Myers-Briggs Type Prediction with Trees

Intro

The Myers-Briggs Type Indicator is a metric aimed at quantifying a person's personality. The result of the Myers-Briggs test gives the person being tested a four letter result. The first letter measuring a person's extrovertedness, the second measuring a person's perception, third measuring method of information processing, and lastly measuring the implementation of information. The Myers-Briggs testing is regarded as "acceptable" in terms of actual scientific validity as a method of measuring personality, however it still a widely used metric in the corpoate world of America. The ability to classify people into like groups based on their personality can be very valuable especially in the human resources setting of an organization. The task of grouping employee's and applicant's can be useful for many reasons; such as grouping candidates for promotions, determining employee's that need to be phased out, figuring ideal placements, or a vast number of analyses. Our objective is to create a model that can quantify a person's Myers-Briggs type based on their written text.

The data

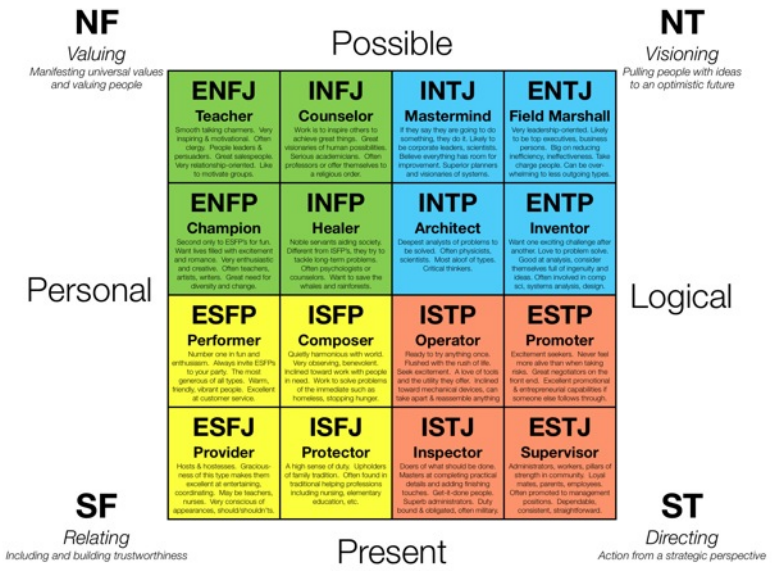
The data in use is from Kaggle.com. This data contains the 50 most recent posts from 8600 users of the PersonalityCafe forum, all of varying personality types. Within the raw data file, each user and their post's are contained in one row, with the posts separated by a triple pipe character ("|||"). After processing that data with python, the resulting data set contains one post per line, with the related Myers-Briggs type and an arbitrary id for that user. This new data set has 422840 observations. The second data set in use contains each unique Myers-Briggs type and an associated x-value and y-value. These two values are what we will use to plot types on a Myers-Briggs matrix. That data set looks like this:

```
##    MBTI xMBTI yMBTI
## 1  ISTJ     3     1
## 2  INTJ     3     4
## 3  INTP     3     3
## 4  ISTP     3     2
## 5  INFJ     2     4
## 6  ISFP     2     2
## 7  ISFJ     2     1
## 8  INFP     2     3
## 9  ESFJ     1     1
## 10 ESFP     1     2
## 11 ENFP     1     3
## 12 ENFJ     1     4
## 13 ENTP     4     3
## 14 ESTP     4     2
## 15 ESTJ     4     1
## 16 ENTJ     4     4
```

For example: a particular user might have been predicted as a ISTJ 45 times and then an INTJ 5 times. This would make their average x-values and average y-values 3 and 1.3, respectively. This would fall into the ISTJ quandrant of the chart, which is the correct prediction for this example user.

Myers-Briggs Matrix

The Myers-Briggs matrix is essentially a scatter plot with values being mapped to the x and y-axes. The higher the x-value the more logical with the opposite being more personal. The higher the y-value the more possible



with the opposite being present minded.

