Survey Analysis of Machine Learning Methods for Natural Language Processing for MBTI Personality Type Prediction

Brandon Cui (`bcui19@stanford.edu`)[1] **Calvin Qi** (`calvinqi@stanford.edu`)[2]

Abstract

We studied various natural language processing techniques in conjunction with machine learning techniques and evaluated their results on classifying someone's Myers-Briggs personality type based on one of their social media posts.

1. Introduction

1.1. Myers-Briggs Type Indicator

The Myers-Briggs Type Indicator (MBTI) is one of the most well-known and widely used descriptors of personality type. It describes the way people behave and interact with the world around them with four binary categories and 16 total types. They are as follows (table 1):

Energy:	**E**xtrovert / **I**ntrovert
Information:	**S**ensing / **IN**tuition
Decision:	**T**hinking / **F**eeling
Lifestyle:	**J**udging / **P**erceiving

Table 1: The Myers-Briggs Type Indicator Attributes

Each person's MBTI personality type is defined as the collection of their four types for the four categories, using the bolded identifying letter for each. For example, one who derives their energy mostly from being around other people (E), trusts their gut and uses intuition to interpret information in the world (N), thinks rationally about their decisions (T), and lives life in a carefully planned manner (J) rather than a spontaneous one would have the personality type ENTJ. This is the personality schema that we will be using throughout this paper.

1.2. Goal

We set out to predict one's MBTI personality type from one of their social media posts. Our algorithm takes in an excerpt of text as input and outputs the predicted MBTI personality label (e.g. ENTJ). We will survey a variety of methods for this task, looking both at classical Supervised Learning and at the efficacy of deep learning with actively trained word embeddings on such a task. Then we perform comparisons and analysis on their resulting error and accuracy to find the method that is most effective for this problem.

[1]Stanford University, Department of Computer Science
[2]Stanford University, Department of Mathematics.

1.3. Motivation

In a world where communication is increasingly social media based, we are interested in finding out if there is a strong relationship between ones use of language online and their actual personality. There are two main implications of this study. First is the possibility of one's 'online persona' as distinct from their in-person one, which suggests that people have a likelihood of behaving in a completely different way online. Second is that social media messages, being a method of communication with its own quirks and styles of language use distinct from prose or speech, contain a certain amount of representational power and reflect the personality of the author.

2. Dataset and Features

2.1. Dataset

We obtained our data from the *(MBTI) Myers-Briggs Personality Type Dataset* from Kaggle. It provides the text of the 45-50 most recent social media posts for 8,600 users along with the user's MBTI personality type. This gives us 422,845 total labeled points in the form (post text, MBTI type). The posts are drawn from the PersonalityCafe online forum, a platform for all kinds of conversations and discussions, and they obtain the labels by allowing the user to input MBTI type as account info. This could lead to many inherent data biases, as we will discuss in future sections.

We shuffle the data and split it into 70-15-15 portions for training, validation (hold-out), and test sets respectively.

2.2. Tools

We collect, process, and analyze all of our data using Python. We also utilize the Natural Language Toolkit

(NLTK) library for much of our text preprocessing, Numpy for matrix computations, sk-learn for conventional learning algorithms, and PyTorch for deep learning.

2.3. Data Analysis

The dataset is quite skewed and is not uniformly distributed among the 16 personality types. For example, the most common label, INFP, occurs 89,796 times whereas the least frequent, ISFJ, only occurs 8,121 times. We found that when training on the data in this original form, the model tends to overfit on the predominant type(s) while underperforming on the others, so to remedy this we perform data duplication and reduction by doubling and halving until no class is twice as large as any other. As a result we are training on 328,650 data points instead of the original amount.

Two example data points are:

ENTP:	"I'm finding the lack of me in these posts very alarming"
INFJ:	"What? There's a series! Thanks for letting me know :)"

2.4. Preprocessing

Since the data is raw text and online chatting language is often irregular and oddly formed (i.e. abbreviations, emojis, punctuation) we apply a significant amount of text preprocessing.

- Converting to lowercase (but we want to incorporate capital letter usage too so we include that as a feature)
- Using NLTK lemmatizer to combine word forms
- Identifying special text (URLs, numbers, dates, emojis) with regex and replacing them with special escape tokens to standardize
- Separating punctuation from text
- Assigning words to numerical indices based on frequency in our training set

2.5. Feature Selection

We begin by featurizing the posts using bag of words. This includes all of the preprocessing and added features/tokens from above. We let B denote the size of our bag, meaning we consider the occurrences of the first B most frequent words in our dataset and treat all other words as a special unknown token. After tuning B as a hyperparamter we found this most effective when $B = 50,000$.

