Personality Type Prediction using Natural Language Processing and Machine Learning

Ragnor Wu, Haotong Wu

Abstract

To classify MBTI based on different styles of comments, this project used different machine learning models to train the dataset. Our project is a multi-classification problem. We compared the accuracy of KNN, SVM, logistic regression, XGBoost, FCNN, and other classifiers, and finally selected SVM and used cross-validation to find the most suitable parameters. We compared the results of 16 types and 4 types and found that 4 types were more accurate. Due to the imbalance of different categories of data, our accuracy is not very high, but it can be improved by reducing the categories.

1 Introduction

1.1 Background

The Myers-Briggs Type Indicator (MBTI) is a widely used personality test categorizing individuals into 16 distinct personality types. It can accurately show the different character traits of people through four dimensions.

The four dimensions of the MBTI each represent a fundamental aspect of personality. With the Extraversion-Introversion dimension, the Sensing-Intuition dimension, the Thinking-Feeling dimension and the Judging-Perceiving dimension.

To categorize more broadly, we can distinguish SJ, SP, NT, and NF as the four groups of personality types in the MBTI framework. Each group is characterized by a dominant set of cognitive functions and preferences in decision making and problem solving.

1.2 Motivation

Despite the popularity of the MBTI, its validity has been questioned, especially in predicting behavior and language style. In this project, we aim to explore the predictive power of the MBTI by using natural language processing and machine learning to predict personality types based on an analysis of an individual's writing style.

1.3 Objective

In this project, we aim to train different models using KNN, Logistic Regression, Naive Bayes, SVM, Decision Tree, Random Forest, AdaBoost, XGBoost and FCNN to predict the mbti type of a person based on the posts they have written. We would like to find ways to improve the accuracy of the results by tuning the parameters of the models and using oversampling techniques to balance the dataset.

2 Dataset

2.1 Data Overview

The dataset used in the project is based on the MBTI Type dataset[1]. And the dataset of over 8600 rows are obtained from the PersonalityCafe forum, consisting of each individual's 4-letter MBTI code/type and a section of the last 50 things they have posted.

The dataset contains 2 types of data, the comment and their corresponding MBTI lable. The dataset is imbalanced, so we need to process it.

2.2 Data Processing

2.2.1 Data cleaning

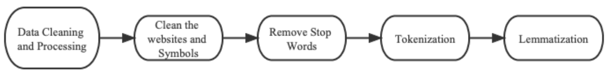


Figure 1: Working flow for data cleaning

Data cleaning is an essential step in data pre-processing, which aims to remove irrelevant or harmful data and improve the accuracy and efficiency of subsequent analysis. In this project, we performed the data pre-processing according to the flow in Figure 1. We loaded a portion of text data and observed that there were some URLs and symbols that were not useful for our analysis and needed to be cleaned. Then, we developed a function to processed the remaining data for further analysis.

Firstly, we remove English stop words, which are words that have no actual meaning in natural language, such as "the" and "and". They do not contribute to our analysis, but can increase processing complexity and computational burden. Next, we performed tokenization, which breaks the text data into individual words (or tokens) for further processing. Tokens play an important role in text analysis as they are the smallest unit of text that can be used to perform tasks such as word frequency counting and semantic analysis.

Finally, we performed lemmatization, which converts words to their base form. In natural language, different forms of a word can create ambiguity, such as ”run,” ”running,” and ”ran,” which can represent different meanings in different contexts. Therefore, converting words to their base form can help eliminate ambiguity caused by word form changes and improve the accuracy of text processing.

After completing these steps, we generated word clouds for different types of data and observed that the most common words in these word clouds were ”m” and ”s.” This indicated that some abbreviations were not removed during data cleaning. However, these individual letters can have an impact on the overall data analysis. Therefore, in the next step, we removed these common abbreviations and generated new word clouds. We found that the new word clouds were more reasonable and informative.

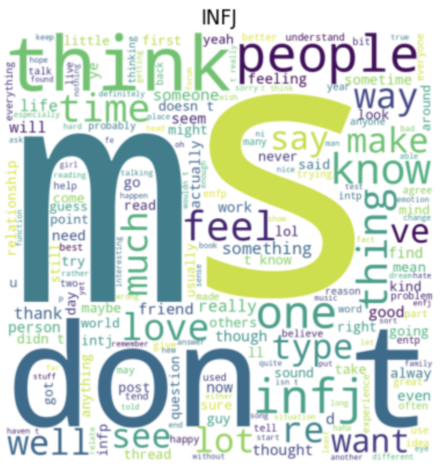


Figure 2: Wordcloud before cleaning

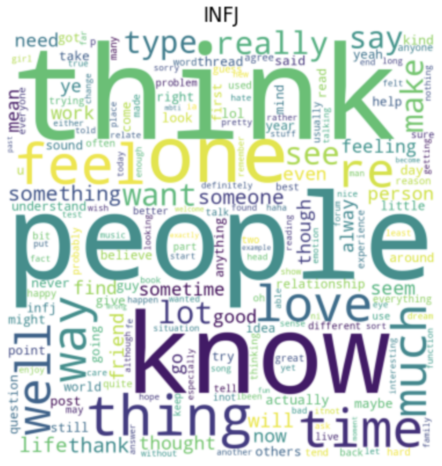


Figure 3: Wordcloud after cleaning

As shown in Figures 2 and 3, we compared the word clouds of the INFJ type before and after removing the common abbreviations. This process helped us to improve the quality of our data and eliminate unnecessary noise, which was crucial for accurate and meaningful analysis.