Let's add our main libraries and read & clean the data sets as necessary. Also factor the Myers-Briggs type variable labeled 0 through 15.

```
library(tidyverse)
library(caret)

dataset_original <- read.csv('data/mbti_kaggle.csv', stringsAsFactors = FALSE, fileEncoding="latin1", qu
mbti_types <- read.csv('data/mbti_types.csv')

# Filter out URLs where string contains 'http://'
dataset_original <- filter(dataset_original, !grepl('http://|https://', post))

# Factor MBTI Type
dataset_original$type <- factor(dataset_original$type, labels = c(0:15),
                                levels = mbti_types$MBTI)
```

Get a sample of the data; 15 user's posts from each Myers-Briggs type

```
set.seed(100)
indexes <- dataset_original %>%
  group_by(type) %>%
  sample_n(15)
indexes <- indexes$idx
dataset_original <- dataset_original %>%
  filter(idx %in% indexes)
rm(indexes)
```

Load the data into a corpus and then a DocumentTermMatrix ("DTM"). The DTM will contain a large number of rows of frequently used English words. Then each observation will contain 0s and 1s if the line of text does not contain or does contain that word, respectively. For example, if one of a user's post contained "Hello world", then the DTM would have a 1 in the hello column, the world column and 0s everywhere else. Additionally, the final column of the DTM contains the Myers-Briggs type.

```
library(tm)
library(SnowballC)
corpus = VCorpus(VectorSource(dataset_original$post))
```

```
corpus = tm_map(corpus, content_transformer(tolower))
corpus = tm_map(corpus, removeNumbers)
corpus = tm_map(corpus, removePunctuation)
corpus = tm_map(corpus, removeWords, stopwords())
corpus = tm_map(corpus, stemDocument)
corpus = tm_map(corpus, stripWhitespace)

dtm = DocumentTermMatrix(corpus)
dtm = removeSparseTerms(dtm, 0.999)
dataset = as.data.frame(as.matrix(dtm))
dataset$type = dataset_original$type
```

Split the data into a train and test set with a split ratio of 80%

```
library(caTools)
set.seed(123)
split = sample.split(dataset$type, SplitRatio = 0.8)
train = subset(dataset, split == TRUE)
test = subset(dataset, split == FALSE)
```

Decision tree

Fitting a decision tree model. Predicting using the test set. Let's see the confusion matrix and the overall results of the model.

```
library(rpart)
dtree <- rpart(type ~ .,
               data = train,
               method = "class")

dtree.pred <- predict(dtree, newdata = test, type = "class")
```

```
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
##         0    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         1    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         2    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         3  135 139 137 147 132 127 135 144 100 107 141 143 137 123 129 122
##         4    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         5    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         6    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         7    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         8    0   1   2   0   0   0   0   0  24   1   1   1   1   3   3   2
##         9    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         10   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         11   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         14   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##         15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.08001872     0.01226864     0.06886328     0.09234134     0.06878802
## AccuracyPValue  McnemarPValue
##     0.02418883            NaN
```

We have extremely poor accuracy at 8%, with most predictions being done on class 3 of the Myers-Briggs type. Let's move to a different model to see how much we can bring up the accuracy.

Bagged tree

```
library(ipred)
library(Metrics)
btree <- bagging(type ~ .,
                 data = train,
                 coob = TRUE)

btree.pred <- predict(btree, newdata = test, type = "class")
```

```
##            Reference
## Prediction   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
##          0  19   1   1   1   2   0   5   0   1   1   2   1   2   1   1   0
##          1  23  35  34  25  27  32  27  18  22  15  18  21  23  23  15  13
##          2   1   1   8   5   2   0   3   0   1   3   3   4   4   2   3   0
##          3  81  98  85 101  78  76  65 107  54  73  96  90  78  69  87  82
##          4   3   0   2   3  12   1   1   0   1   1   3   3   3   4   1   0
##          5   0   0   0   1   1   9   0   0   2   1   1   0   0   2   2   2
##          6   2   0   1   2   1   2  22   1   4   3   2   2   2   1   0   2
##          7   2   3   1   1   0   0   2   8   3   2   2   2   0   1   2   3
##          8   0   2   2   0   0   1   0   0  24   0   2   0   1   2   2   2
##          9   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0
##          10  1   0   0   3   1   3   2   2   4   4   6   0   2   1   0   5
##          11  1   0   1   1   1   0   1   1   3   0   3  12   2   0   2   0
##          12  1   0   1   1   0   1   3   2   4   2   0   3  13   1   1   3
##          13  0   0   1   1   0   2   3   2   1   1   0   1   7  17   5   1
##          14  0   0   0   1   2   0   1   2   0   1   1   3   0   2   9   3
##          15  1   0   2   0   5   0   0   1   0   1   3   2   1   0   2   8
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   1.417876e-01   8.020208e-02   1.272627e-01   1.573023e-01   6.878802e-02
## AccuracyPValue  McnemarPValue
##   3.094795e-32            NaN
```

Our accuracy is slightly better here at 14%.

