# Myers-Briggs Type Indicator (MBTI) Personality Prediction

**Satya Shah (1002161494), Devarsh Vora (1002159763), Vraj Patel (1002166096)**

## ABSTRACT

This paper introduces a machine learning approach to predict Myers-Briggs Type Indicator (MBTI) personality types based on survey responses. By leveraging advanced algorithms, our model accurately classifies individuals into their respective personality types, contributing to the intersection of psychology and artificial intelligence.

## INTRODUCTION

The MBTI personality prediction project utilizes machine learning methodologies to predict personality types based on individuals' behavioral patterns and preferences. The MBTI divides individuals into 16 personality types using four pairs of characteristics. By analyzing a dataset of MBTI survey responses, the machine learning model is trained to accurately classify individuals into one of the 16 personality types. This project aims to deepen our understanding of human behavior and has potential applications in psychology, team dynamics, and personalization. It showcases the intersection of psychology and artificial intelligence, highlighting the predictive capabilities of machine learning in categorizing human personality traits.

The Myers-Briggs Type Indicator (MBTI) sorts people into one of 16 personality types, determined by four dimensions:

•INTROVERSION (I) – EXTROVERSION (E)

• INTUITION (N) – SENSING (S)

• THINKING (T) – FEELING (F)

• JUDGING (J) – PERCEIVING (P)

Fig. 1. Different types of personalities.

## 1. DATASET DESCRIPTION

The dataset focuses on exploring the Myers Briggs Type Indicator (MBTI), a widely used personality type system categorizing individuals into one of 16 distinct types based on preferences in four axes: Introversion/Extroversion, Intuition/Sensing, Thinking/Feeling, and Judging/Perceiving. This system, rooted in Carl Jung's cognitive functions theory, has enjoyed extensive usage across various domains including businesses, online platforms, research, and personal interest. Despite its popularity, the MBTI's validity has faced scrutiny due to inconsistencies in experimental results. The dataset aims to investigate if any discernible patterns exist between specific MBTI types and writing styles, thereby contributing to the ongoing discourse surrounding the test's efficacy in predicting and categorizing behavior.

The dataset consists of over 8600 rows, with each row representing a person's MBTI type (a 4-letter code denoting their personality type) along with a section containing the last 50 posts made by that person. The posts are separated by "|||" (3 pipe characters).

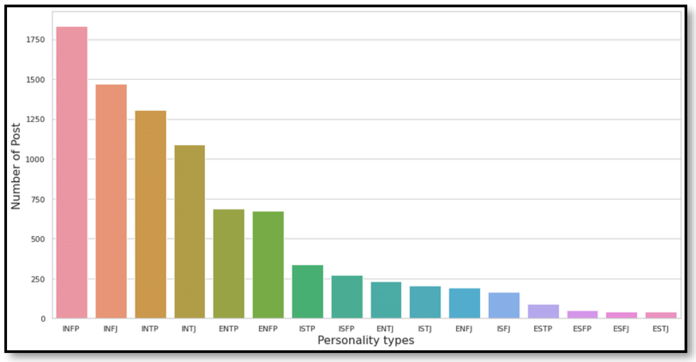


Fig. 2. Displaying the aggregate count of posts attributed to each personality type.

### 1.1 Exploratory Data Analysis (EDA):

We meticulously examined the dataset's composition and delved into the distribution of personality types. Through correlation and comparative analyses, we unveiled intriguing patterns and insights into behavioral trends across different MBTI types. While mindful of limitations, our findings fuel our quest for deeper understanding, guiding future investigations into the rich tapestry of human personality dynamics.

