Myers Briggs personality predicting

Matthew Rogers *
School of Computing
Clemson University
mwr2@clemson.edu

Yue Zhang
School of Computing
Clemson University
yzhng@clemson.edu

Abstract

The Myers Briggs personality assessment puts a personality type to an individual. There are four components to an individual's personality. For each component, an individual takes on one out of two attributes. Since each of the four components has two attributes, then there are sixteen different personality types that an individual can be labeled as. For the first component there is introversion vs. extroversion, for the second there is intuition vs. sensing, for the third thinking vs feeling, and for the last component there is judgment vs perception. The objective of this project is to investigate the effectiveness of the perceptron to label each component of an individual's personality type.

Due to the nature of the Myers-Briggs type , we can break down the classification task with 16 classes in to 4 binary classification tasks. This is because a MBTI type is composed of 4 binary classes, where each binary class represents a dimension of personality of the MBTI personality model as theorized by the inventors. Therefore, instead of training a multi-class classifier, we instead train 4 different binary classifiers, such that each specializes in one of the dimensions of personality.

Once these new datasets were produced, each dataset (each one was a csv file) was transformed into a table in MATLAB. The rows were sorted on the attribute column so that one type would appear first before the other one. For example, if the dataset with the first attribute was being used, then each individual labeled as an introvert would appear first and then the individuals labeled as extroverts would appear second. Then, a certain number of rows for each type were taken and were combined to form a training set. The other rows remaining were combined to form the test set. After each set was randomly permutated and standardized (the zscore function was used), training set was put through a linear perceptron algorithm to find the weights. Using these weights, the test set was put through another algorithm to test the accuracy of this perceptron.

1 Introduction

2 Background

3 Methods

4 Based on letters

As mentioned in the proposal, for each row (anonymous individual) there was a personality type made up of four attributes and a certain number of columns with each column containing a whole or

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

Preprint. Under review.

part of a twitter post. The first objective was to combine all these columns into one so that there was just one column containing all the posts for each individual. The second objective was to split up the personality type into four separate columns where each column represented an attribute. For example, if an individual was labeled as IFSP, then this column would be split into four attributes where the first attribute would be labeled as I, the second attribute labeled as F, etc. These transformations were done in Excel.

For the first part of our research project, it wanted to be determined if there was an association between the average of different letters and the individual's specific attribute type. To see if there was an association, the frequency of each letter was found for each row (individual). Since, each individual varied on the number of letters used for all of their posts, the average was then calculated for each letter frequency. For example, if an individual used a total of 20000 characters and used the letter 'a' in 1000 instances, then there would be a score of .05 (instances/ total # of characters). After, the average was calculated for each letter for each individual, then this new dataset was split into four datasets where each dataset concentrated on a specific attribute (I vs E, F vs T, etc.). This part of the project was done in Python.

After using this process for each dataset and varying the number of entries in the training set and varying the number of iterations the perceptron took to find the weights, it was observed the accuracy varied significantly. Accuracy scores approximately ranged from 35% to 70%. It was questioned whether perhaps the specific sample used could explain these observations. In other words, maybe choosing an appropriate training set could make the perceptron more accurate by being more representative of the rest of the dataset.

A clustering approach in Python was used to investigate this question. Specifically, Kmeans was used from the Sci-kit Learn package. The intent of the overall process about to be outlined was to cluster the whole population (all the entries in a particular dataset) into groups based on a mean value. Then, a representative would be chosen from each group to be used for the training set. Perhaps, this representative training sample could lead to a higher accuracy score using the linear perceptron.

The specific process started with using just one dataset with the second attribute (F vs T). The columns that were not going to be used in the clustering algorithm were dropped. The data frame was then grouped by the binary classification (1 or -1) and then was split into two separate data frames based on this classification.

Kmeans was then used for each dataset with a certain number of means (but the same) to cluster both data frames. After adding a cluster group column in each data frame, each row (in each data frame) was labeled with its corresponding group number. Each data frame, was then sorted in ascending order based on its cluster group number. A representative was then chosen from each group from each data frame. This representative was chosen by being first the first entry for each group. All the representatives were then put into a training set and the rest of the entries were put in a test set. Each set was then randomly permutated and the cluster group column was dropped. The training and test set were then converted to an array so that it was an acceptable form to be used by the linear perceptron model from the Sci-kit Learn package. The training set was used to train this model and then the test set was used to measure its accuracy.

4.1 letters result

The accuracy was measured using a different number of training 'representatives'. This number was only changed a few times (so there were only a few accuracy scores) but the accuracy did not get above 55%. This process was rudimentary since it did not find the 'best' representative of the group. In other words, it did not find the group member with the lowest average mean. At this time, it is also uncertain how 70% was even achieved in the initial perceptron process (MATLAB). These questions still need to be investigated. However, it was concluded that looking at frequency averages of letters did not provide a significant way of labeling each attribute. Therefore, the next portion of the project was dedicated to investigating the words used instead of the frequency averages of the letters.

