It's Personal: Using Text to Predict Myers-Briggs Personality Types

Eric Chiu , Hannah Sheetz , Tristrum Tuttle
University of Pennsylvania
chiueric@seas.upenn.edu, hsheetz@seas.upenn.edu, tut@seas.upenn.edu

Abstract

Can a computer model figure out a person's personality just based on the text of their conversations? We want to predict Myers-Briggs personality types purely based on conversational quotes. By building an effective model to categorize conversational quotes as Myers-Briggs type indicators, we will help workers understand themselves and their interactions better, improving productivity and success in the workplace. We will also build a generative model to attempt to create quotes that fall into a specified Myers-Briggs (MB) category.

1 Problem Background

In today's society, more and more companies rely on personality types to help establish a company culture and even streamline recruitment. Companies like Bridgewater and Amazon use MB personality type tests to help them build effective teams and reduce friction among employees. Additionally, knowing ones own personality type is useful for figuring out career plans, building relationships, and achieving more. Because the Meyer's Briggs test is prone to bias as people are likely to answer questions with how they want to be rather than how they actually are, they can often be placed into the wrong personality type. Our project alleviates this problem of self identification by analyzing sentences said in every day social media interaction and placing them into MBTI personality types.

2 Approach

In this project, we attempt to answer two main questions. The first question is, "can we train a model to accurately predict a person's Myers-Briggs type using just blog posts?" The second question is, "can we train a model to reproduce blog posts with a specified Myers-Briggs type?"

2.1 Data Overview

Our dataset was from Kaggle, and contains 8,675 instances of people's text posts and their MBTI type scraped from an online forum called PersonalityCafe. MBTI personality types consist of ISTJ, INTP, ISFJ, INFJ, ISTP, ISFP, INFP, INTJ, ESTP, ESTJ, ESFJ, ENFJ, ESFP, ENTJ, ENTP, and ENFP. In total, there are 16 personality types, and for our purposes, 16 different labels. The text for each instance contains the last 50 things a person has posted separated by three vertical lines.

2.2 Principle Data Analysis

We looked at the distribution of the number of instances for each MBTI personality type. We found that the number of instances for each MBTI personality type vary greatly, from 42 instances to 1832 instances. At first we thought this was due to the actual distribution of personality types in the real world, but after checking some statistics online and finding that ESTJ, ESFJ, ESTP, and ESFP are the most popular MBTI personalities, we concluded that our dataset is not representative of the real world population. Instead, it is representative of the users on the online forum PersonalityCafe. We were also concerned about the unequal number of words per instance for each MBTI personality type. If the number of words for specific personality types were abnormally low, the training dataset for those personality types would be low, thus causing a weaker prediction compared to the other personality types. Luckily, after further analysis, we found that each personality type generally had the same number of words per instance, at an average of 1253 words per instance.

2.3 Data Cleaning

We have cleaned the text data by removing the text post break characters (three vertical lines, or |||). We also removed URLs from the text, because they were highly variable and did not provide useful emotional analysis in IBM Watson. We decided to leave punctuation, capitalization, and emojis in place because they imply certain types of emotion in IBM Watson. After analyzing the data, we realized that many posts referenced MBTI types, so we removed all mentions of MBTI types. After cleaning the data, we used a Python script to add the IBM Watson emotion features to the data set. The emotional features we included were anger, joy, sadness, disgust, and fear. Each emotional feature is represented by a score from 0 to 1, corresponding to how likely the text is to convey that emotion. We decided to create the training and testing dataset by an 80-20 split of the full dataset. We first shuffled the instances around in case they were ordered in some particular way, such as by personality type. This results in 6998 instances for the training dataset, and 1677 instances for the testing dataset. We also double checked if the distribution of

the number of instances for each MBTI personality type was the same among the training and testing dataset.

3 Problem 1: Myers-Briggs Prediction

To answer the question "can we train a model to predict a person's Myers-Briggs type based on their blog posts", we decided to experiment with both sentiment prediction and text prediction models. When using sentiment to predict Myers-Briggs type, we used LogReg, Random Forest, and Support Vector Machine models. When using text based prediction, we used a Naive Bayes model implemented using TF-IDF transformed documents.

3.1 Models

Emotion Prediction Our initial models used just the Emotional Scores to train and classify the Myers-Briggs types. We tweaked the hyper-parameters of the LogReg, SVM, and Random Forest models to find the best performance for each model. The results of these models are shown below, and in the appendix chart.