2.2.2 Data Splitting

After cleaning and preprocessing the text data, the next step is to split the data into training and testing sets. This is necessary to evaluate the performance of the machine learning model on unseen data and to avoid overfitting.

In this project, we used the train_test_split function from the sklearn library to split the data into training and testing sets. We split the data using an 80/20 ratio, where 80% of the data was used for training and 20% for testing.

We then applied TF-IDF vectorization to the text data. TF-IDF stands for

"Term Frequency-Inverse Document Frequency", which is a numerical statistic that reflects how important a word is to a document in a collection or corpus. It helps to give more weight to important words while downgrading the unimportant ones. The max_features parameter is set to 5000, which means that the vectorizer will consider the top 5000 most frequent words in the corpus.

Next, we used label encoding to convert the categorical labels into numerical labels. The LabelEncoder class from the sklearn library was used to encode the labels as integers ranging from 0 to the number of classes minus one. This ensures that the model can process the labels as numerical values.

Finally, we returned the processed dataset, which are used to train and evaluate the machine learning model.

2.2.3 Oversampling

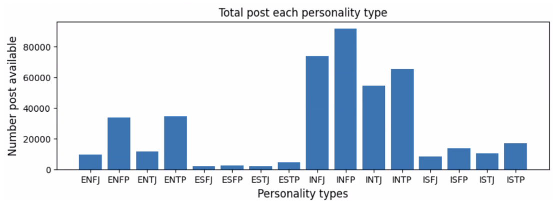


Figure 4: Distribution of different types

The data we use for training and testing is imbalanced, as shown in Figure 4 with different distribution for each class. To address this issue, we use oversampling to balance the data. Specifically, we apply the Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic samples for the minority class. After oversampling, each class has the same number of data, which makes the training process more fair.

3 Methodology

3.1 Model Building

In this project, we used various machine learning algorithms to solve a multi-class classification problem. Specifically, we compared the performance of several classifiers including KNN, SVM, logistic regression, XGBoost, FCNN, and others. For each algorithm, we used the standard implementation available in the corresponding library, and did not deviate from the standard algorithm in any significant way.

For SVM, we used cross-validation to find the most suitable parameters. We searched through a range of C values and selected the one with the highest cross-validation accuracy. This helped us avoid overfitting the model to the training data.

For FCNN, we used a dense neural network architecture with three layers. We used the ReLU activation function for the first two layers, and the softmax activation function for the final layer since we have a multi-class classification problem. We trained the model for 10 epochs using the Adam optimizer and a batch size of 64. We used the sparse categorical cross-entropy loss function since our target labels were integers.

Overall, we used standard implementations of various machine learning algorithms and did not deviate from their standard usage in a significant way.

3.2 Evaluation

In the evaluation section, we assessed the performance of our models using three metrics: accuracy, F1 score, and classification report.

The accuracy is a measure of the proportion of correct predictions made by the model. We calculated the accuracy using the accuracy_score function provided by the sklearn.metrics module.

The F1 score is a weighted average of the precision and recall of the model. It is a measure of the model's accuracy on a dataset and ranges from 0 to 1, with 1 being the best score. We calculated the F1 score using the f1_score function provided by the sklearn.metrics module.

We also generated a classification report for each model, which provided a detailed breakdown of the precision, recall, and F1 score for each class. We used the classification_report function provided by the sklearn.metrics module.

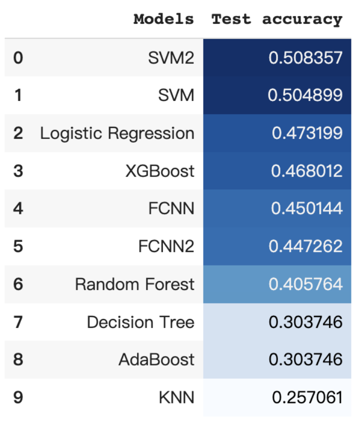
We then organized the results into tables for easy comparison. Specifically, we created a table that showed the accuracy of each model on the test dataset and another table that showed the F1 score of each model on the test dataset. These tables allowed us to easily compare the performance of our models and select the best one for our task.

4 Results and Discussion

We trained our models with and without oversampling for the 16-class classification task. However, we only observed an improvement in accuracy with oversampling for the Logistic regression model, while the SVM model showed a small decrease in accuracy. But other models, such as KNN, saw a significant decrease in accuracy, which suggests that the models were overfitting to the oversampled data. This suggests that oversampling can enhance the performance of Logistic Regression, but may not be beneficial for other models.