5 Based on words

5.1 Preprocessing

When we examined other studies of MBTI using machine learning, we were surprised to find that researchers rarely made a point of cleaning their data set to accord with the actual proportions of MBTI types in the general population (e.g. ISTJ = 0.11, ISFJ = 0.09, etc.)[3]. Since our raw data set

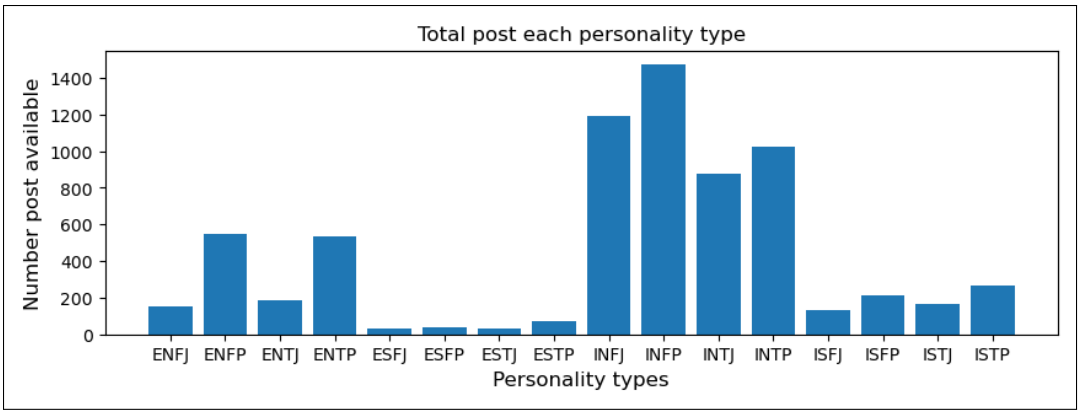

Figure 1: Distribution of MBTI types in the data set.

is severely disproportional (see fig. 1) compared to the roughly uniform distribution for the general population, it was clear to us that some cleaning of the proportional representation of each MBTI type would be necessary. Therefore, we artificially made our test set reflect the proportions found for each type in the general population, so as to prevent any misinterpretation of results due to skewed representation of classes in the test set.

5.2 Meaningless words removal

As the dataset is sourced from an online forum where communication is solely through written text, certain modifications were necessary. Notably, instances of data points containing hyperlinks were eliminated, aiming to foster the model's generalization to the English language. Additionally, to enhance the meaningfulness of each word in the dataset, so-called "stop words," such as common fillers like "a," "the," "or," etc., were excluded using the NLTK library in Python. Lastly, considering the dataset's origin from a website dedicated to explicit discussions about personality models, particularly MBTI, references to specific types (e.g., 'INTJ,' 'INFP,' etc.) were expunged. This measure was taken to prevent the model from potentially "cheating" by learning to recognize MBTI mentions explicitly by name.

5.3 Transfer to root words

To transfer to root words on the text, we employed the nltk.stem.WordNetLemmatizer. This process involved transforming inflected forms of the same root word into their dictionary form; for instance, "walking," "walked," and "walk" were all lemmatized to "walk." This approach enables us to capitalize on the shared meaning inherent in various inflected forms of a given word.

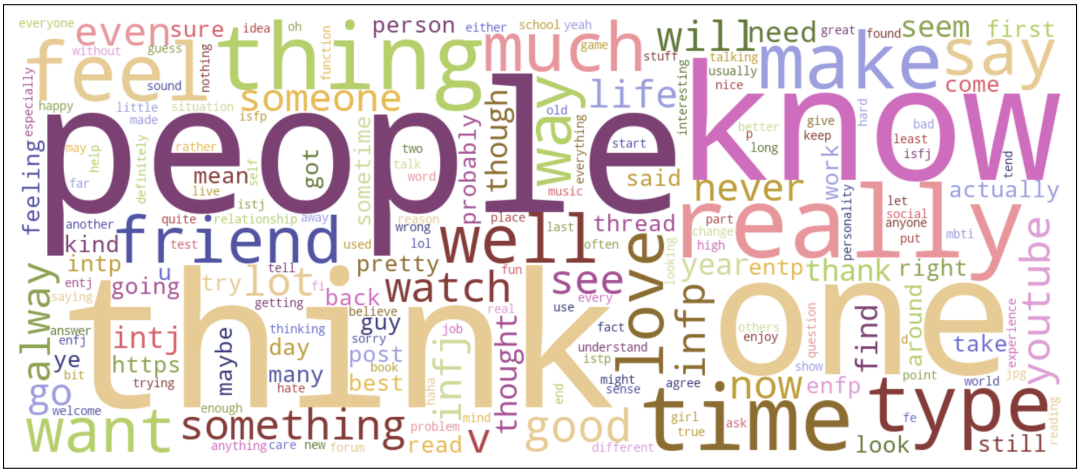

Figure 2: Before Preprocessing.

Figure 3: After Preprocessing.