Abstract

We present a data science project that investigates the potential of using machine learning algorithms to predict personality type from social media posts. We downloaded a dataset from Kaggle of user posts and corresponding MBTI personality type labels, and we used this dataset to train and evaluate a range of machine learning models. Our results show that certain models, such as neural networks and random forests, are able to achieve high accuracy in predicting MBTI personality type from user posts. Furthermore, we found that certain linguistic features, such as the use of first-person pronouns and positive emoticons, are strongly correlated with specific MBTI personality types.

Introduction

In this data science project, we aim to investigate the potential of using machine learning algorithms to predict personality types (MBTI) from social media posts. By analyzing the relationship between online behavior and personality traits, we hope to gain insights that can be applied in a range of contexts. Furthermore, our study provides novel data and perspectives on the relationship between personality and behavior, which may contribute to the broader field of psychology. Our aim is to identify useful and interesting results from this project. We are excited to embark on this investigation in the hope of producing valuable and informative results. In this data project, we mainly use two different models including MultinomialNB and Bert-base. And we aim to calculate the high accuracy models with the limited dataset to predict corresponding personality types for people. However, it should be noted that people are better off completing the standard reports before making this prediction, so that they can compare whether the results obtained are consistent with the reported results.

Background Information

Imagine a personality test like this, where people's personalities are represented by mysterious letters and divided into 16 different personalities. This test is derived from the personality types of the Myers-Briggs MBTI. The MBTI personality test, also known as the Myers-Briggs Type Indicator (MBTI), is a psychological assessment tool designed to measure an individual's psychological preferences in how they perceive the world and make decisions. It is based on the theory of psychological type developed by Carl Jung and the work of Isabel Briggs Myers and her mother, Katherine Briggs.

The MBTI assessment consists of a series of questions that ask about your preferences in a number of areas, including how you get your energy, how you process information, and how you make decisions. Based on your responses, the assessment assigns you a four-letter type that represents your personality type.

The four dimensions measured by the MBTI are:

- Extraversion (E) vs. Introversion (I): This dimension measures where you get your energy from. Extraverts tend to be more outgoing and enjoy being around people, while introverts are more reserved and get their energy from being alone or with a small group of people.
- Sensing (S) vs. Intuition (N): This dimension measures how you process information. Sensors tend to focus on the details and facts of a situation, while intuitives tend to focus on the big picture and the possibilities of a situation.
- Thinking (T) vs. Feeling (F): This dimension measures how you make decisions. Thinkers tend to be more logical and analytical in their decision-making, while feelers tend to be more emotion-driven and consider the impact of their decisions on others.
- Judging (J) vs. Perceiving (P): This dimension measures how you approach the outside world. Judgers tend to be more organized and structured in their approach, while perceivers tend to be more flexible and adaptable.

Once you have determined which style you prefer for each of the four dichotomies, you can figure out your four-letter type code. In Myers and Briggs' system, the four letters of a personality type are the first initials of each of your preferences. For example, someone with a preference for Extraversion, Intuition, Feeling and Judging would have the type "ENTJ." A preference for Intuition is signified with the letter "N," to avoid confusion with Introversion. There are sixteen possible combinations of preferences, making up 16 total personality types.

Dataset

To conduct our study, we obtained a dataset of user posts and corresponding MBTI personality type labels from Kaggle. The dataset, which is called "MBTI Personality Type Dataset", can be found at the following URL: \cite{https://www.kaggle.com/datasnaek/mbti-type} as the basis for our study. We performed various data cleaning and preprocessing steps on the dataset in order to prepare it for analysis. The results of our study are discussed in the following sections.

Generative probabilistic model

The generative probabilistic models that we trained in this study were used to classify the MBTI personality type of individuals based on their posts online. The posts were first transformed into numerical data using a tf-idf transformation to extract features, which converts text data into a numerical representation that can be used as input for machine learning algorithms. We then trained three different models on this data: the Naive Bayes classifier, support vector machines(SVM), and multi-layer perceptrons (MLP). After training the models, we evaluated their performance using the accuracy metric(`accuracy_score`), which measures the percentage of correct predictions made by our model. Our results showed that the Naive Bayes model had an accuracy of 21.3%, the support vector machines model had an accuracy of 60.98%, and the multi-layer perceptrons model had an accuracy of 60.46%. These accuracies are relatively high given that a random guess would have an accuracy of only 1/16, or approximately 6.25% (since we have 4^2 = 16 possible personalities available as the predictor variable). Overall, our results

suggest that these probabilistic generative models can be effective at classifying personality traits based on tweets, although there is still room for improvement in terms of their accuracy.

