {A, B, C, D, E} is being graded on accuracy. Cross Validation was done via using `createDataPartition(...)` and splitting the provided "training" dataset randomly between training and testing datasets. The in sample and out of sample error was computed and used to compare model / prediction options. The results upon submision of this assignment was 20/20 - 100% correct prediction!

Please refer to the PML_Proj_RScript.R file in the GitHub Repository which contains all the code used in building and evaluating the prediction algorithm options.

Data Load

The data was loaded using read.table, comma-deliminated, and setting `na.strings = c("NA", "#DIV/0!", "")` .

Pre-Processing

First, in viewing the training data, I noticed lots of NAs and more generally that NA strings should be on `na.strings = c("NA", "#DIV/0!", "")` thus updating the `read.table` accordingly.

Next I noticed for 1 column in the first 250 rows had 244 NAs and in the first 500 rows had 490 NAs using `sum(is.na(pml.training.orig$max_yaw_dumbbell))` . Since it is difficult to use a column with little data in it for modelling or prediction, I removed these columns from the dataset using `pml.training.vFilNA <- pml.training.orig[, (apply(pml.training.orig,2, function(x){sum(!is.na(x))/length(x)})) > 0.99]` , reducing the columns from 160 down to 60.

Further, looking at the training data, I noticed the first few columns in essence sequentially ordered the data, which as the `classe` variable in the dataset was also in essence sequentially ordered by the 5 motion pathways {A, B, C, D, E} for the bicep curl. When left in the model, these first few columns resulted in non-useful models (take first X rows == A, take next Y rows == B, etc.). Thus, I removed these 6 columns { `select(., -X, -grep("time", colnames(pml.training.vFilNA)), -grep("window", colnames(pml.training.vFilNA)))` } resulting in 53 predictor variables and 1 result variable (classe).

Components of a Predictor

The components of the predictor chosen here are described next.

Question

Can I use on-body sensing data to predict which of 5 motion pathways were used to do a bicep curl?

Input Data

Data was provided from the following research (previously mentioned): http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

Features

The features that were used as predictors were a subset of the variables provided in the original dataset. From the 160 original variables, I reduced these variables, as described in the pre-processing section of this write-up, down to 54 variables, by either eliminating columns containing little information or containing non-relevant information (when the measurement was taken or the order in which the measurement was taken).

This allowed for data compression, while retaining relevant information. In addition, as the columns were removed with lots of NAs vs. rows with lots of NAs, I compressed data while being careful to not throw away information unneccesarily. All of these are properties of good predictors.

Prediction Algorithm

The prediction algorithm chosen was a Random Forest using model form `train(classe~., data=training, method="rf")` . Some benefits of Random Forests is Accuracy at the expense of Speed, Interpretability, and Overfitting. To improve the speed, I originally subsetted the data by test subject, in essence splitting the data into 6 subsets of data. As seen in the next section on Evaluation / Cross Validation / In Sample and Out of Sample Error, this improvement in speed significantly degraded accuracy, so I instead went with random subsampling to improve accuracy.

Evaluation / Cross Validation / In Sample and Out of Sample Error

Cross Validation was done via using `createDataPartition(...)` and splitting the provided “training” dataset randomly between training and testing datasets. The in sample and out of sample error was computed and used to compare model / prediction options.

First, I wanted to explore the prediction model accuracy if the `rf` was run against one person and then that model used to predict the other individuals. This led to a prediction algorithm on one person’s weight lifting exercises with very low in sample error (<1%), but very high out of sample error (50-90%) when applied to the other individuals. This relates most likely overfitting of the model to the uniqueness of how any individual does the motion pathways for the weight lifting exercise, the low predictive value when using non-randomization of sample data, and also the consideration of how many individuals do you need to accurately predict across a population if each individual really does the exercises that uniquely.

Next, when using random subsampling, I first created a subsample `createDataPartition(...)` for 5% of the samples in the training dataset, with resulting in sample error (11.6%) and out of sample error (10.6%) when applied to the 95% of sample testing dataset and out of sample error (4.5-15.5%) when applied to each individual. Interestingly, using random subsampling even when creating the predictor with only 5% of the data points, I obtained a better out of sample error (10.6%, with a range of 4.5-15.5%) than when estimating based on any one of the individuals which had an out of sample error rate (50-90%).

Next, again when using random subsampling, I created a subsample `createDataPartition(...)` for 25% of the samples in the training dataset, with resulting in sample error (2.34%) and out of sample error (2.28%) when applied to the 75% of sample testing dataset and out of sample error (0.76-3.03%) when applied to each individual. Even though only using 25% or the training data provided to train the dataset, the accuracy has been significantly improved from the non-randomized, subject-based samples, where out of sample error rates were 50-90%, and from the randomized sampling using only 5% of the samples, where out of sample error rates were 10.6%. With 25% of the samples in the training dataset, the out of sample error rates have been reduced to 2.28%.

Next, again when using random subsampling, I created a subsample `createDataPartition(...)` for 75% of the samples in the training dataset, with resulting in sample error (<1%) and out of sample error (<1%) when applied to the 25% of sample testing dataset and out of sample error (<1%) when applied to each individual. This is the final model (`rf`) I used for my prediction algorithm, split of training (75%) / testing (25%), and cross-validation (training model on 75% of available data, then evaluating on test datsaset) resulting in <1% in sample error amd <1% out of sample error.

Output

```
> print(modelFit.p75$finalModel)

Call:
 randomForest(x = x, y = y, mtry = param$mtry)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 29

        OOB estimate of  error rate: 0.67%
Confusion matrix:
     A    B    C    D    E  class.error
A 4181    3    0    0    1 0.0009557945
B   18 2824    6    0    0 0.0084269663
C    0   15 2541   11    0 0.0101285547
D    0    2   29 2379    2 0.0136815920
E    0    0    5    6 2695 0.0040650407
```

Results

The results upon submision of this assignment was 20/20 - 100% correct prediction!

```
> modelFit.p75.predict.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
```

Discussion

Ideally, when creating a prediction algorithm, you want it to be - Interpretable - Simple - Fast - Accurate - Scalable Here, as we were being graded on accuracy in the submission of each result, accuracy was given a higher weighting in the predicton algorithm choice of Random Forest. The Random Forest algorithm is accurate and simple, but lacks the interpretability. scalability, and fastness of some other models. It does do built-in bootstrapping internal to the method. Again, due to the nature of being graded on each submission for the 20 sample testing submission dataset, accuracy was weighted the highest in prediction model choice.