Naive Bayes: The Naive Bayes model implemented using TF-IDF transformed documents. The model took the MBTI and a long post that a person with that type wrote. The vectorizer had a maximum of 2000 features, used English stop words, and had an n-gram range of $(1, 3)$. Additionally, as the model was refined, more stop words were added to the NLTK stopwords. At the end of training, the top twenty words used in each MBTI type's blog posts. If a word was included in all 16 of the MBTI or in 15 of the 16 MBTI, it was added as a stop word as there was no or negligible effect on what personality types use a word more or less often.

LDA Model: Since emotional scores were not able to provide much insight, and TF-IDF has 2000 features, we decided to try using Latent Dirichlet Allocation (LDA) to generate features that incorporated aspects of the vocabulary used in the text posts without requiring a single feature for each word. To prepare the data, we used the NLTK SnowballStemmer to replace each word in each blog post with its root. We then generated a dictionary using the processed blog text, filtering out lower extremes (words with fewer than 15 appearances) and keeping the size of our dictionary below 100000 entries. We used the LdaMulticore model from the gensim package to generate our LDA categories. We built two distinct datasets incorporating LDA categories. The first dataset added 15 LDA topic features. After training prediction models on this first dataset and getting poor results, we decided to build a second dataset with 50 LDA category features. After building the two datasets, we trained and tuned a Random Forest Model, Logistic Regression Model, and SVM Model using the LDA category features along with the emotion scores from IBM Watson. The top 3 words from each of the 15 LDA categories has been included in the appendix, the 50 LDA categories can be found in our Github.

3.2 Results

Emotion Prediction LogReg model with penalty=L1, tol=e-5, and solver=liblinear had a validation accuracy of 0.229, validation precision of 0.494, validation recall of 0.238, training accuracy of 0.231, training precision of 0.495, and training recall of 0.237. SVM model with kernel=linear had a validation accuracy of 0.218, validation precision of 0.285, validation recall of 0.09, training accuracy of 0.219, training precision of 0.288, and training recall of 0.09. Random Forest model max_depth=20, n_estimators=2000 had a validation accuracy of 0.211, validation precision of 0.507, validation recall of 0.488, training accuracy of 1.00, training precision of 1.00, and training recall of 1.00. Among the three emotion score models, based on our performance results, the LogReg model is the best model.

Naive Bayes: We measured the performance of the Naive Bayes with TF-IDF by training the model on 6,999 MBTI types and their corresponding blog posts. This ensured that there were enough words to create significant features. When the training data was directly used for testing, the model returned a cross validation score of 0.5367. For five fold cross validating using the training data, the model returned cross validation scores ranging from 0.3204 to 0.3457. Finally, when the preseparated testing data was used, the model returned a cross validation score of 0.3554. The performance of all of these models increased when the additional stop words discussed above were added. Additionally, we calculated the top twenty words used by each type. Interesting, regardless of if the type was thinking or feeling, the word "feel" always scored higher than "think" or "know" within these lists. We can see differences associated with the letters of MBTI. Six out of the eight MBTI types that had E, extroverted, as their first letter, also had "friend" or "friends" in their top twenty words compared with only two out of the eight MBTI types that had I, introverted.

LDA Models: A table of both the LDA Model results can be found in the appendix. We evaluated each model using 5 fold cross validation. Each of the 15 category + emotional score Models had below 25% accuracy, even when we increased the number of LDA features to 50. However, the Random Forest Model was able to score statistically better than random chance, reaching a validation accuracy of over 23%. The SVM models did the worst, and had a tendency to predict just one or two categories.

4 Problem 2: Myers-Briggs Text Generation

To answer the question "can we train a model to reproduce blog posts with a specified Myers-Briggs type?", we decided to experiment with different types of Recurrent Neural Nets (RNNs). RNNs are useful tools for text generation because they keep track of state, allowing the net to make predictions using information from previous inputs along with the current input.

4.1 Models

We initially built two different character-level RNNs to generate text posts. Based on our research, we reasoned that a character-level RNN would work the best for our task of text generation because they can keep track of both short term and long term memory, and the amount of training data needed is much smaller due to the number of possible characters (about

100) being much smaller than the number of possible words (about 200,000).

