Practical Machine Learning Project

B Mahoney

March 24, 2017

Project Practical Machine Learning - Coursera March, 2017

This document summarizes the work done to complete the Course Project for the Coursera Practical Machine Learning course offered in March, 2017. The purpose of this project is to apply predictive analytics to a set of data captured by accelerometers (referred to as “on-body sensing” devices) worn by subjects executing a number of Unilateral Dumbbell Biceps Curls in five different ways, corresponding to five classes, A to E, with class A representing the correct exercise method. Classes B to E were variations of incorrect methods for conducting the excercise. Analytic models applied to the data provided were then used to predict the manner in which the specific exercise was performed.

The following are excerpts from the Instructions and overview provided on the course website:

Instructions

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 dumbell of 6 participants.

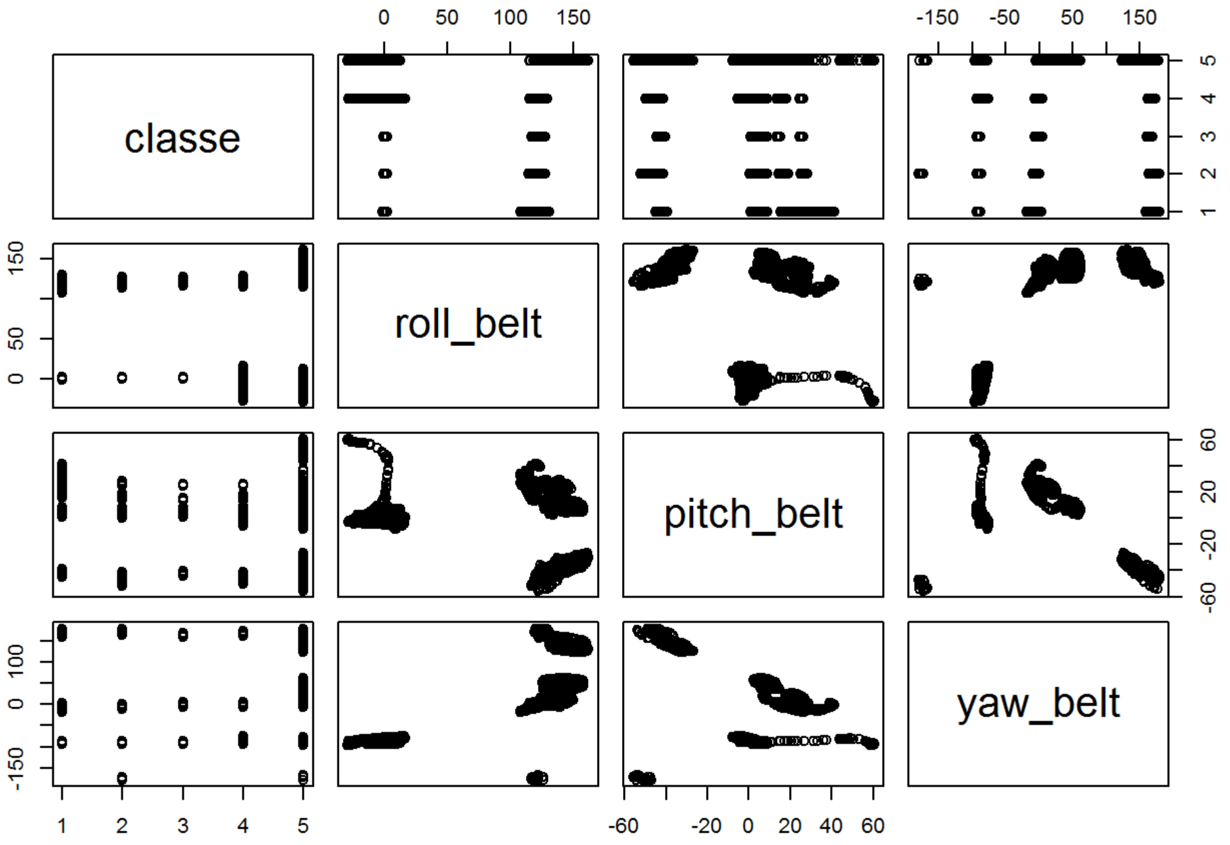
Review criteria - What you should submit

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. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Data Investigations

Two separate datasets were provided for this project. The first is a set of 19622 records to be used to train the selected predictive models. 159 predictors were included in the training set provided, along with the “classe” variable where the actual manner of execution (A, B, C, D, or E) was noted.