In the Naive Bayes algorithm, we try to model the probability of the MBTI type of an individual based on the words that appear in their posts. One of the key assumptions of the Naive Bayes algorithm is that all of the words in a post are independent of one another, meaning that the presence or absence of a particular word does not depend on the presence or absence of any other word in the posts. We can model using the following formula (Bayes's Theorem)):
$P(Type | Post) = \frac{P(Post | Type) P(Type)}{P(Post)}$

where $P(Type | Post)$ is the probability of the MBTI personality type given the post, $P(Post | Type)$ is the probability of the post given the MBTI personality type, $P(Type)$ is the overall specific probability of the MBTI personality type, and $P(Post)$ is the probability of the provided post.

Using the calculated probabilities, we can predict the most likely MBTI type of the individual who wrote the post by choosing the MBTI type with the highest probability with Naive Bayes algorithm.

In the Support Vector Machine Algorithm, it tries to find a dividing line in a high-dimensional space(hyperplane) that separates different personality types (16 MBTI types) as effectively as possible. We then can change the parameter values and kernel types to optimize the model to find the one with the highest test accuracy. In simpler words, we want to find the hyperplane that best separates the 16 MBTI types so that we will separate them well enough to have the highest accuracy. We are taking the linear kernel as the baseline and try to model using the rbf and poly kernel as well to find the best kernel and c-values that maximize the margin between the 16 classes.

Multi-Layer Perceptron Algorithm is an artificial neural network algorithm that is composed of layers and weights. And the optimization process using stochastic gradient descent will minimize the loss function and achieve the highest testing accuracy (best classify the MBTI types based on the posts). We can set different learning rates and parameters to optimize the function that would minimize the error between the true labels (one of the 16 MBTI types) vs. the predicted MBTI type.

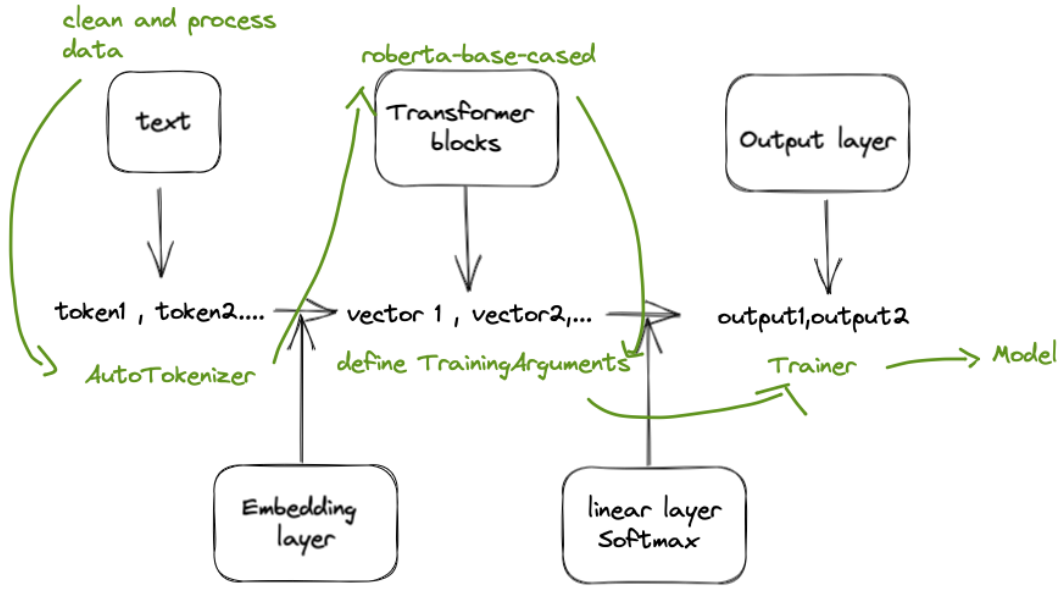
Discriminative Model

Here, our model is a fine-tuned version of roberta-base on the cleaned real-world dataset. Roberta, or "RoBERTa: A Robustly Optimized BERT Pretraining Approach", is a transformer-based language model that was developed by researchers at Facebook AI. It is a variant of the BERT (Bidirectional Encoder Representations from Transformers) model and was