Linear RNN: The first RNN had an input of a one-hot encoded character (0 or 1 for 100 features corresponding to the printable characters in python), along with 128 additional recurrent features initialized to 0. These features are fed into two linear layers simultaneously. The first linear layer has an output size of 100, and feeds into a dropout layer with dropout rate of 1%, then the dropout layer feeds into a LogSoftmax layer with a size of 100. The second linear layer is a hidden layer, and has an output size of 128, and gets output directly alongside the LogSoftmax output. In the training loop, the LogSoftmax output is used to predict which character will follow, and the hidden layer outputs are fed into the RNN along with the actual next character as the next training instance.

Gated Recurrent Unit RNN: The second RNN we built used a Gated Recurrent Unit (GRU) layer instead of a linear hidden layer. While our previous RNN is essentially only able to see one layer into the past, GRUs keep track of more previous inputs than just the most recent using a series of internal gates to decide how much an input should affect each node's output. Our GRU RNN uses an initial Embedding layer to map inputs (a number from 1 to 100, indicating the character input) to vectors of size 120, which will act as our hidden layer. Each of these vectors is then input to the GRU layer, which has an output size of 120 as well. Finally, we take the output from the GRU and feed it into a linear layer with an output size of 100, representing the character prediction.

4.2 Results

We measured performance of each RNN by training each RNN on 1000 blog posts randomly selected from a single category. This ensured that the RNN could receive enough information to train while not repeating a single blog post more than a few times. We used Cross-Entropy Loss as the criterion for back propagation, and used the Adam optimizer for forward propagation. Since training each RNN could take several hours, we decided to start by training each RNN on a single category, then choosing the best RNN to compare two different categories. We quickly discovered that the Linear RNN was not as accurate as the GRU RNN, and despite several rounds of parameter tuning, we could not reduce its Cross-Entropy Loss to below 2.50. Here is an example generated blog post from our Linear RNN, which registered a Cross-Entropy Loss of 2.5398:

> *I, arsd, ant oha geve if te itt ou mounde te geor I pereaste ihat oEn I de mave amer an Na5x fer al und ond 8in t peat ounoo he link pallo tho wast I ouend at somessint At you,quot at oor bit ome ting than thay pe thing ab tha gconly time tore golles my /iouly acthet ing croe son tha faverinn thith bfo.*

As you can see, although some words like "thing" and "time" appear, many other words are jumbled or ill-formed. The GRU performed much better, reaching a Cross-Entropy Loss of around 1.40. Here is an example GRU generated blog post:

> *I'm an abcd and sleep because you can think a lot of the phone (probably the word of something they say they think it such the interesting the first than the consider the nice they were all of the same than the abcds are many parted to see look and every article that so I like to it.*

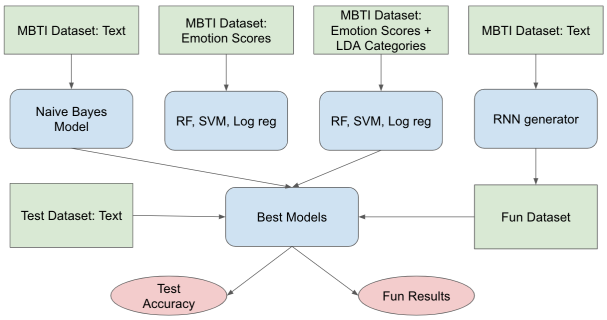
The GRU RNN was able to generate posts of almost entirely correct English words and phrases, even putting together some long segments of reasonable sounding english text: "they were all of the same", "I'm an abcd", etc. This is fairly impressive considering the RNN works at a character level and has no real understanding of the English language outside of character relationship frequency. We decided to use the GRU for our final text generation analysis. We trained the GRU RNN on a single category for 1000 iterations, and evaluated the Cross-Entropy Loss after each 50 iterations. The table with those results can be found in the appendix. The example sentences in the table were generated using the starting character "I", then predicting the next 1000 characters. After around 500 iterations, the model seems to reach a minimum loss of around 1.5. The lowest training loss reached was 1.4, but after 500 iterations the model seems to bounce around quite a bit, as shown in the table.

