Models with MBTI

Arnav Joshi, Kevin Andrews

1 Abstract

The aim of this project was to utilize and compare different models to perform natural language processing on input text data and classify it into one of the 16 personality types under the MBTI personality classification model.

2 Introduction

The goal when coming up with the concept for this project was to attempt to classify on a relatively subjective set of multi class labels. The hope was to come up with a classifier that could be utilized for recreational purposes such as entertainment, but took in text samples so that we could design a nice input system to be able to show the user the outcome of their text prediction. We wanted to do this by trying out different models we learned in class.

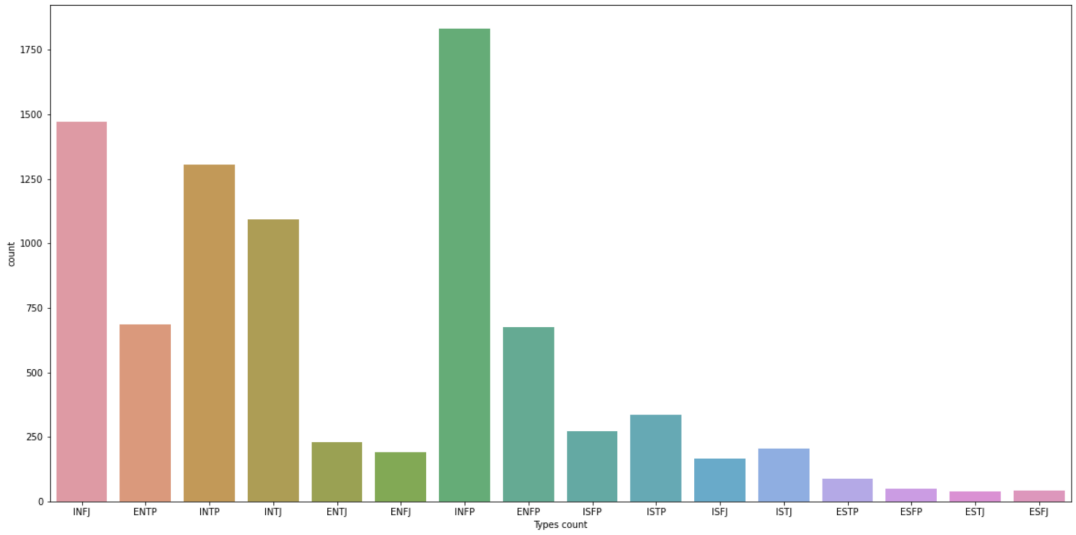
There are 16 MBTI personalities, each of them being a combination of 4 categories: Introverted vs. Extroverted, Intuitive vs. Sensor, Feeler vs. Thinker, and Judger vs. Perceiver. For a long time, the internet has used these as personality classifiers for different forms of entertainment. A classic example of this was that many internet quizzes would match your personality to a character from a work of fiction using these personality types as a basis. More recently, filters on TikTok attempt to guess personality type based solely on a person's facial features and expressions (likely just random).

At first, we wanted to use an approach like LDA, but after some testing we decided that this approach seemed to be difficult with the approach that we wished to take. After deliberation, we decided upon trying pLSA to see how well the personalities could be predicted and to also get a mixture of personalities that were represented in the text — instead of receiving one personality type the user could see what percentage of each type they were. Other language processing models we ended up using were an RNN (specifically an LSTM) and BERT. We thought it would be interesting to use the two most popular language processing models for their times and see how they held up to each other.

3 Experimental Setup & Results

After searching for a while, we found a dataset we were happy with on Kaggle that mapped groups of text to a distinct personality type. It was expansive with 8675 points of data, each with multiple text entries separated by 3 pipes ('|||'). The data required some cleaning — we took out the pipes and other special characters (like ':D') and a lot of the text had Youtube links and links to other websites that we wanted to take out. This is something that will be touched more on later in this paper, but one definite problem with this set was that some personalities were more well represented than

others, which is something we believed led to some problems with our models later down the line.



3.1 pLSA

First, we looked at pLSA to see how well it could predict the personalities and give a spread of the different personalities represented within a text. In addition to working with all 16 personality types as topics, we also wanted to try working with each of the 4 categories and seeing it how worked with a binary decision.