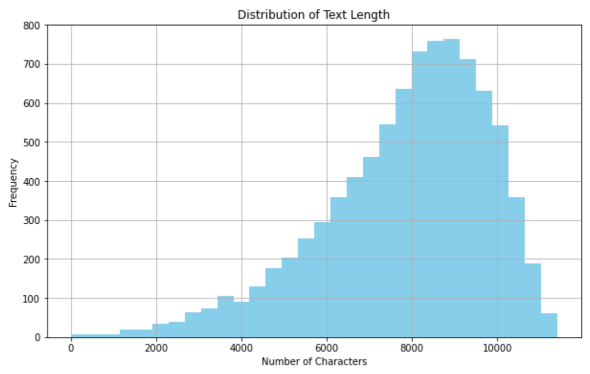


Fig. 3. This shows that the length of characters in each sentence ranges from 2000 to 11500 approx. and most of the text in the data has characters ranging from 6000 to 10000.

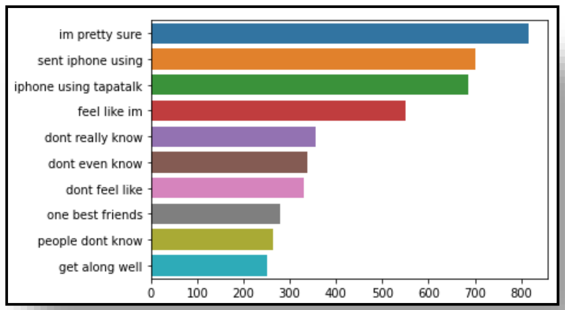


Fig. 6. n gram plot(n=3)

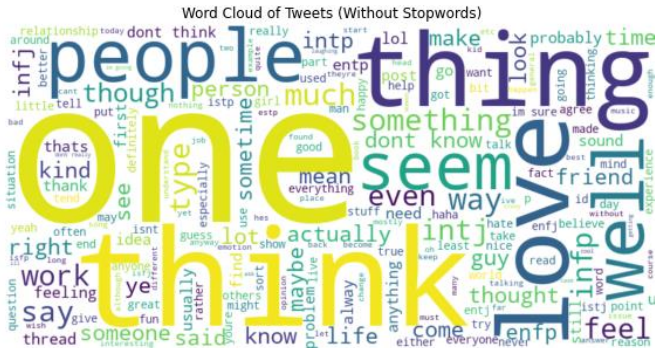


Fig. 4. Word cloud of the most common words in the tweets without stop words.

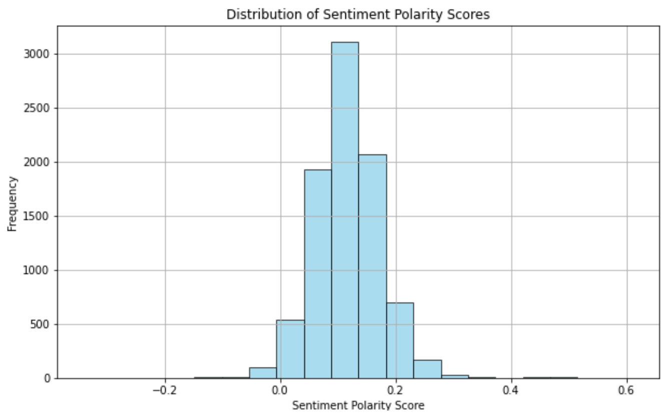


Fig. 5. This shows that most of the posts are neutral in this dataset.

## 2. PROJECT DESCRIPTION

### 2.1 Description (Data Preprocessing):

#### 2.1.1 Regular Expressions (Regex):

Regular expressions (Regex) are powerful tools used in machine learning and natural language processing to define search patterns within text data. They consist of sequences of characters that represent rules for matching strings of text. These patterns can include literal characters, wildcards, and special symbols, enabling ML algorithms to efficiently search for, extract, and manipulate text based on complex criteria. Regex is particularly useful for tasks such as data cleaning, text preprocessing, and information extraction in various ML applications.

#### 2.1.2 Stop Words Removal:

Stop words removal is a preprocessing step in natural language processing (NLP) and machine learning (ML) that involves filtering out common words that are considered non-informative or irrelevant to the task at hand. These words, such as "the," "is," "and" etc., occur frequently in text but typically do not carry much semantic meaning. By removing stop words from text data before analysis or modeling, ML algorithms can focus on the more meaningful words, improving performance and efficiency in tasks like text classification, sentiment analysis, and information retrieval.