Gradient boosting

```
library(xgboost)
set.seed(100)
xgboost <- xgboost(data = as.matrix(select(train, -type)),
                   label = as.numeric(train$type)-1,
                   nrounds = 10,
                   objective = "multi:softmax",
                   num_class = 16)
```

```
## [1]  train-merror:0.798900
## [2]  train-merror:0.773046
## [3]  train-merror:0.755615
## [4]  train-merror:0.743683
## [5]  train-merror:0.733856
## [6]  train-merror:0.720870
```

```
## [7]  train-merror:0.715372
## [8]  train-merror:0.709991
## [9]  train-merror:0.700749
## [10] train-merror:0.692560
```

```
xgb.pred <- predict(xgboost, newdata = as.matrix(select(test, -type)))
```

```
##           Reference
## Prediction  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##         0  31 10 11  6  7  7 11  5  3  5  9  4  2  4  6  8
##         1   9 13  5  6  7  6  4  5  6  2  2  6 10  4  6  5
##         2   8  8 25  9  9  6  8  6  4  8  9  8  7  7  6  5
##         3  31 30 30 57 19 30 20 28 22 34 28 21 32 29 36 31
##         4   5  6  6  8 22  3  5  8 14  4  9 14  7 12  8  6
##         5   3  6  5  3  5 18  3  4  4  4  6  6  5  6  7  4
##         6   7  4  9  4  9  6 32  8 13  9 12  7  6  6  1  5
##         7   5 11  1  6  9  3  8 24  4  4  8 15  6  7  5  8
##         8   2  4  5  4  4  9  7 12 25  0  4  4  5  6  7  5
##         9   6  2  2  3  7  4  2  2  3 10  4  4  4  2  2  3
##         10  8  8 13  7 16 12 15 16 13  9 25 16 10  5  4  6
##         11  4  6  5 11  5  5  6  4  5  3  8 20  5  3 10  3
##         12  7 13  4  7  2  3  6  6  5  6  5  6 21  7  6 12
##         13  1  5  6  6  1  5  2  5  1  1  2  3 13 19  6  5
##         14  3  4  5  2  4  3  2  5  1  4  2  6  1  5 16  4
##         15  5 10  7  8  6  7  4  6  1  5  9  4  4  4  6 14
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   1.740758e-01   1.174504e-01   1.582210e-01   1.908298e-01   6.878802e-02
## AccuracyPValue  McnemarPValue
##   3.271811e-60   4.049799e-25
```

This gradient boosting model with 10 rounds yields us only a 17% accuracy rate.