5 Conclusion

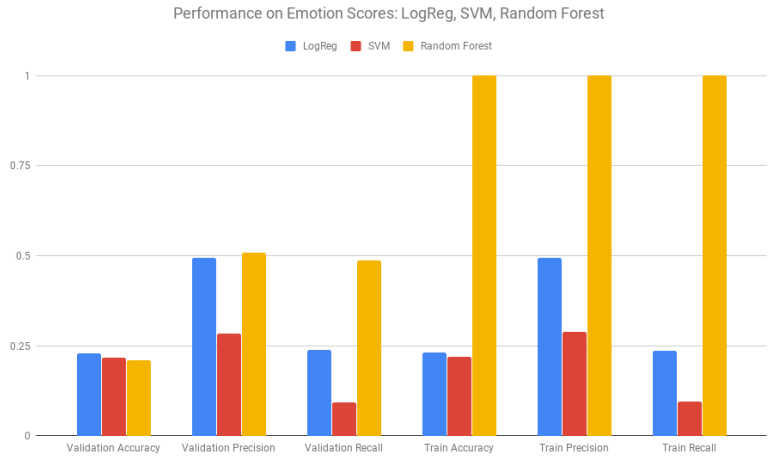
For fun, we decided to analyze our own text using our best TF-IDF model. We also analyzed four large text blocks generated by the GRU RNN. The results can be found in the appendix. Our model was correct almost 50% of the time! It was able to correctly classify Hannah as ENFP, and correctly labelled 2 of the 4 generated text blocks. In conclusion, although sentiment was a poor predictor of MBTI, the TF-IDF model was able to predict MBTI fairly well using a reasonably sized dictionary of 2000 word features. Our RNN generated more or less non-sensical text, but was able to simulate the vocabulary of a specific MBTI type fairly well. Overall, this project demonstrated the usefulness of Machine Learning for predicting Meyers-Briggs types, and proved that an RNN can generate text that accurately represents the word usage of a specific MBTI type.

6 Appendix

Project Structure



Plot of Emotional Score Model Results



LDA Model Results

Model	15 LDA + Emotion Validation Accuracy	50 LDA + Emotion Validation Accuracy
SVM(kernel='linear')	0.22034899789067727	0.21852301552781245
RandomForest(max_depth=10, n_estimators=20)	0.2286492659587235	0.23024160513679587
LogReg(penalty='l1', tol=0.0005,solver='liblinear')	0.23106660503080026	0.22566510500108744

GRU RNN Results

Iteration	Time	Cross-Entropy Loss	Example Line
100	14m 33s	2.2738	I I thing and me mich,even or the sare wart the gone the the mally a as a pinden
200	34m 18s	1.8723	I... OZmVlyboUd I I give thought, not might to fatate like do bout eting
350	58m 49s	1.7740	I've who denical adsels (we in are fact worring this? White.
450	79m 45s	1.8029	If. I do who hate ancound happens to have a seen a dirlabcd and it an times
550	90m 47s	1.6376	I'd that at... abcd is abcd accomput persone and my for I'm she here it of an abcd
950	150m 59s	1.6561	I be what your Scare she were doing and always tried strange the called the thread for
1000	159m 12s	1.4018	I have a little with that how to make term of person, and much done to

Fun Results

Blog Post	TF-IDF Prediction
Tristrum's Why Penn Essay (Category 11)	7
Hannah's Text Message Collection (Category 15)	15
Eric's Why Penn Essay (Category 15)	4
RNN Generated Blog with Category 1 Training Data	1
RNN Generated Blog with Category 1 Training Data	7
RNN Generated Blog with Category 7 Training Data	6
RNN Generated Blog with Category 7 Training Data	7