We also append even more additional features: bigrams, skip grams, part of speech tags, capital letter count

3. Methods

3.1. Baseline

For our baseline, we used a multiclass Softmax classifier on all 16 personality types, with minimal preprocessing (only the first two steps mentioned above), using minibatch Stochastic Gradient Descent with a minibatch size of 100 and a learning rate of $\alpha = 0.1$. Softmax regression is a generalization of binary logistic regression to distinguish between multiple classes, with a normalized output that provides confidence probabilities for each class.

More formally, we have $h_\theta(x)$ outputting a vector of 15 real values between 0 and 1 representing the prediction confidence of each class (with the 16th being implied from the other 15). Our parameters are $\theta_1, \theta_2, \ldots, \theta_{15}$ and each output is given by

$$p(y = i|x; \theta) = \phi_i = \frac{\exp(\theta_i^T x)}{1 + \sum_{j=1}^{15} \exp(\theta_j^T x)}$$

The baseline performs with training accuracy 19% and test accuracy **17%**, which beats randomly choosing classes for 6.25%.

3.2. Individual Personality Categories

One flaw in the full 16-class approach is that there is a lot of overlap between classes and not necessarily any clear way to distinguish between them. For example, INTJ and INTP are treated as distinct classes even though they overlap completely in most aspects and only demonstrate a minor difference. Given that social media text is already quite ambiguous and varied, this forces our classifier to have to find tiny differences among highly similar, noisy data, which is very difficult and not fruitful. These classes aren't actually independent, which thwarts a classifier that seeks to find complete separation.

Instead, we transition to building binary classifiers for each of the four personality categories (i.e. E/I, S/N, T/F, J/P) then aggregating the four outcomes to get the overall predicted MBTI label. This provides a host of advantages:

- Distinguishing between actual dichotomies gives more strongly separable data which improves accuracy dramatically
- There is more training data for each class when we split in halves (e.g. E/I) compared to 16 parts.
- By training four different classifiers, each one can be optimized separately to best fit its own purpose, instead of having a one-size-fits-all model
- By having a different prediction confidence for each personality trait, we get more meaningful output that can

show for example if someone is clearly 90 percent extroverted but only 70 percent thinking over perceiving. This gives more gradations and nuance.

3.3. Naive Bayes

One method of text classification is Naive Bayes. This chooses to model $p(x|y)$, the likelihood of our data, using the assumption that words/features are probabilistically independent of each other conditioned on the labels, computing $p(x|y) = \prod_i p(x_i|y)$. This tends to be effective because: (1) it doesn't require much training or computational power and can obtain all of its parameter estimates from proportions in the data itself, and (2) it can estimate and incorporate the influence of each word/feature on the class's likelihood

For the result below we present the naive bayes results with the traditional add-1-laplace smoothing, and get the following accuracies (Table 2):

Classifier	Train Accuracy	Test Accuracy
E/I	0.799	0.750
S/N	0.869	0.845
T/F	0.701	0.624
J/P	0.641	0.603
Overall	0.3174	0.2586

Table 2: Classification accuracies for Naive Bayes

However, overall, the train accuracy of Naive Bayes is 32% and the test accuracy is **26%**, which is a noticeable improvement from our Softmax baseline of 17%. The large gap between training and test errors shows heavy overfitting and weak generalization, which we will remedy in the next model with regularization.

3.4. SVM

Support Vector Machines (SVMs) are very well-performing, robust, and customizable supervised learning algorithms. A SVM, alike logistic regression, seeks to find a hyperplane separating the classes in the dataset, but it uses the *hinge loss* and has the added optimization goal of maximizing *margin*. The optimization problem is:

$$\min_{\gamma,w,b} \frac{1}{2} \|w\|^2$$
$$\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m$$

We also add $L2$ regularization so the model generalizes more effectively. Then, we tuned the following hyperparameters by running trials varying the values on a log scale and then honing in on smaller ranges and found the best values to be (table 3):

SVM Hyperparameter	value
SGD minibatch size:	100
Learning Rate:	$\alpha = \frac{1}{\lambda(t+t_0)}$
Regularization Rate:	$\lambda = 0.005$
Bag of Words size:	$B = 50,000$

Table 3: Table of optimal SVM hyperparameters

Then, we train using minibatch Stochastic Gradient Descent until the error converges. The plot of dev error of the E/I classifier for the first few epochs of one particular trial is shown in Figure 1, and the error for the full MBTI prediction is in Figure 2, where error is shown the proportion of incorrect predictions on the dev set. After tuning all