The predictor data is voluminous and complex. Graphical representations of the data that might normally produce insights do not seem to work well with this data. A pair plot showing the classe variable against just 3 of the 159 predictors is shown below. This and other comparisons, however, hint strongly that a number of predictors may be correlated with each other. Based on this, I decided to use pre-processing on the predictor inputs for all of the remaining analysis.



Another issue discovered in the data was the presence of a number of variables with few observations (and a majority of NA values) while the majority of variables had no instances of NA values.

A total of 67 variables having the majority of NA values were found in the data. Each of these variables had values in exactly 406 records, out of a total of 19622 observations.

My research indicated that there may be methods for dealing with sparse data like this, but I decided that I would not have time to learn enough about them to help with this project. Instead, for the purposes of the remaining analysis, the NA majority predictors were excluded from subsequent steps in the study.

Analysis

Data partitioning and cross-validation were the two ways used to estimate the accuracy of the analysis on out of sample error.

To begin with, the training set was partitioned into 3 components:

First, 30% of the records were set aside for model validation and prediction. Then 70% of the remaining records (so 49% of the total) were selected randomly for model building. Finally, 21% of the total records remained to be used to test the models.

The training data, therefore, has 9619 rows, and testing was performed against a dataset containing 4118 rows.

For modeling purposes, all the fitting was done using pre-processed variables for predictors. The training dataset provided the relationships among the subject, time, and movement variables (a total of 58 variables with the majority NA columns excluded) using the pca method (i.e. principal components analysis) and a threshold of 0.95, with results summarized below.

```
## Created from 9619 samples and 57 variables
##
## Pre-processing:
##   - centered (54)
##   - ignored (3)
##   - principal component signal extraction (54)
##   - scaled (54)
##
## PCA needed 27 components to capture 95 percent of the variance
```

Model I: Random Forest

The train function in the caret package was used for this model.

The first model used the random forest method, with repeated cross validation for five resampling iterations and three repeats. The number of variables per level (mtry) was set equal to the square root of the number of predictors, which is the default for this method.

The summary of the model is shown below. The random forest produced a model with a pretty low OOB estimate of error rate of 2.66%. This represents the accuracy of the cross-validation inherent in the model, and is a good estimate of the out of sample error expected.

```
##
## 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: 6
##
##         OOB estimate of  error rate: 2.54%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 2705   16   14    0    0  0.01096892
## B   40 1793   28    0    0  0.03653949
## C    1   31 1631   14    1  0.02800954
## D    0    0   70 1498    9  0.05009512
## E    0    0    2   18 1748  0.01131222
```

We further tested the accuracy of the model by using predictions based on the random forest model against the testing data set we created earlier (using the pre-processed predictors from the training set).

```
## [1] 0.9776591
```

```
## $table
##    b
## a      A    B    C    D    E
##   A 1162   12    0    0    0
##   B    4  777   18    0    0
##   C    5    8  696   22    1
##   D    0    0    4  646   11
##   E    0    0    0    7  745
##
## $misclass.prob
## [1] 0.02234094
```

We calculated an Accuracy measure by comparing the class predicted by the model versus the actual class from the test data. The testing data produced an Accuracy measure of 97.7659058%, or an error rate of 2.2340942%, which is quite close to the expected out of sample error reported by the random forest algorithm.

Model II: Generalized Boosted Regression Model

The second model type tested was a gbm classification model using the same pre-processed variables on a multinomial classification scheme, n.trees set to 500, and interaction.depth set to 2. We used the gbm function in the 'gbm' package to construct this model. We then used the predict.gbm function first on the training data to project the classes on the data set, and then on the testing data, and compared the predicted values against the actual values in the corresponding data sets.

The predict.gbm function produced a set of probabilities, which then needed to be converted to predicted values. These values were in turn converted into the factor values A, B, C, D, and E before comparison against the classe value in the training and testing datasets.

Shown below are the calculated Accuracy and a confusion matrix for the training data, and then on the testing data. Clearly, Model II did not perform nearly as well as Model I on either dataset.

