Random Forests - Analysis

Introduction

Our goal for this phase is to use the reduced variable data set from our exploration phase to create a model predicting human activity, using Random Forests.

To remind ourselves, the variables we will use are:

- tAccMean, tAccSD tJerkMean, tJerkSD
- tGyroMean, tGyroSD tGyroJerkMean, tGyroJerkSD
- fAccMean, fAccSD, fJerkMean, fJerkSD,
- fGyroMean, fGyroSD, fGyroJerkMean, fGyroJerkSD,
- fGyroMeanFreq, fGyroJerkMeanFreq fAccMeanFreq, fJerkMeanFreq
- fAccSkewness, fAccKurtosis, fJerkSkewness, fJerkKurtosis
- fGyroSkewness, fGyroKurtosis fGyroJerkSkewness, fGyroJerkKurtosis
- angleAccGravity, angleJerkGravity angleGyroGravity, angleGyroJerkGravity
- angleXGravity, angleYGravity, angleZGravity
- subject, activity

Of these,

- all except the last two are numeric.
- 'subject' is an integer identifying a person, one of 21 from 1 to 27 with some missing.
- 'activity' is a categorical variable - one of six activities identified earlier -
- 'sitting', 'standing', 'lying', 'walking', 'walking up', 'walking down'.

Why do we use Random Forests? We are using Random Forests [4] in our model due to the relatively high accuracy of this method and the complexity of our data.

These are two major reasons to bring out the heavy artillery of Random Forests, especially when we have too many attrubutes even in a simplified set of attributes.

Methods

Expository Segue on Experiment design

Typically in analysing such data sets we are creating a model that uses the data we are given. How do we know the model will work for other data? The real answer is "We don't". And there's no way we can be sure that we can create a model that will work for new data.

But what we can do is reduce the chances that we are creating an "overfitted" model. That is a technical term for a model that works wonderfully on the given data (fitted to it) and fails on the next data set that comes along. There's a way to design our modeling experiment so we avoid that trap. Here's how.

We take the data set and we keep some of the given data aside and we don't use it for modeling at all. This "held out" set is called the test set.

Then we take our remaining data and we further divide it so that we have a larger set called the training set and a smaller set we call the validation set. Then we create our model using the training set and look at how well it performs on the validation set (i.e. not counting the "held out" data). We are allowed to tweak our modeling as much as we want using the training and validations sets but we are not allowed to look at the held out, test set until we are ready to declare we are done modeling. Then we apply the model to our held out test data – when that test data also shows an acceptable error rate we have a good model.

However if we get a bad error rate from the test data we have a problem. We cannot keep tweaking the model to get a better test result because then we are simply overfitting again. So what do we do? We are allowed to mix up all the data, hold out a new test set which has to be different at least in part from the old one, and then we repeat the exercise. In some cases when we are given a data set by a third party we are not shown the held out set, and we have to submit our model without testing agains the held out set.

The third party then applies our model to the held out test set and we don't get to mix it all up. We only get one shot. We're going to do that here and see how well we do.

Our experiment design

We hold out the last 4 subjects in the data as a test set and the rest are used for our modeling. Why do we do this? The data set, if we look at the supporting docs, suggests that we use the last 4 as test subjects. So in our first pass at this we might as well just follow the instructions. All rows relating to these 4 will be held out and not used during modeling at all.

Of the 17 remaining subjects we use the first 12 subjects as the training set and remaining 5 as the validation set. Why this proportion? Typically 30% of the the training data is used as validation set and 70% used for actual training. The validation set is used as our "internal" test set, not used in modeling and held out for each validation step. The difference between the actual test set and the validation set is that we are allowed to keep tuning our model as long as we keep mixing up the data after each attempt and re-extraction of a validation set.

There is also a methodology that takes this step even further and does n-fold validation. The training set is divided into n (usually 10) equal parts and then each part is used as a validation set while the rest used for training, with n such modeling exercises being conducted. Then some averaging is done to create the best model.

We do not do n-fold validation here.

We divided our data based on the 'subject' variable as we included 'subject' in our model and want to keep all test data separate. What does this mean? The test data should actually be data about which we have no information at all - i.e. it needs to be independent of the training data. So suppose we did not separate out the data on the 4 test individuals but we just decided that we would mix up all the rows and take say 20% as test data, chosen randomly.

Note that we have some 7,000 plus rows so we have a few hundred rows on each individual. So if we mixed it all up and chose randomly, then we would most probably get data from all 21 individuals in our test set. And all 21 in our training set. The test set would not be independent of the training set as both would have somewhat similar mixtures of data. Thus the held out set would not really provide a useful reality check - we have statistically similar info in our training set already i.e. the test set has leaked into the training set.

This would be similar to the situation where we had a homework exercise which was later solved in class the next day. Then we received a quiz question set which had questions very similar to the homework with just some numbers changed. It would not really test our understanding of the subject matter, only our ability to understand the homework (= overfitting).