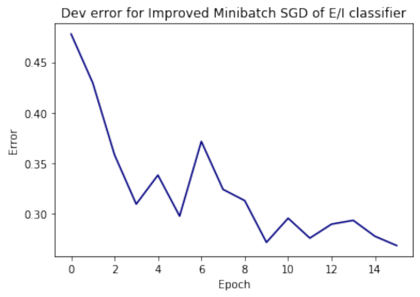

Figure 1. Single trait error

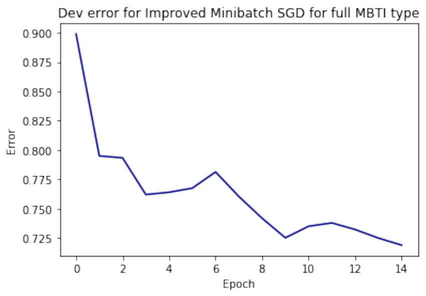

Figure 2. Total 4-trait error

of these parameters and performing the error analysis that will be described next, our best classifier has a training accuracy of 33.7% and a test accuracy of **32.6%**, improving upon both our Baseline and Naive Bayes models.

3.5. Error Analysis

In the process of refining our model and deciding how to obtain our best SVM model, we performed ablative error analysis by removing components one by one from our full SVM model to find out which parts of our data+classification pipeline caused the most significant improvements.

These are the ablative error analysis results on an intermediate E/I classifier obtaining 76% dev accuracy before we obtained our best SVM models (Table 4):

Component Removed	Dev Accuracy
Full System	76.1%
Tuning α and λ	75.6%
Tuning B	74.7%
Equalizing Classes	72.3%
Preprocessing Text	68.7%

Table 4: Ablative error analysis on the various components in our data processing and classification system

We find that removing the preprocessing step has the largest effect on our classifier accuracy. This is reasonable because our entire model's ability to understand text and find relationships depends on receiving input that is consistent and meaningful, which comes from preprocessing and feature selection. Thus, with this in mind, we focused more effort on improving the preprocessing stages, which led to the addition of lemmatization, bigrams, skip grams, part of speech tags, and capital letter counts to our text processing and features.

3.6. Deep Learning

3.6.1. Encoder Decoder Framework

Our main deep learning framework was drawn from neural machine translation, where they use an encoder-decoder framework (Wu et al. 2016). Below we describe the utilized framework:

Encoder Framework
For our encoding system we used a multi-layer long-short term memory (LSTM) recurrent neural network as the encoder (Fig 3.)

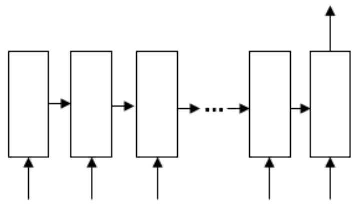

Figure 3. Encoding Mechanism

The overall idea is to represent a single sentence with a high dimensional vector. We considered every word in the vocabulary to be represented by a high dimensional word embedding. As seen in previous image captioning work all word embeddings were actively trained to fit our specific model (Karpathy et al 2015) (Lu et al 2017).

Decoder Framework
Our decoder framework was always a 3-layer feed-fowards neural network, with every activation being rectified linear units (ReLU) and the last layer outputs the probability of each class via a softmax function.

3.6.2. Training, Loss Function, and Fine-Tuning

For every experiment, we trained the neural network for 25 epochs, with a minibatch size of 500. We also used Xavier initialization in order to have better gradient flow over our deep network (Glorot et al 2010). Additionally, for the encoder we used the RMSProp optimizer while for the decoder we used a Adam optimizer (Kingma et al 2014). Our loss function is the traditional cross-entropy loss which is defined as follows:

$$l(y, \hat{y}) = -\sum_i y_i log(\hat{y}_i)$$

here, y represents the true label's value and $\hat{y}$ is the predicted label probability from the softmax function. We note that y is always a one-hot vector representing the class label for the given datapoint.

We varied multiple hyperparameters including dropout size, hidden size, embedding size, and number of encoding hidden layers in a random fashion as described in (Bergstra et al. 2012).

3.6.3. 16-class classifier

We initially trained a single 16-class classifier to try to see if our deep network could obtain a more favorable result than our softmax baseline. However, after tuning multiple hyperparameters our best training accuracy was 55% and the test accuracy was **23%**, This indicates that there was heavy overfitting when considering all 16-classes conglomerated together.

3.6.4. 4 binary classifiers

Since the division of all 16-classes based on such short text passages proved to be too difficult, opted to create 4 different binary classifiers, one for every category. We present some of our results below (Table 5)

We note that the random search of hyperparameters yielded varying outcomes, but overall we were still able to achieve slightly better results using deep learning. Also, between classifiers there was no strong correlation between the various hyperparameters, since depending on the personality class, different optima were found during training, which come from different parameters. Our network reached 40% training accuracy and **38%** test accuracy.