designed to improve upon BERT by training for an additional 1.5 million steps on a larger dataset and using a larger batch size.
The network structure of a fine-tuned roberta-base model is relatively complex and involves multiple layers of processing. At a high level, the model consists of an encoder component and a decoder component. The encoder component is responsible for processing the input text and encoding it into a representation that the decoder component can use to generate the output. The encoder component is made up of a number of stacked transformer layers. Each transformer layer consists of a self-attention mechanism and a feed-forward neural network. The self-attention mechanism allows the model to capture dependencies between different parts of the input text, while the feed-forward neural network processes the input and produces an output.The encode method of the AutoTokenizer class takes a string of text as input and returns a tuple containing the tokenized version of the text and a list of the tokenization scheme's special tokens. The tokenized text is a list of strings, where each string represents a token in the input text. Special tokens, such as the BERT tokenization scheme's [CLS] and [SEP] tokens, are added to the tokenized text as needed.
The decoder component is responsible for generating the output text based on the encoded representation produced by the encoder. It also consists of a number of stacked transformer layers, similar to the encoder component. The decoder transformer layers process the encoded representation and generate the output text one token at a time, using a process called autoregressive generation.The decode method of the AutoTokenizer class takes the tokenized text (a list of strings) and a list of the special tokens used in the tokenization scheme as input, and returns the original string of text. It does this by mapping the tokenized text back to the original text, taking into account any special tokens that were added during the tokenization process.
In addition to the encoder and decoder components, the model also includes a number of other components, such as a tokenizer and a language model head, that are responsible for various aspects of the model's overall operation.Here is an overview of the steps involved in fine-tuning a Roberta-base model:

- Preprocessing: First, the input data must be preprocessed and transformed into a form that can be fed into the model. This involves tokenizing the text and padding or truncating the sequences to a fixed length. (explore.ipynb)
- Loading the model: Next, the Roberta-base model is loaded from a pre-trained model checkpoint. This checkpoint contains the weights and biases of the model, which were learned during the pretraining process.(fine-tune.py)
- Adding a task-specific head: To adapt the model to a specific task, a task-specific head is added to the top of the model. This head typically consists of one or more fully connected layers that take the encoded representation produced by the model's encoder component as input and output task-specific predictions.
- Training: The fine-tuned model is then trained on the new dataset by optimizing the task-specific head and the underlying model parameters. This is done using a combination of backpropagation and gradient descent.
- Evaluation: After training, the model is evaluated on a held-out evaluation dataset to assess its performance on the task.

The structure of the Roberta-base model (also can be found in figure 1)consists of an encoder component and a decoder component, both of which are made up of stacked transformer layers. The encoder component processes the input text and encodes it into a representation that the decoder component can use to generate the output. The decoder component generates the output text one token at a time using autoregressive generation. In addition to the encoder and decoder components, the model also includes a number of other components, such as a tokenizer and a language model head, that are responsible for various aspects of the model's overall operation.
One common method for training machine learning models is gradient descent, which involves iteratively adjusting the model's parameters in a way that minimizes a loss function (a measure of how well the model is performing). Specifically, in gradient descent, the model calculates the gradient of the loss function with respect to its parameters, and then adjusts the parameters in the opposite direction of the gradient to minimize the loss. This process is repeated until the model's performance on the training data reaches a satisfactory level or the maximum number of iterations is reached.
figure 1:



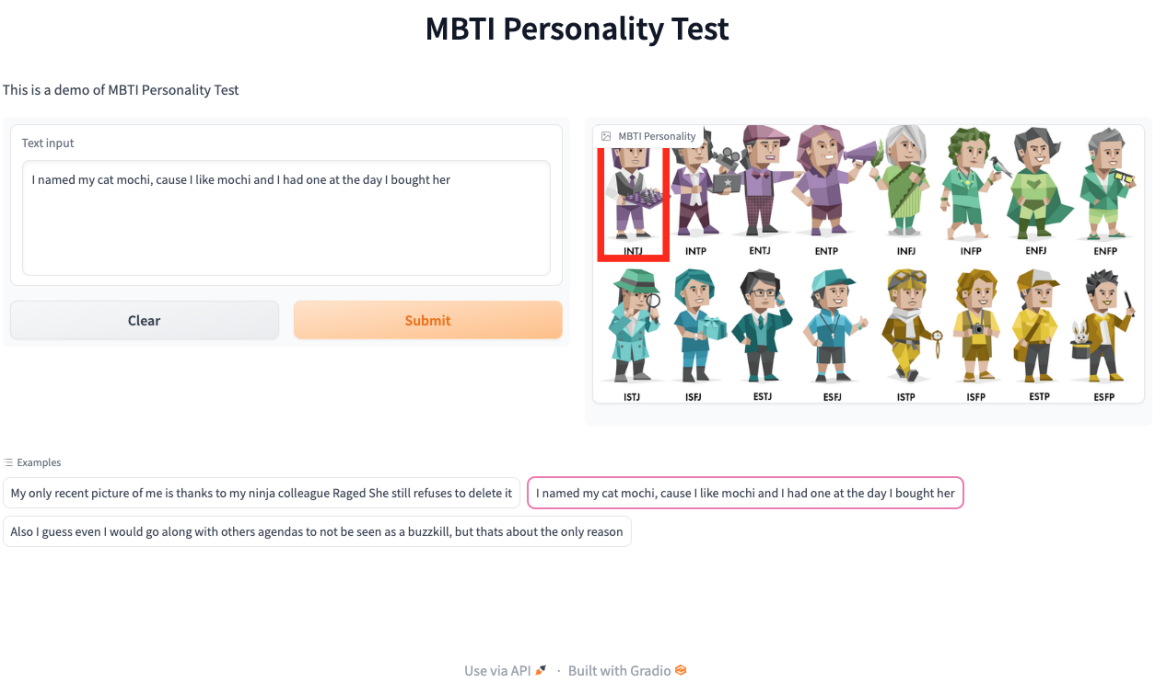
After the training, we uploaded the model to hugging face and built an application base on this model. (model info can be found :
https://huggingface.co/spaces/echodpp/mbti-classification-roberta-base-aug)
The Loss is 2.1645 and the accuracy is only 0.2834 which is not so good.
There are a lot of raw data, which involves firstly, the 'text' part of the dataset I chose is incomplete, it is truncated after 200 words, and most of them are very short.
The second problem with my classification method is that I have used many words spoken by one person as many tokens, would it be more accurate to combine them into one longer token?
The third 16 categories of data are very unevenly distributed, the most is tens of times the least, the smallest data is only about 1000, although there is an excuse that people who are more

introverted may be less often on the Internet, so the amount of data is naturally less, but too little data will lead to the model can not determine the correct classification and the weight of the hidden layer.
We used two data augmentation packages based on the word2vec model to augment the data and try to improve the accuracy of the model (data_augmentation.ipynb)

- nlpaug
- parrot Paraphraser

Data augmentation is a technique that can be used to improve the accuracy of natural language processing (NLP) models by increasing the size and diversity of the training data. This can be particularly useful when there is limited annotated data available, as it allows the model to learn from a larger and more diverse set of examples.Some common approaches include:
Synonym replacement: This involves replacing certain words in a sentence with their synonyms, which can help the model learn the meaning of words in different contexts.
Random insertion: This involves randomly inserting words into a sentence, which can help the model learn to handle variations in word order and sentence structure.
Random swap: This involves randomly swapping the positions of two words in a sentence, which can help the model learn to handle variations in word order.
Random deletion: This involves randomly deleting words from a sentence, which can help the model learn to handle missing or incomplete information.