Similar to how we discussed in class, we initialized a theta matrix to represent the topic distributions per document, a phi matrix to represent the word probabilities per topic, and a Z matrix to represent the probability of a certain word of document coming from a certain topic. Estimation-Maximization was used to optimize parameters, the E-step being used to update the Z matrix, and the M-step being used to update the theta and phi matrices. Using the CountVectorizer class from Sci-kit learn, we vectorized our text data and ran 15 iterations of E-M. pLSA didn't end up performing nearly as well as we wanted to — here are the results for how it performed:

```
16 PERSONALITIES REPORT
              precision    recall  f1-score   support

           0       0.15      0.06      0.09      1470
           1       0.10      0.09      0.10       685
           2       0.17      0.07      0.10      1304
           3       0.13      0.07      0.09      1091
           4       0.03      0.06      0.04       231
           5       0.03      0.08      0.04       190
           6       0.19      0.06      0.09      1832
           7       0.09      0.07      0.08       675
           8       0.03      0.04      0.03       271
           9       0.02      0.04      0.03       337
          10       0.02      0.06      0.03       166
          11       0.04      0.10      0.06       205
          12       0.01      0.06      0.02        89
          13       0.01      0.08      0.01        48
          14       0.00      0.05      0.01        39
          15       0.00      0.05      0.01        42

    accuracy                           0.07      8675
   macro avg       0.06      0.07      0.05      8675
weighted avg       0.13      0.07      0.08      8675
```

We also output the top 10 words for each personality, and the personality spreads within a text. For example, here is the spread of personalities of our fourth data point:

```
Document type:
INTP
Percentage of Personality mixtures:
INFJ: 4.11524194388917%
ENTP: 0.8490583809101209%
INTP: 8.654900882980073%
INTJ: 16.327029396806722%
ENTJ: 4.914057509245255%
ENFJ: 14.83167097024076%
INFP: 5.954649100858942%
ENFP: 5.257788154377346%
ISFP: 3.493182669252658%
ISTP: 12.777400471488543%
ISFJ: 1.7242716065167534%
ISTJ: 1.9953908313293787%
ESTP: 4.761958950548421%
ESFP: 7.407731268681466%
ESTJ: 6.3906340376215525%
ESFJ: 0.5450338252528384%
```

The actual label for this text is INTP, but as we can see from the output, the predicted personality type is 'INTJ' — 'INTP' seemed to be in the middle of the pack for the personality spread. 3 out of the 4 categories matched — the judger versus perceiver categories are what's different. This

made us rethink how we should measure the accuracy of this. For the model, this was regarded as completely the wrong output when in actuality it was 75% correct. Let's take a look at the spread for judger vs perceiver in the fourth data point:

```
Document type:
P
Percentage of Personality mixtures:
J: 51.53999456890315%
P: 48.46000543109685%
```

Even when presented as a binary option (J vs. P), it was barely classified as a J. In fact when looking at the other 3 categories, the two categories that it was wrongly classified in was the I vs. E category and the F vs. T category (again, E and F barely edged out). This is significant as the second high personality in the spread was the 'ENFJ' type — exactly two letters apart from 'INTJ'.

When looking at the top 20 words for 'INTJ', 'ISTJ', and 'ENFJ', we can see some similarities between them. 'INTJ' shares the words 'like' and 'time' (these two appear lower in the list in 'INTJ') with 'ENFJ'. 'INTJ' shares the words 'think', 'said' and 'like' with 'ISTJ' — again these words appear lower in the list for 'INTJ'. 'ENFJ' and 'ISTJ' share the work 'understand', a word that appears high in the list for both types. Overlaps like these among personality types and category classes in this document and others are likely what causes the discrepancies we see.

3.2 LSTM

The next model we wanted to try out was an RNN, a topic we talked about in class. Before transformers, it was the popular model for language processing and we wanted to put it to the test. After some research, we found a popular type of RNN called LSTM (long short term memory). A traditional RNN struggles with long term information — as it iterates through a given text, important

information that may have occurred earlier in the sentence has less and less weight in future iterations. The LSTM attempts to remedy this by introducing the 'cell state', another pipeline of sorts that travels through iterations — information in the cell state is relatively harder to modify thanks to the use of gates which control how much information from each word to let into the cell state. Because of this, long term information is able to persist through the network, producing better results than a traditional RNN [2].

To accomplish this, we found a fairly popular library called *fastai*, a library built on top of PyTorch — it had some built in methods that allowed us to use and train a neural network and make predictions with it [3]. The specific LSTM used by the library used average stochastic gradient descent (ASTD) for optimization, which " instead of returning the last iterate as the solution, returns an average of the iterates past a certain, tuned, threshold T" [1].

We created and trained the model for 10 total epochs and settled on a learning rate of 1e-2. We ended up with a training loss of 0.89, a validity loss of 1.69, and an accuracy of around 50%. Though this performed significantly better than pLSA, the numbers for our losses seemed to indicate signs of overfitting as the training loss was lower — and decreased at a significantly lower rate — than the validity loss. Upon further investigation it did turn out this was the case — we trained the network (a separate time) with more epochs and training loss ended decreasing even quicker and the validity loss began to stay somewhat stagnant.

