Practical Machine Learning Course Project

Mike G.

4/25/2018

Loading the data

```
pml_training = read.csv('~/Documents/HopkinsDataScience/PracticalMachineLearning/pml-training.csv')
pml_testing = read.csv('~/Documents/HopkinsDataScience/PracticalMachineLearning/pml-testing.csv')
```

Background

The data come from a weight lifting dataset where participants were asked to perform one set of 10 repititions of the unilateral dumbbell biceps curl in five different fashions.

The variable 'classe' is divided into 5 factors: A, B, C, D, and E. * A is exactly according to specification * B is throwing the elbows to the front * C is lifting the dumbbell only halfway * D is lowering the dumbbell only halfway * E is throwing the hips to the front

Goal

Using a variety of machine learning techniques, we will use the model with the highest accuracy as the model to use for prediction on the testing data set. In particular, we are predicting the manner in which the subjects in the testing dataset performed the exercise (dumbbell biceps curl).

```
#load relevant libraries
library(caret, quietly = TRUE)
library(randomForest, quietly = TRUE)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(rpart, quietly = TRUE)
library(RColorBrewer, quietly = TRUE)
```

Set the seed to 1203.

```
set.seed(1203)
```

Partition the training set

We'll take the original training set and create a partition: 70% to the new training set and 30% to a testing set. The testing set will be used as cross-validation for the model built using the training set. In other words, 70% of the training dataset will be used to build the model. Then, the model will be tested on the other 30% of the data (named 'subTesting').

```
inTrain = createDataPartition(y=pml_training$classe, p=.7, list=FALSE)
subTraining = pml_training[inTrain,]
subTesting = pml_training[-inTrain,]
```

The subTraining dataset contains 13737 observations while the subTesting dataset contains 5885 observations.

Data cleaning

We need to clean the data before modeling. The first step is to remove any timestamp, name, and window data as they aren't relevant as predictors.

```
Cleaner = grep('name|timestamp|window', colnames(subTraining), value=FALSE)
subTraining = subTraining[,-Cleaner]
```

Next, we find possible non-zero variance variables in the training data. And then remove them.

```
subDataNZV = nearZeroVar(subTraining, saveMetrics=TRUE)

#grab the variables that make sense to use as predictors
subNZVariables = names(subTraining) %in% c("new_window", "kurtosis_roll_belt", "kurtosis_picth_belt",
"kurtosis_yaw_belt", "skewness_roll_belt", "skewness_roll_belt.1", "skewness_yaw_belt",
"max_yaw_belt", "min_yaw_belt", "amplitude_yaw_belt", "avg_roll_arm", "stddev_roll_arm",
"var_roll_arm", "avg_pitch_arm", "stddev_pitch_arm", "var_pitch_arm", "avg_yaw_arm",
"stddev_yaw_arm", "var_yaw_arm", "kurtosis_roll_arm", "kurtosis_picth_arm",
"kurtosis_yaw_arm", "skewness_roll_arm", "skewness_pitch_arm", "skewness_yaw_arm",
"max_roll_arm", "min_roll_arm", "min_pitch_arm", "amplitude_roll_arm", "amplitude_pitch_arm",
"kurtosis_roll_dumbbell", "kurtosis_picth_dumbbell", "kurtosis_yaw_dumbbell", "skewness_roll_dumbbell",
"skewness_pitch_dumbbell", "skewness_yaw_dumbbell", "max_yaw_dumbbell", "min_yaw_dumbbell",
"amplitude_yaw_dumbbell", "kurtosis_roll_forearm", "kurtosis_picth_forearm", "kurtosis_yaw_forearm",
"skewness_roll_forearm", "skewness_pitch_forearm", "skewness_yaw_forearm", "max_roll_forearm",
"max_yaw_forearm", "min_roll_forearm", "min_yaw_forearm", "amplitude_roll_forearm",
"amplitude_yaw_forearm", "avg_roll_forearm", "stddev_roll_forearm", "var_roll_forearm",
"avg_pitch_forearm", "stddev_pitch_forearm", "var_pitch_forearm", "avg_yaw_forearm",
"stddev_yaw_forearm", "var_yaw_forearm")

#drop these variables
subTraining = subTraining[!subNZVariables]

#dropping the ID variable
subTraining = subTraining[c(-1)]
```

Another issue we need to deal with is variables with a high rate of missing data (NAs) and exclude them from the analysis.

```
#removing variables with too many NAs
NAthreshold = .8
NApercent = apply(subTraining, 2, function(x) sum(is.na(x)))/nrow(subTraining)
subTraining = subTraining[!(NApercent)>NAthreshold]
```

With this done, we also clean the testing sets so they are consistent with the training set.

```
#clean the testing sets like above
clean1 = colnames(subTraining)
clean2 = colnames(subTraining[, -53])
subTesting = subTesting[clean1]
pml_testing = pml_testing[clean2]
```