So when we keep aside our test set separated by all rows for certain individuals we know that the training set has no leaked information about these individuals. It is important to be very diligent about the test data, in this fashion, so that we can have some confidence that our model is not overfitting our sample data.

Results

Training

We now run our RandomForest modeling software on our training set, described earlier, and derive a model along with some parameters describing how good our model is.

```
#Training dataset
samtrain = read.csv('./datasets/samsung/samtrain.csv')
#Test datasets
samval = read.csv('./datasets/samsung/samval.csv')
samtest = read.csv('./datasets/samsung/samtest.csv')
```

```
#In R it would have been a "factor" type and R would have used that for classification.
# We map activity to an integer according to
# laying = 1, sitting = 2, standing = 3, walk = 4, walkup = 5, walkdown = 6

samtrain$activity = factor(samtrain$activity,
                           levels = c('laying', 'sitting', 'standing', 'walk', 'walkup', 'walkdown'),
                           labels = c(1,2,3,4,5,6))
samval$activity = factor(samval$activity,
                           levels = c('laying', 'sitting', 'standing', 'walk', 'walkup', 'walkdown'),
                           labels = c(1,2,3,4,5,6))
samtest$activity = factor(samtest$activity,
                           levels = c('laying', 'sitting', 'standing', 'walk', 'walkup', 'walkdown'),
                           labels = c(1,2,3,4,5,6))
```

```
# We use the R 'randomForest' package
#install.packages("randomForest")
library(randomForest)
```

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

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

```
set.seed(1234)
# Create a Random Forest model with default parameters
model1 <- randomForest(activity ~ .-subject, data = samtrain,
                       importance = TRUE, ntree=500,
                       oob.prox = TRUE)
```

```
# use the average error rate (OOB score) which is an estimate of accuracy of our model
1-mean(model1$err.rate)
```

```
## [1] 0.9826481
```

```
### TRY THIS
# use "feature importance" scores to see what the top 10 important features are

#??
```

```
model1
```

```
##
## Call:
##  randomForest(formula = activity ~ . - subject, data = samtrain,      importance = TRUE, ntree = 500
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 1.37%
## Confusion matrix:
##     1   2   3   4   5   6 class.error
## 1 221   0   0   0   0   0 0.000000000
## 2   0 194   4   0   0   0 0.020202020
## 3   0   6 221   0   0   0 0.026431718
## 4   0   0   0 264   1   1 0.007518797
## 5   0   0   0   0 210   0 0.000000000
## 6   0   0   0   6   0 187 0.031088083
```

```
#This is the extractor function for variable importance measures as produced by randomForest.
#type = either 1 or 2, specifying the type of importance measure (1=mean decrease in accuracy, 2=mean d
importance(model1, type = 1)
```

```
##                       MeanDecreaseAccuracy
## X                                30.147943
## tAccMean                         12.186363
## tAccStd                          20.678961
## tJerkMean                        13.033841
## tJerkSD                          13.951413
## tGyroMean                         8.168829
## tGyroSD                           6.172762
## tGyroJerkMean                    10.951783
## tGyroJerkMagSD                   10.919344
## fAccMean                         16.144948
## fAccSD                           24.114091
## fAccMeanFreq                     16.326393
## fJerkMean                        12.798678
## fJerkSD                          12.112688
## fJerkMeanFreq                    13.575680
## fGyroMean                        13.483579
## fGyroSD                           9.160397
## fGyroMeanFreq                     6.544682
## fGyroJerkMean                     6.738659
## fGyroJerkSD                       5.641690
## fGyroJerkMeanFreq                 7.637813
## fAccSkewness                     22.870410
## fJerkSkewness                     7.161170
## fGyroSkewness                     5.479448
## fGyroJerkSkewness                 7.618883
## fAccKurtosis                     13.667608
## fJerkKurtosis                    13.142118
## fGyroKurtosis                    12.714085
## fGyroJerkKurtosis                12.914349
## angleAccGravity                   4.680408
## angleJerkGravity                  5.905401
```

```
## angleGyroGravity                12.555914
## angleGyroJerkGravity             5.535531
## angleXGravity                   37.391525
## angleYGravity                   45.734378
## angleZGravity                   24.165311
```

We use the predict() function using our model on our validation set and our test set and get the following results from our analysis of errors in the predictions.

```
val_pred = predict(model1, newdata=samval[,-c(38)], type = 'class')
#table(observed = samval[,38], predicted = val_pred)

test_pred = predict(model1, newdata=samtest[,-c(38)], type = 'class')
#table(observed = samtest[,38], predicted = test_pred)
```