Random forest

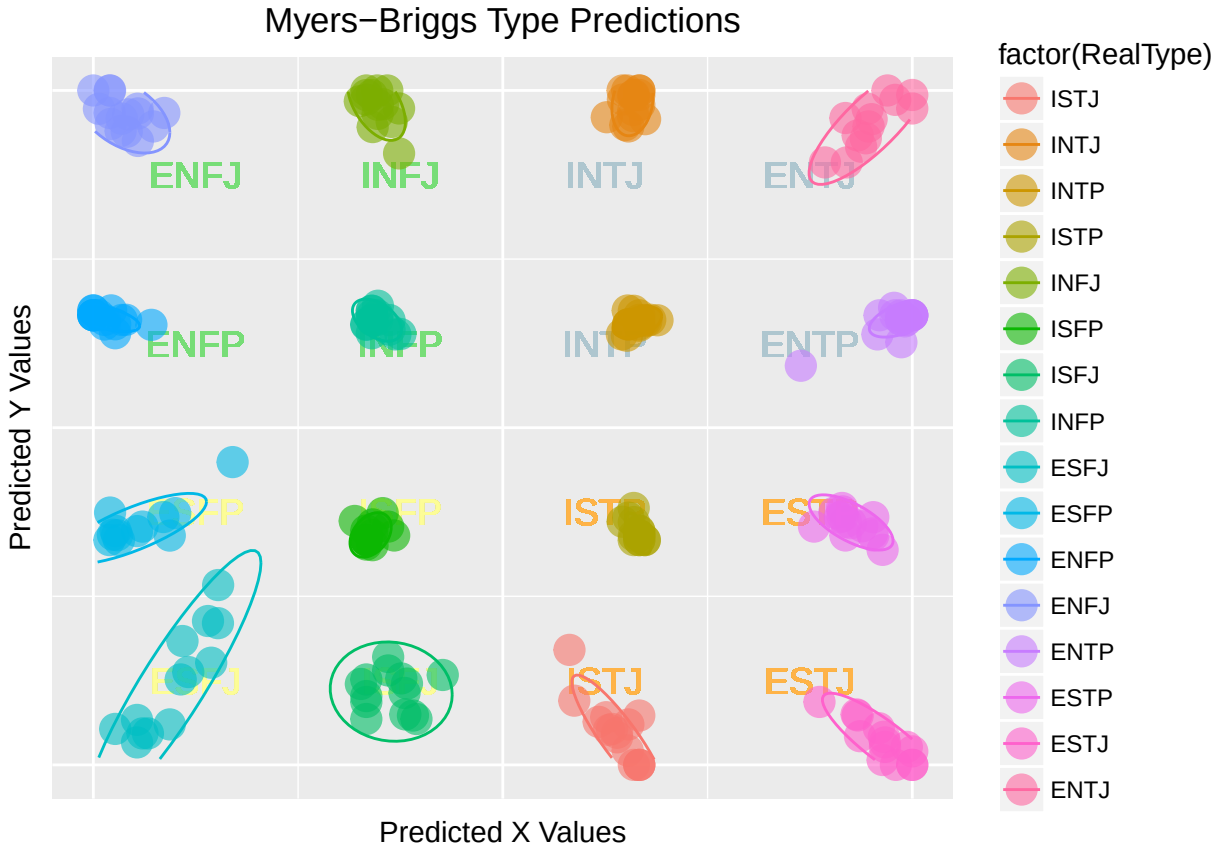
```
library(randomForest)
set.seed(100)
rforest <- randomForest(x = train,
                        y = train$type,
                        ntree = 10)

rforest.pred <- predict(rforest, newdata = test, type = "class")
```

```
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
##         0   96   7   3   0   4   1  11   9  10   2   3   2   0   4   0   3
##         1    3 116   1   3   3   3   4   4   4   4   1   4   3   2   4   3
##         2    1   2 116   4   4   2   9   0   6   4   2   4   6   3   2   0
##         3    1   2   2 129   1   2   2   0   3   1   1   1   2   1   3   3
##         4    4   0   4   1 103   2   7   0   9   1   1   9   4   3   3   4
##         5    0   2   1   1   1 106   1   0   3   1   1   0   0   4   4   2
##         6    9   1   2   1   3   2  54   1  13   5   0   4   6   6   1   8
##         7    4   2   0   1   1   1   7 116   4   4   2   0   0   0   6   2
##         8    3   1   3   0   2   1   6   0  38   2   1   2   0   4   6   2
##         9    1   1   0   0   0   0   0   0   2  74   0   1   2   3   0   1
##         10   4   1   1   0   3   0   2   3   4   0 121   5   0   3   0   4
```

```
##          11   4   0   1   2   1   2   8   4   9   0   6 103   1  10   2   2
##          12   2   1   2   1   0   1   5   1   5   4   1   1 104   2   5   3
##          13   2   2   3   2   2   1   8   2   4   2   0   4   3  72   4   3
##          14   0   0   0   2   1   1   3   3   1   1   0   1   7   4  89   4
##          15   1   2   0   0   3   2   8   1   9   3   2   3   0   5   3  80
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.70987365     0.69029101     0.69011993     0.72905016     0.06878802
## AccuracyPValue  McnemarPValue
##     0.00000000            NaN
```

At last we have accuracy that is significantly higher than those from our previous three models. At roughly 71% accuracy let's take a look at how the actual predictions fit into the Myers-Briggs matrix.



This chart is generated by averaging all of the predicted x and y values of the Myers-Briggs type. This chart demostrates to us that, despite only achieving a 71% accuracy with the random forest model, we have a model that predicts the correct Myers-Briggs type very accurately. We have confidence ellipses surrounding the predicted points. These confidence ellipses represent, in few words, that 95% of ellipses calculated on different samples of this population would have the same mean as these ellipses. For future use of this model, we can be confident that the means will often fall into the proper quadrant. Additionally, on future samples of this population or on completely different data sets, a point falling slightly outside a particular quadrant is not a cause for concern. This is due to the fact that many Myers-Briggs types are close in terms of actual personality traits so a person that may be an ESFJ could be confused as a ESFP, ISFP, ISFJ, or even ENFJ.

Conclusion

We have constructed a nice model to classify a person into a Myers-Briggs grouping based on their written text. There is definitely room for improvement, however we have a 71% accuracy rate on individual MBTI

predictions; and even better results once the predictons are averaged as seen when we plot the average values on the matrix.