### 2.1.3 Tokenization:

Tokenization refers to the process of breaking down text into smaller units, typically words or sub words, called tokens. These tokens serve as the basic building blocks for natural language processing tasks. Tokenization involves splitting a text document into individual words or sub words, removing punctuation and other non-essential characters, and possibly converting all text to lowercase for consistency. Each token represents a distinct unit of meaning within the text, making it easier for machine learning algorithms to analyze and understand the underlying information.

Tokenization is a fundamental step in various NLP tasks such as text classification, sentiment analysis, named entity recognition, and machine translation. It allows ML models to process and learn from textual data more effectively by representing words or sub words as numerical inputs that can be fed into neural networks or other machine learning algorithms.

### 2.1.4 Padding:

Padding typically refers to the process of adding extra elements (usually zeros) to input data sequences to make them uniform in length. This is often necessary when working with sequences of variable lengths, such as sentences in natural language processing tasks or time series data in sequential models like recurrent neural networks (RNNs) or transformers.

For example, if you have a batch of sentences of varying lengths:

- "The cat sat on the mat."
- "The quick brown fox jumped."
- "She sells sea shells by the sea shore."

You might pad these sentences with zeros to make them all the same length, such as:

- "The cat sat on the mat.0000"
- "The quick brown fox jumped."
- "She sells sea shells by the sea shore"

This padding ensures that all inputs to the model are of the same shape, which is often required for efficient computation in neural networks. After processing, the padded elements are typically ignored or masked to prevent them from affecting the model's predictions.

## 2.2 HANDLING IMBALANCED DATA

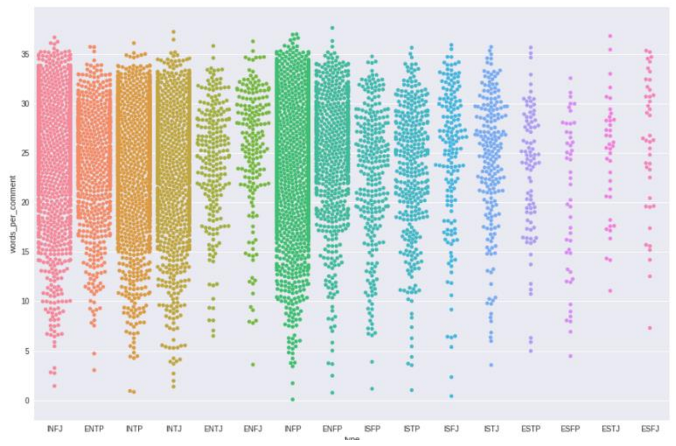


Fig. 7. It shows words per comment for each category.

### 2.2.1 Random Oversampling:

To address class imbalance in our project on MBTI personality prediction, we've implemented machine learning strategies such as random oversampling. Random oversampling involves augmenting instances in the minority class until it matches the count of the majority class. Unlike SMOTE, which generates new instances based on nearest neighbours, we opt to duplicate instances from the minority class randomly. However, we're cautious about potential pitfalls; careless application may lead to overfitting as it introduces redundancy and may accentuate specific patterns unique to the minority class.

## 2.2 REFERENCES USED FOR PROJECT

- *Ontoum, S., & Chan, J. H. (2022). Personality Type Based on Myers-Briggs Type Indicator with Text Posting Style by using Traditional and Deep Learning. arXiv preprint arXiv:2201.08717. [Jan 2022]*
  Link

- *M. A. Akber, T. Ferdousi, R. Ahmed, R. Asfara and R. Rab, "Personality Prediction Based on Contextual Feature Embedding SBERT,"* 2023 IEEE Region 10 Symposium (TENSYMP), *Canberra, Australia, 2023, pp. 1-5, doi: 10.1109/TENSYMP55890.2023.10223609.*
  Link

## 2.3 OUR APPROACH

### 2.3.1 CNN for Text Classification:

Convolutional Neural Networks (CNNs) are used in natural language processing, an application of the architectural framework that was initially created for image processing, to classify text. The CNN model extracts localized information by interpreting text inputs as one-dimensional word embedding sequences through the use of convolutional layers. The recovered features are subsequently reduced by max-pooling layers, which concentrate on the most pertinent data. Filters are used to capture patterns of various sizes across the input text. The network gains the ability to automatically recognize and exploit hierarchical patterns in text data through training, facilitating effective classification into several categories. In tasks including sentiment analysis, subject categorization, and document classification, this method has demonstrated encouraging outcomes.