MBTI Personality Type Prediction

Classifier	Embedding Size	Hidden Size	Dropout	# Hidden Encoding Layers	Dev Accuracy	Test Accuracy
E/I	256	256	0.1	1	**0.8974**	**0.8951**
E/I	128	300	0.15	2	0.8955	0.8851
S/N	256	256	0.1	1	**0.8856**	**0.89848**
S/N	200	300	0.15	1	0.8691	0.8665
T/F	512	256	0.1	1	0.6910	**0.6909**
T/F	256	256	0.15	1	**0.6912**	0.6848
J/P	256	256	0.15	1	**0.6605**	**0.6765**
J/P	128	300	0.1	1	0.6594	0.6837

Table 5: Comparison of various deep learning hyperparameters and result dev and test accuracies. Here the bolded values indicate the best dev/test accuracy for that specific classifer.

4. Results

4.1. Comparison of Different Methods

Our best performing models were as follows (Table 6):

Model Type	Train Accuracy	Test Accuracy
Softmax (baseline)	19%	17%
Naive Bayes	32%	26%
Regularized SVM	34%	33%
Deep Learning	40%	38%

Table 6: Comparison of different methods for MBTI-classification

We find that the Regularized SVM on individual personality types yields better accuracy than our Baseline and Naive Bayes models, and deep learning further outperforms SVM. This is reasonable since a deep learning architecture involves many more parameters and a much more sophisticated set of operations, which gives it more representational power and a much larger hypothesis class at the expense of significantly longer training time.

4.2. Discussion

From an absolute standpoint, the overall final accuracy still isn't jaw-droppingly high, since it doesn't even surpass 50 percent. However, when we examine each personality category, the performance is much better, and it is clear that our models are able to distinguish effectively within these personality dichotomies.

The error that remains can be due to a variety of factors. One is that the data could have a large amount of inherent bias. Since users are only drawn from one particular forum, we are receiving a very limited sample of the actual population; in particular, the joining of that forum could already favor certain personality types and act as a layer of selection. In addition, the ground truth MBTI types are self-reported, so there is a lot of room for error for people who don't remember their type exactly or who have changed in their personality/worldview/lifestyle since the last time they took the MBTI test. In fact, one psychological study reports that when people are tested just a few months apart, 50 percent end up with different results, so these personalities are fluid by nature and change with time. This could also come from a flaw in the test itself, or perhaps it can be attributed to the temperament and inconsistency of the people taking them.

Moreover, there are many who dispute the quality of the MBTI schema itself. Some psychologists believe that the four categories are not the most salient traits of personality, and others have noted that the traits aren't entirely orthogonal, so there is actually overlap and dependence among them. These factors add variation, uncertainty, and user error, which together make it very challenging and somewhat implausible to have an extremely accurate classifier.

5. Conclusion

5.1. Future Work

Moving forwards, we hope to incorporate richer data and features to allow for a stronger understanding of the input text as well as improved performance. Our dataset was quite limited and didn't include any other user information or metadata for posts, such as time or the surrounding conversation, and that additional data would be hugely important for understanding the bigger picture of one's personality as well as the context of each post.

For deep learning, it would be desirable to look towards other mechanisms of representing word embeddings, including char and k-char word embeddings (Karpathy 2017) and pretrained GloVe vectors on our corpus. Additionally, it should be possible to achieve even better results by adding on attention mechanisms to our current framework. Lastly, because of new discoveries in NLP always arising from current research, it'd be interesting to see results from implementing these most cutting edge methods.

We would also like to try unsupervised learning to find out if people's social media posts naturally form clusters based on personality, and to see if these clusters coincide with or have any similarities to the MBTI types.

6. Contributions

Both group members contributed to the ideas, planning, and decision making involved in this project. Brandon Cui worked on the data parsing, Naive Bayes model, and deep learning. Calvin Qi worked on text preprocessing, error analysis, and SVM optimization. All other remaining work was shared.

7. Bibliography

Bergstra J., Bengio Y. Random Search for Hyper-Parameter Optimization. 2012. Journal of Machine Learning Research.

Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks. 2010. Internaional Conference on Artificial Intelligence and Statistics.

Karpathy A., Fei-Fei L. Deep Visual-Semantic Alignments for Generating Image Descriptions. 2015. IEEE Computer Vision and Pattern Recognition.

Karpathy A. The Unreasonable Effectiveness of Recurrent Neural Networks. 2017. [online] Available at *http://karpathy.github.io/2015/05/21/rnn-effectiveness/*

Kingma D., Ba J. Adam: A method for stochastic optimization. 2014. *CoRR*.

Lu J., Xiong C., Parikh D., Socher R. Knowing When to Look: Adaptive Attention via a Visual Sentinel for Image Captioning. 2017. IEEE Computer Vision and Pattern Recognition.

Wu Y., Schuster M., Chen Z., Le Q., Norouzi M. et al. Google's Neural Machine Translation System: Briding the Gap between Human and Machine Translation. 2016. *arXiv preprint*.