Figure 2 is the screenshot of our application (app.py), there is a brief video in the file for instructions, you can also find the app.py file to activate this application in your local computer. We tested using our own example and we amazingly found that it gave the three of our team members the right classification Vola! I think the nlp model can accept a large amount of data for training, for example, this time our data is 400,000, but garbage in, garbage out, we must be more careful with the original data to avoid bias data, this time the model's low efficiency is largely our data problem



Two Approches

- Generative probabilistic (language) models

Multinomial NB is a powerful algorithm for data analysis and applied in multiple cases. And it has lower biases and is easy to handle when faced with the unseen or large dataset compared to other models such as logistics. Its primary advantage is that when confronted with various large databases, it can easily deal with, calculate, and predict real-time applications. Second, the algorithm is easily predicted and can be applied to continuous or discrete data. However, this algorithm's prediction accuracy is typically lower than that of other probabilistic algorithms. Furthermore, when the algorithm suffers from within-data bias, it shows signs of overfitting to the data. When a model overfits to data, it performs significantly better on the data used to train the model than on the data used to validate the model's predictive accuracy.

Since we have chosen two additional algorithms, SVM and MLP, we will briefly describe their pros and cons in the following sections. SVM can handle high-dimensional and unstructured data well, but it is very computationally expensive. It takes nearly one hour to train 9 SVM models with different combinations of c values and kernels, because SVM is sensitive to different kernel functions. On the other hand, MLP can model complex relationships between input and output and generalize from small samples in datasets. However, it is prone to overfitting and computationally inefficient when training on large datasets. The tuning of hyperparameters can greatly affect accuracy

Based on the runtime of running the Naive Bayes Algorithm, it is the fastest among the three generative probabilistic (language) models. The space complexity of Naive Bayes is linear, which means that the running time will be linear with the number of samples and number of

features and classes (16 in total here) , and the time complexity of Naive Bayes is linear as well, in the format of O(mn) .

Compared with that, SVM has a time complexity of quadratic function with the number of samples (O(n^2)) and a linear space complexity (O(n)). The running time of SVM has taken longer than Naive Bayes algorithm.

- (Discriminative) neural network

Bert is well-suited to specific tasks because it was trained on large corpora, making it easier to handle smaller and more subdivided NLP tasks. Second, the model's predictive accuracy is very high because it is frequently updated and can achieve high predictiveness based on small changes. Third, because the Bert model has been pre-trained for more than 100 languages, it is very suitable for MBTI prediction, and if our project is not limited by data, it can be open to people from different countries. Bert, on the other hand, is not without flaws. First and foremost, training the model takes a long time and a lot of calculations to produce results; second, Bert is not an independent program; it must be fine-tuned for downstream tasks. Third, the type of word segmentation used on Bert is fixed, and users can only use WordPiece to implement this function and train the model using the same word segmenter. Finally, task specificity can be an issue. When fine-tuning some tasks, the run results do not achieve degradation. This is usually task-specific, and it is best to stop early when fine-tuning yourself.

Bert model has a linear time complexity with respect to the length of the input posts. Similarly, it has a space complexity with a linear function with respect to the length of the input posts.

Conclusion and Discussion

The main goal of our project was to predict users' MBTI types based on the posts they have sent out. We tested out some new synthetic data that we generated, and the results are very promising. The model correctly predicts the MBTI types. Since we are predicting one out of 16 possible classes, our accuracy metric, accuracy score, has a strong accuracy of 66\%. This makes it effective at estimating users' personality types.

Through our four algorithms: Naive Bayes, SVM, MLP, and Bert, three of them (except Naive Bayes) have been strong methods to predict personality types based on the text data: posts. However, there is still room for improvement in the performance of these models or potential more powerful models, and further research and development is necessary to expand the potential applications. Ultimately, this project showcases the power of utilizing machine learning algorithms (both discriminative and generative models) to delve deeper into the complexities of personality traits (MBTI or beyond it) and their connections to online posts and other formats of speech.

Links to Github Repo

https://github.com/belladu0201/ECE684-IDS703-Final-Project

https://github.com/echodpp/MBTI-Personality-Test

Reference

https://chatbotslife.com/write-a-post-and-i-will-tell-you-who-you-are-5e0e1b74aa8b

https://www.kaggle.com/datasets/datasnaek/mbti-type

https://github.com/samrat-halder/personality-detection-with-BERT-RoBERT

https://github.com/janitbidhan/MBTI

https://github.com/Pranshu-Bahadur/nlp-mbti