First, the training metrics:

```
## [1] 0.6853103
```

```
## $table
##    b
## a      A    B    C    D    E
##   A 2294  527   16    0    0
##   B  244  905  298  129    0
##   C  197  410 1167  220   16
##   D    0   19   75  680  206
##   E    0    0  122  548 1546
## 
## $misclass.prob
## [1] 0.3146897
```

Here are the performance metrics on the testing dataset:

```
## [1] 0.6687712
```

```
## $table
##    b
## a     A   B   C   D   E
##   A 986 253   3   0   0
##   B 100 331 131  64   0
##   C  85 202 495  79   6
##   D   0  11  41 290  99
##   E   0   0  48 242 652
## 
## $misclass.prob
## [1] 0.3312288
```

Selected Model - Model I

Model I, the random forest model was clearly the better predictor and was selected as the final model for the project. Model I was applied against the validation dataset we set aside earlier in the initial data partition step. Here are the performance metrics on the validation dataset. The performance against the validation data is slightly better even than that for the testing data. This gives confidence that the model will produce reliable predictions outside the available sample.

```
## [1] 0.9757009
```

```
## $table
##    b
## a       A    B    C    D    E
##   A 1657   22    0    0    0
##   B    7 1107   25    0    0
##   C   10   10  997   45    4
##   D    0    0    3  909    6
##   E    0    0    1   10 1072
## 
## $misclass.prob
## [1] 0.02429907
```

Apply Model I to test data set (20 records quiz output)

We read in the 20 records from the testing data set provided. The dataset has 20 rows and 160 columns. The classe variable is missing of course, and an index column has been added (labeled ‘problem id’). The random forest model was used to predict the classe variable, based on Model I, with pre-processed variables as predictors. Here are the predictions:

```
##    predict(modrf, quizPC)
## 1                       B
## 2                       A
## 3                       B
## 4                       A
## 5                       A
## 6                       E
## 7                       D
## 8                       B
## 9                       A
## 10                      A
## 11                      B
## 12                      C
## 13                      B
## 14                      A
## 15                      E
## 16                      E
## 17                      A
## 18                      B
## 19                      B
## 20                      B
```

Appendix - R Code

```
#set working directory and read the training set in
setwd("C:/My Data/2017/Coursera")
dat=read.csv("pml-training.csv")
quiz=read.csv("pml-testing.csv")
quizn=dim(quiz)
trainn=dim(dat)[2]
trainm=dim(dat)[1]
require(pander)
```

Plot selected data

```
pairs(dat[,c(160,8:10)])
```

Check NA-dominated columns

```
na=sapply(dat,function(y) sum(length(which(is.na(y)))))
naonly=na[na>0]
naonly.ct=length(naonly)
```

Set Partitioning Parameters

```
#set parameters for data partition, testing portion of total
#validation percent of testing

buildpercent=.7
trainpercent=.7
```

Partition the data

```
#partition training set into build v validation
set.seed(3523)
require(caret)
library(caret)

inBuild=createDataPartition(dat$classe,p=buildpercent)[[1]]
buildData=dat[inBuild,]
validation=dat[-inBuild,]
#partition training into training v testing

inTrain=createDataPartition(buildData$classe,p=trainpercent)[[1]]

truetest=buildData[-inTrain,]
truetrain=buildData[inTrain,]
```

Ignore NA dominated columns

```
#preliminary models use only the subject, time and positional variables,
#ignores sparse data columns

training1=truetrain[,c(2:11,38:49,60:68,84:86,102,113:124,140,151:160)]
pcount=dim(training1)[2]
```

Pre-processing code

```
#pre-Process all 57 subject, time and movement variables
prep=preProcess(training1[,-58],method="pca",thresh=.95)
prep
#match up the validation dataset variables
testing1=truetest[,c(2:11,38:49,60:68,84:86,102,113:124,140,151:160)]
valid1=validation[,c(2:11,38:49,60:68,84:86,102,113:124,140,151:160)]
quiz1=quiz[,c(2:11,38:49,60:68,84:86,102,113:124,140,151:159)]

trainPC=predict(prep,training1[,-58])
testPC=predict(prep,testing1[,-58])
validPC=predict(prep,valid1[,-58])
quizPC=predict(prep,quiz1)
trainPC=as.data.frame(trainPC)
testPC=as.data.frame(testPC)
quizPC=as.data.frame(quizPC)
validPC=as.data.frame(validPC)
trainPC=cbind(trainPC,training1$classe)
testPC=cbind(testPC,testing1$classe)
validPC=cbind(validPC,valid1$classe)
```