Prediction Errors and Computed Error Measures

```
sprintf("mean accuracy score for validation set: %f", mean(val_pred == samval$activity))
```

```
## [1] "mean accuracy score for validation set: 0.834385"
```

```
sprintf("mean accuracy score for validation set: %f", mean(test_pred == samtest$activity))
```

```
## [1] "mean accuracy score for validation set: 0.874074"
```

We now compute some commonly used measures of prediction "goodness".

```
test_cm_details$byClass
```

```
##          Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
## Class: 1   1.0000000   1.0000000      1.0000000      1.0000000 1.0000000
## Class: 2   0.8560606   0.9410319      0.7583893      0.9679865 0.7583893
## Class: 3   0.7420495   0.9575707      0.8045977      0.9403595 0.8045977
## Class: 4   0.9170306   0.9960191      0.9767442      0.9850394 0.9767442
## Class: 5   0.8518519   0.9700552      0.8288288      0.9746635 0.8288288
## Class: 6   0.8750000   0.9836576      0.8928571      0.9806051 0.8928571
##             Recall        F1 Prevalence Detection Rate
## Class: 1 1.0000000 1.0000000  0.1973064      0.1973064
## Class: 2 0.8560606 0.8042705  0.1777778      0.1521886
## Class: 3 0.7420495 0.7720588  0.1905724      0.1414141
## Class: 4 0.9170306 0.9459459  0.1542088      0.1414141
## Class: 5 0.8518519 0.8401826  0.1454545      0.1239057
## Class: 6 0.8750000 0.8838384  0.1346801      0.1178451
##          Detection Prevalence Balanced Accuracy
## Class: 1            0.1973064         1.0000000
## Class: 2            0.2006734         0.8985463
## Class: 3            0.1757576         0.8498101
## Class: 4            0.1447811         0.9565248
## Class: 5            0.1494949         0.9109535
## Class: 6            0.1319865         0.9293288
```

```
#Accuracy for each class / levels
test_cm_details$byClass[,11]
```

```
##  Class: 1  Class: 2  Class: 3  Class: 4  Class: 5  Class: 6
## 1.0000000 0.8985463 0.8498101 0.9565248 0.9109535 0.9293288
```

```r
mean(test_cm_details$byClass[,11])
```

```
## [1] 0.9241939
```

```r
#Precision for each class / levels
test_cm_details$byClass[,5]
```

```
##  Class: 1  Class: 2  Class: 3  Class: 4  Class: 5  Class: 6
## 1.0000000 0.7583893 0.8045977 0.9767442 0.8288288 0.8928571
```

```r
mean(test_cm_details$byClass[,5])
```

```
## [1] 0.8769029
```

```r
#Recall for each class / levels
test_cm_details$byClass[,6]
```

```
##  Class: 1  Class: 2  Class: 3  Class: 4  Class: 5  Class: 6
## 1.0000000 0.8560606 0.7420495 0.9170306 0.8518519 0.8750000
```

```r
mean(test_cm_details$byClass[,6])
```

```
## [1] 0.8736654
```

```r
#F1 Score for each class / levels
test_cm_details$byClass[,7]
```

```
##  Class: 1  Class: 2  Class: 3  Class: 4  Class: 5  Class: 6
## 1.0000000 0.8042705 0.7720588 0.9459459 0.8401826 0.8838384
```

```r
mean(test_cm_details$byClass[,7])
```

```
## [1] 0.8743827
```

Conclusions

We can make the following concrete conclusions looking at the above results.

Random Forests give us satisfactory error rates and predictive power in this scenario.

Using domain knowledge it is possible to get suprisingly high values of predictive measures, and low error rates on validation and test sets.

This is supported by the results, i.e. ~90% on predictive measures, OOB error estimates ~2%.

We only did this once and did not go back and forth tweaking the models. Note that we stuck to the rules here and did not see the test set until we were done modeling.

Focusing on magnitude and angle information for acceleration and jerk in the time and frequency domains gives us a model with surprising predictive power. It's possible that a brute force model will give better predictive power but it would simply show us how to blindly apply software. If for some reason the model misbehaved or failed, we would not have any insight at all as to why. Instead we used domain knowledge to focus on insight and in the process created a model with interpretive value.

Model performance on the test set is better than on the validation set as seen in the two "Total" rows above and each individual activity.

Let's see how we might be able to improve the model in future. It's always good to note the possible ways in which our model(s) might be deficient or incomplete or unfinished so we don't get overconfident about our models and overpromise what they can do.

Critique

- Our model eliminated a number of Magnitude related attributes such as -mad, -max, -min also a number of Gyro related variables during the variable selection process using domain knowledge. These may be important but this was not tested. We may want to look at that the next time we do this.
- Variable importance should be investigated in detail - i.e. we really ought to look at how we can use the smaller number of attributes identified as important, to create the model and see what the difference is. Computationally this would be more efficient. We could even use simpler methods like Logistic Regression to do the classification from that point on, using only the reduced set of variables.