It's *Personal* - Myers-Briggs Machine Learning

Eric Chiu
Hannah Sheetz
Tristrum Tuttle

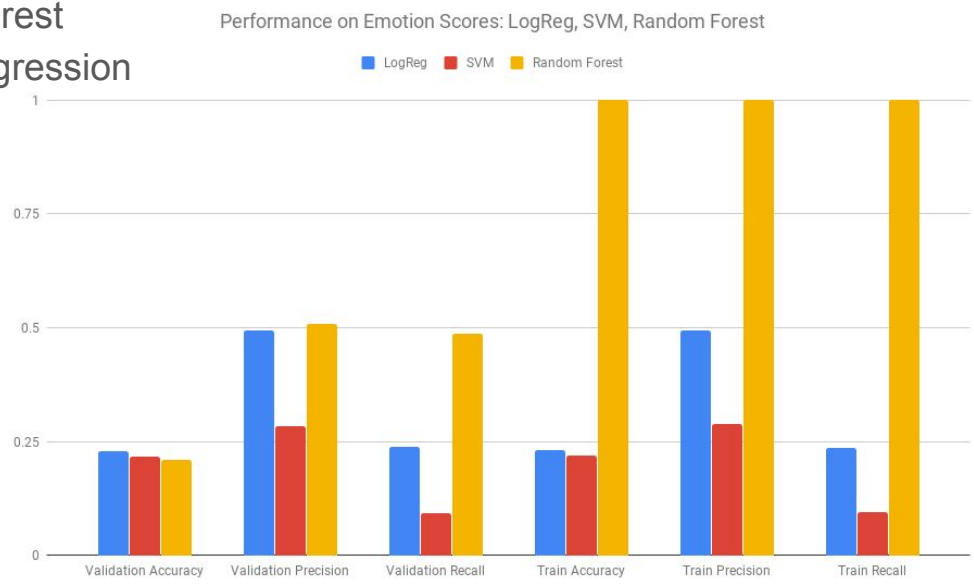
Problem

- Understanding self
- Alleviates self identification

INFP Top 20 Words

Predictive Approach

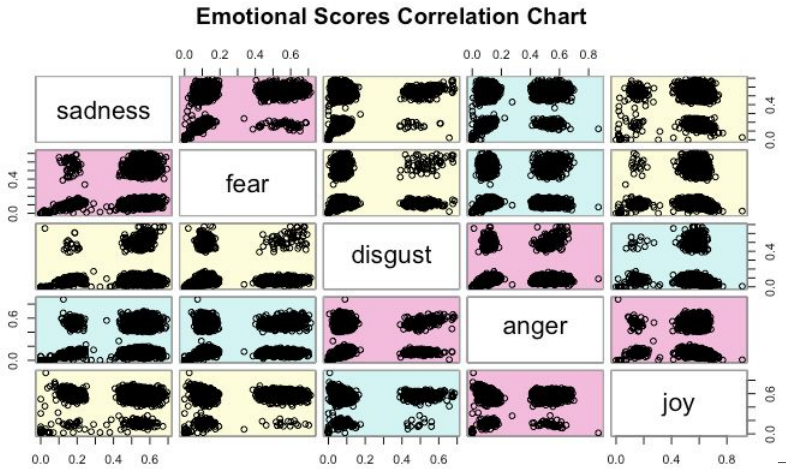
- Naive Bayes
  - TF-IDF
  - MBTI top 20 words
- LDA Models
  - Random Forest
  - Logistic Regression
  - SVM



It's *Personal* - Myers-Briggs Machine Learning

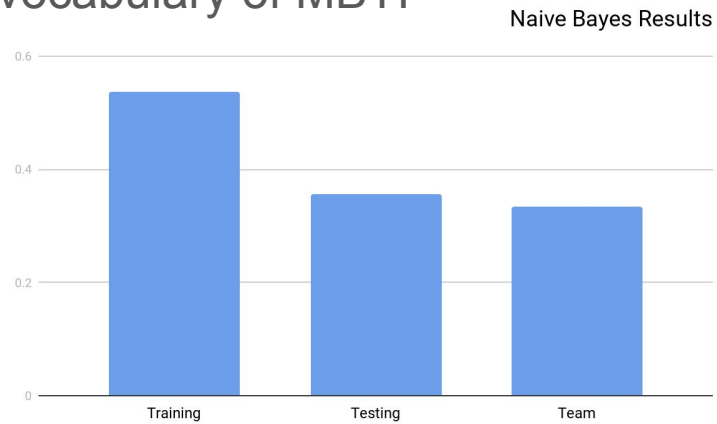
Generative Approach

- Recurrent Neural Net
  - Neural net with memory



Conclusions

- TF-IDF relatively good predictor
- Sentiment not a good predictor
- RNN simulate vocabulary of MBTI type well



GRU RNN Results

Cross-Entropy Loss	Example Line
2.2738	I I thing and me mich,even or the sare wart the gone the the mally a as a pinden
1.8723	I... OZmVlyboUd I I give thought, not might to fatate like do bout eting
1.7740	I've who denical adsels (we in are fact worring this? White.
1.8029	If. I do who hate ancound happens to have a seen a dirlabcd and it an times
1.6376	I'd that at... abcd is abcd accomput persone and my for I'm she here it of an abcd
1.6561	I be what your Scare she were doing and always tried strange the called the thread for
1.4018	I have a little with that how to make term of person, and much done to