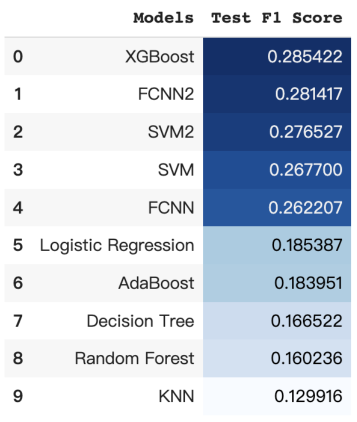
As a result, we trained our models without oversampling and found that the SVM model achieved the highest accuracy. Figure 5 and Figure 6 show a comparison of our model performances in different models.

By the overall observation of accuracy and f1-score we selected the SVM model with C = 0.24 and without oversampling as our final model for the 16-class classification task. While our approach was successful in achieving higher



	Models	Test accuracy
0	SVM2	0.508357
1	SVM	0.504899
2	Logistic Regression	0.473199
3	XGBoost	0.468012
4	FCNN	0.450144
5	FCNN2	0.447262
6	Random Forest	0.405764
7	Decision Tree	0.303746
8	AdaBoost	0.303746
9	KNN	0.257061

Figure 5: Accuracy without oversampling

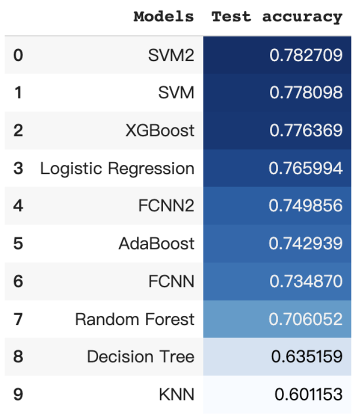


	Models	Test F1 Score
0	XGBoost	0.285422
1	FCNN2	0.281417
2	SVM2	0.276527
3	SVM	0.267700
4	FCNN	0.262207
5	Logistic Regression	0.185387
6	AdaBoost	0.183951
7	Decision Tree	0.166522
8	Random Forest	0.160236
9	KNN	0.129916

Figure 6: F-1 score without oversampling

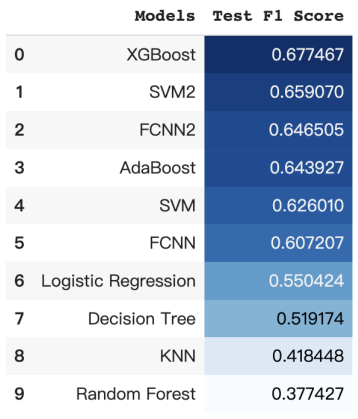
accuracy, we also acknowledge the shortcomings of our method, such as the underfitting that occurred without oversampling.

Since it is not very accurate to classify the data into 16 types, we expanded the categories to 4. This greatly improved the accuracy of the model during the training process. Our results are shown in Figure 7 and Figure 8, where the SVM with C=0.27 is the most appropriate model overall in the 4 categories.



	Models	Test accuracy
0	SVM2	0.782709
1	SVM	0.778098
2	XGBoost	0.776369
3	Logistic Regression	0.765994
4	FCNN2	0.749856
5	AdaBoost	0.742939
6	FCNN	0.734870
7	Random Forest	0.706052
8	Decision Tree	0.635159
9	KNN	0.601153

Figure 7: 4 types Accuracy without oversampling



	Models	Test F1 Score
0	XGBoost	0.677467
1	SVM2	0.659070
2	FCNN2	0.646505
3	AdaBoost	0.643927
4	SVM	0.626010
5	FCNN	0.607207
6	Logistic Regression	0.550424
7	Decision Tree	0.519174
8	KNN	0.418448
9	Random Forest	0.377427

Figure 8: 4 types F—1 score without oversampling

5 Future Look

After working on this dataset, we have realized that the biggest challenge we faced was the class imbalance problem. We observed that there were significantly fewer data points starting with the letter ”E” compared to those starting with ”I”. This could be attributed to the fact that ”E” represents extroverted personalities, while ”I” represents introverted ones. As such, people who do not engage in social interactions as much tend to have more data points starting with ”I”, leading to the class imbalance problem.

In the future, we would attempt to address this issue by exploring different methods for oversampling or undersampling the data to achieve better training results. Furthermore, in the MBTI system, some individuals may not fit perfectly into a single letter category, but rather have a tendency towards a certain trait in each dimension. If we can collect data that reflects these percentages of different features, it could potentially make our predictions more accurate. However, this would require more data and more complex classification features, which may necessitate the use of more sophisticated models.

Given another five years to work on this problem, we would try to obtain a larger and more diverse dataset that reflects these complex traits, as well as explore more advanced modeling techniques such as deep learning. Overall, our experience with this project has taught us the importance of data preprocessing and the challenges of working with imbalanced datasets.

References

[1] Datasnaek, “Mbti type dataset,” https://www.kaggle.com/datasnaek/mbti-type, 2017, accessed on May 13, 2023.