Construct Model I - Random Forest

```
#random forest model on pre-processed predictors
#train on trainPC to create model, mod5
#test on validation data set

#set up the trainControl vector, for tuning
control <- trainControl(method="repeatedcv", number=5, repeats=3)
seed <- 3233
set.seed(seed)
#set up tuneGrid vector, for tuning
mtry <- sqrt(ncol(trainPC))
tunegrid <- expand.grid(.mtry=mtry)

#run the model on training dataset results using random forest and
#pre-processed variables
modrf=train(`training1$classe`~.,method="rf",data=trainPC,
           tuneGrid=tunegrid,trControl=control)
modrf$finalModel
#function to calculate accuracy of predictions
#inputs are data frame vectors, actual v predicted
#output is percent accuracy for predictions
AccCalc=function(actuals,predictions){
  acc=cbind(actuals,predictions)
  acc=transform(acc,actual=as.character(acc[,1]))
  acc=transform(acc,predicted=as.character(acc[,2]))
  acc=transform(acc,matching=(acc$actual!=acc$predicted))
  AccCalc=1-sum(acc$matching)/length(acc$matching)
}
```

Use model I to predict on test data, calculate Accuracy, Confusion

```
#Test random forest model against test data
#using pre-processed predictors (trainPC pre-process model-->testPC)
predrf=as.data.frame(predict(modrf,testPC))
#set up accuracy measure on test set

testAccrf=AccCalc(testPC$`testing1$classe`,predrf)
testAccrf

#confusion matrix function
confusion=function(a,b){
  tbl=table(a,b)
  mis=1-sum(diag(tbl))/sum(tbl)
  list(table=tbl,misclass.prob=mis)
}

cmrf=confusion(as.factor(predrf$`predict(modrf, testPC)`),as.factor(testPC$`testing1$classe`))
cmrf
```

Calculate Model II GBM

```
#model is gbm classification model

require(gbm)
model = gbm(`training1$classe`~., data=trainPC, n.trees=500,
             interaction.depth=2,distribution="multinomial")
predictiontrain = predict.gbm(model, trainPC, type="response", n.trees=500)
prediction = predict.gbm(model, testPC, type="response", n.trees=500)



#convert probabilities in prediction to values from 1 to 5 (A to B)
#function converts integers to alpha
train.pred = apply(predictiontrain, 1, which.max)
train.pred=as.data.frame(train.pred)
p.pred = apply(prediction, 1, which.max)
p.pred=as.data.frame(p.pred)

alphavalue=function(prediction){
  alphavalue=transform(prediction,letter=ifelse(prediction==1,"A",
                                              ifelse(prediction==2,"B",
                                                     ifelse(prediction==3,"C",
                                                            ifelse(prediction==4,"D",
                                                                                   "E")))))
}

predtrain=alphavalue(train.pred)
predtrain=as.data.frame(predtrain)
predgbm=alphavalue(p.pred)
predgbm=as.data.frame(predgbm)

compvalid=cbind(testPC$`testing1$classe`,predgbm)
```

```
compvalid=transform(compvalid,actual=compvalid[,1])
compvalid=transform(compvalid,actual=as.character(actual))
compvalid=transform(compvalid,predicted=as.character(compvalid[,2]))
compvalid=transform(compvalid,matching=(compvalid$actual!=compvalid$predicted))
testAcc2=1-sum(compvalid$matching)/length(compvalid$matching)
trainAccgbm=AccCalc(trainPC$`training1$classe`,as.data.frame(predtrain$train.pred.1))

cmtrain=confusion(as.factor(predtrain$train.pred.1),as.factor(trainPC$`training1$classe`))
testAccgbm=AccCalc(testPC$`testing1$classe`,as.data.frame(predgbm$p.pred.1))

cmtable=confusion(as.factor(predgbm$p.pred.1),as.factor(testPC$`testing1$classe`))
```

Metrics for Model II

```
trainAccgbm
cmtrain
```

```
testAccgbm
cmtable
```

Selected Model - test against validation data

```
predrfval=as.data.frame(predict(modrf,validPC))
validAccrf=AccCalc(validPC$`valid1$classe`,predrfval)
validAccrf
cmrfvalid=confusion(as.factor(predrfval$`predict(modrf, validPC)`),as.factor(validPC$`valid1$c
lasse`))
cmrfvalid
```

Predictions for 20 test subjects

```
predquiz=as.data.frame(predict(modrf,quizPC))
predquiz
```