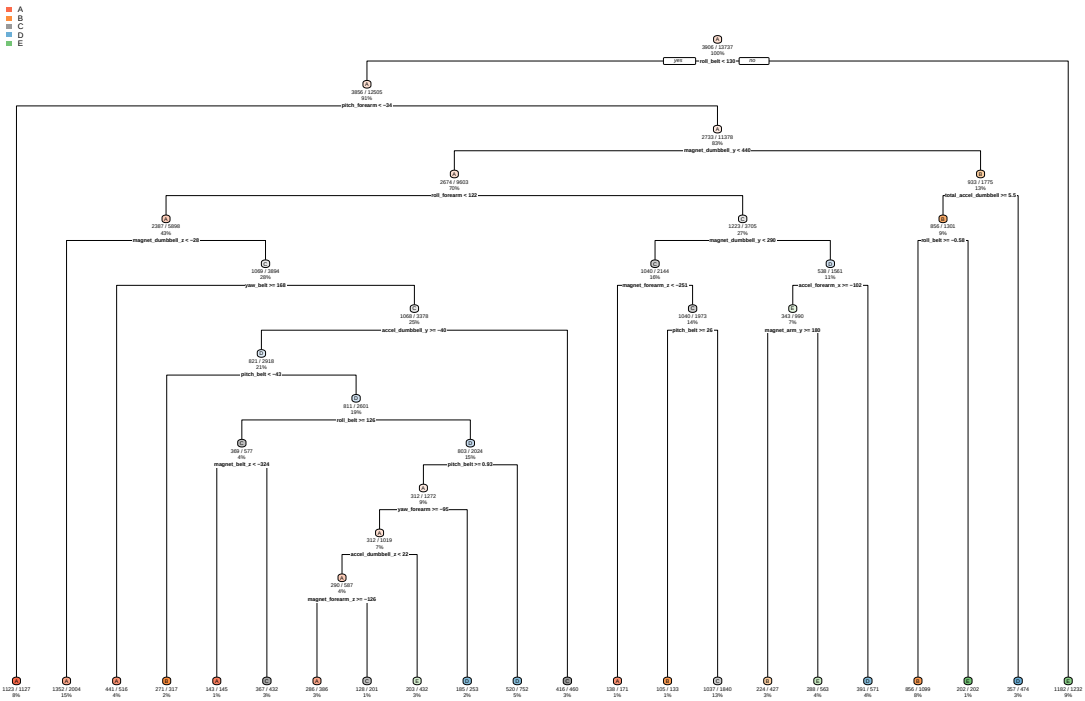
Finally, we set up the data for the classification by coercing the data into the same data type.

```
#set up for decision trees/random forest

for(i in 1:length(pml_testing)) {
  for(j in 1:length(subTesting)) {
    if(length(grep(names(subTesting[i]), names(pml_testing)[j]))==1) {
      class(pml_testing[i]) = class(subTesting[i])
    }
  }
}

pml_testing = rbind(subTesting[2, -53], pml_testing)
pml_testing = pml_testing[-1,]
```

Model 1: Decision tree

Our first model is a decision tree. We use the 'rpart' function with defaults and then plot the tree. Then, we predict the 'classe' variable on the 'subTesting' dataset. Finally, we show the confusion matrix.

```
#model #1
modDecTree = rpart(classe ~. , data=subTraining, method='class')

#look at the decision tree model
library(rpart.plot)
rpart.plot(modDecTree, extra=102, under=TRUE)
```




```
#prediction with model 1
predictionDecTree = predict(modDecTree, subTesting, type='class')
confusionMatrix(predictionDecTree, subTesting$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1472  233   16  114   34
##          B   36  585   79   30   73
##          C   46  112  834  155  110
##          D   53   84   75  595   50
##          E   67  125   22   70  815
##
## Overall Statistics
##
##                Accuracy : 0.7308
##                  95% CI : (0.7193, 0.7421)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6584
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8793  0.51361   0.8129   0.6172   0.7532
## Specificity            0.9057  0.95407   0.9129   0.9468   0.9409
## Pos Pred Value         0.7876  0.72852   0.6635   0.6943   0.7416
## Neg Pred Value         0.9497  0.89099   0.9585   0.9266   0.9442
## Prevalence             0.2845  0.19354   0.1743   0.1638   0.1839
```

```
## Detection Rate         0.2501  0.09941   0.1417   0.1011   0.1385
## Detection Prevalence   0.3176  0.13645   0.2136   0.1456   0.1867
## Balanced Accuracy      0.8925  0.73384   0.8629   0.7820   0.8471
```

With the decision tree model, there is 0.731 accuracy.

Model 2: Random forest

Now, we move to a random forest model using the 'randomForest' function with the default settings. We then predict the 'classe' variable in the 'subTesting' dataset and show the confusion matrix.

```
#machine learning model (model 2) with random forest
modRF = randomForest(classe ~ ., data=subTraining)

predictionRF = predict(modRF, subTesting, type='class')
confusionMatrix(predictionRF, subTesting$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    3    0    0    0
##          B    2 1134    7    0    0
##          C    0    2 1014    7    0
##          D    0    0    5  956    5
##          E    0    0    0    1 1077
##
## Overall Statistics
##
##                Accuracy : 0.9946
##                  95% CI : (0.9923, 0.9963)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9931
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9956   0.9883   0.9917   0.9954
## Specificity            0.9993   0.9981   0.9981   0.9980   0.9998
## Pos Pred Value         0.9982   0.9921   0.9912   0.9896   0.9991
## Neg Pred Value         0.9995   0.9989   0.9975   0.9984   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1927   0.1723   0.1624   0.1830
## Detection Prevalence   0.2846   0.1942   0.1738   0.1641   0.1832
## Balanced Accuracy      0.9990   0.9969   0.9932   0.9948   0.9976
```

As expected, the random forest model is much more accurate than the decision tree model. The accuracy is 0.995.

With the random forest model, the expected out-of-sample error is 0.005, or 1 - the accuracy.

Prediction for test set

Finally, we predict the class of the test set using the random forest model.

```
#for the real test set
predictions_test = predict(modRF, pml_testing, type='class')
predictions_test
```

```
##  1  2  3  4  5  6  7 81  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
```