After the training was completed, we had little fun with predicting the personality types. We tried using the opening monologue from the *The Batman* to predict the character's MBTI type and

got ‘INFP’ — which isn’t too far off considering the internet’s consensus on his personality type being an ‘INTP’.

We also tried predicting Ferris Bueller’s personality type (internet consensus seems to be either ‘(ESFP’ or ‘ENTP’) by using the opening monologue from his movie as well, and we ended up with ‘ENFP’.

3.3 BERT

The last model that we wished to try was the relatively new model that was presented in class for natural language processing, BERT. BERT stands for Bidirectional Encoder Representations from Transformers and functions precisely as the name suggests: it is pre-trained for deep bidirectional representations by conditioning on left and right context in all layers [4]. BERT is a strong model for our requirements as it is pre-trained and we can simply fine tune it without major architectural changes to fit our requirements [4]. All we must do is provide a final output layer that matches the requirements of our architecture and the model will be fine tuned to our requirements (once trained on our dataset of course). BERT is not pre-trained using left-to-right or right-to-left models, but with two unsupervised tasks instead: a “Masked Language Model” and “Next Sentence Prediction” [4]. In MLM, to train bidirectionally, some percentage of the input tokens are masked at random and then the model attempts to predict the IDs of the masked tokens [4]. The NSP task is based in the idea that the relationship between two sentences should be captured in language modeling and is done by using sentence separator and a sequence separator while pre-training on sentences [4].

For our purposes, we utilized Hugging Face’s transformers libraries, with the most important parts of the library being the “BertTokenizer” and the “BertForSequenceClassification”. This library

seemed to be one of the most advanced BERT libraries and had a good amount of documentation, so we decided to move forward with it. Other libraries lacked a lot of pre-training or had convoluted architecture, so this seemed like the most logical next step.

We then read all of the cleaned text data, split the data into training and testing, and encoded all of the data using the library's BertTokenizer. After encoding, we created the masks and data loaders that we would require for training. We set up the model to train for 10 epochs with learning rate $3 * 10^{-5}$ and a seed value so we could recreate our results. We then created some helper functions and evaluation functions and ran the training loop for 10 epochs.

Upon completion of the training loop, we found that the model has sorely overfit to the training data as the test loss bottomed out around epoch #2 and even the F1 score bottomed out around epoch #9. Nonetheless, we moved forward with these results to see its accuracy in prediction manually. Surprisingly, the model performed decently with manual predictions, usually at least doing alright with introvert versus extrovert predictions (although yes, these are binary predictions at the end of the day). The quote from *The Batman* mentioned earlier receives 'INFJ' as its prediction and the accurate personality type should be 'INTP' so it is not incredibly far off.

Overall, BERT seemed to perform decently although its performance could likely be improved further with more fine tuning (and possibly a higher amount of memory as the GPU we were using seemed to reach its limit occasionally).

4 Conclusion / Future Steps

Ultimately, we ran into a lot of difficulty when faced with the task of natural language processing on a heavily biased dataset in order to create a 16 class classification model. While BERT seemed to classify the best, there still seems to be a lot of work left to fine tune it to be able to accurately pin down everyone's personality type. As an exercise in trying to create an entertaining system to classify personality based on text, the project was a great success as it was definitely incredibly entertaining to see both the models' successes and its unfortunate mishaps (BERT seems to believe parties are exclusively for introverts)! Future steps would involve changing hyperparameters for BERT, modifying the dataset, or downright scraping data from a different source as the dataset seems deeply flawed. While F1 score is important, we may also decrease the number of epochs significantly as test loss is also incredibly important and was heavily sacrificed by training for so many epochs. We may also test different models like BART (but we think we can achieve greater success with BERT with more tinkering and better data).

Furthermore, a major disclaimer that must be observed when going about classifying text inputs into personality types is that in no way should these be used for anything except entertainment purposes. There are major ethical concerns when it comes to possible discrimination with this model as a basis and even if there were not, the model is somewhat inaccurate and clearly biased, which is a problem in and of itself.

This project gave us a greater insight into the wonders and difficulties of natural language processing and allowed us to look into recent research when designing models to take in text and output a personality type.

6 References

1) https://arxiv.org/pdf/1708.02182.pdf?ref=https://githubhelp.com

2) http://colah.github.io/posts/2015-08-Understanding-LSTMs/

3) https://docs.fast.ai/tutorial.text.html

4) https://arxiv.org/pdf/1810.04805.pdf