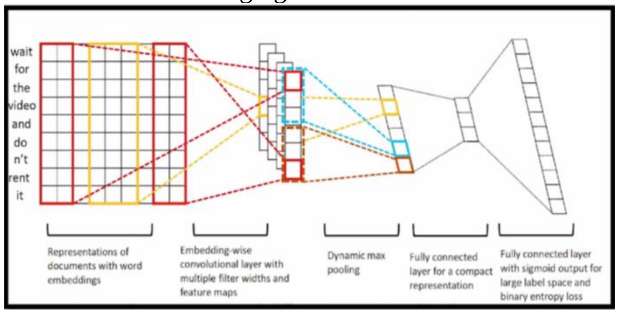

wait for the video and do n't rent it
Representations of documents with word embeddings
Embedding-wise convolutional layer with multiple filter widths and feature maps
Dynamic max pooling
Fully connected layer for a compact representation
Fully connected layer with sigmoid output for large label space and binary entropy loss

Fig. 8. This shows CNN application for Text Classification

**Fasttext:** Facebook's AI Research group developed the freely available library FastText, which is a potent tool for effective text classification and learning representations. It works with the idea of word embeddings, where words are represented as dense vectors in a continuous space. FastText stands out due to its capacity to create embeddings for character n-grams as well as individual words. This allows it to process new words more effectively and understand morphological subtleties than traditional word-based embeddings. FastText is widely used for applications like as sentiment analysis, language modeling, and text classification; it is especially well-suited for contexts with constrained resources and big datasets "Crawl-300d-2M" is a pre-trained word embedding model that was developed using a large text corpus that was crawled across the internet. Compact vector representations of words used in machine learning that capture semantic relationships based on contextual usage are called word embeddings. A 300-dimensional vector is used to represent each word, as indicated by the "300d" symbol. The "2M" stands for around two million unique words or tokens, which is the dataset that the model was trained on. In many natural language processing applications, such as text classification, sentiment analysis, and machine translation, these word embedding models are used as feature extractors.

**CNN Implementation (In our code):**
For text classification, a convolutioal neural network (CNN) architecture is used in the sequential model. It starts with an Embedding layer that uses non-trainable 300-dimensional pre-trained word embeddings. ReLU activates two Conv1D layers, each of which has 256 filters with a kernel size of 5, for the purpose of extracting features. Layers of MaxPooling1D with a pool size of 5 are then applied. GlobalMaxPooling1D further reduces the dimensionality of the feature map. Len(label_encoder.classes_) determines the classification probabilities for each class in the final Dense layer, which is activated by softmax. For added abstraction, two more Dense layers are added, each having 256 neurons and triggered by ReLU. With sparse categorical cross-entropy loss, the model is put together.

### 2.3.2 BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMERS:

BERT (Bidirectional Encoder Representations from Transformers) is a notable development in the field of natural language processing (NLP). BERT was created by Google researchers and was first presented in a seminal work by Jacob Devlin et al. in 2018. It has significantly improved the performance of a number of NLP tasks, including question answering, sentiment analysis, and language inference. Its primary characteristic is its capacity to train language representations in both directions, which enables it to understand a word's context by taking into account all of the surrounding context inside a phrase. This bidirectional training sets a new benchmark for the creation of more complex and context-aware natural language processing (NLP) systems, diverging from previous models that were only concerned with unidirectional text comprehension.

**Architecture of BERT**

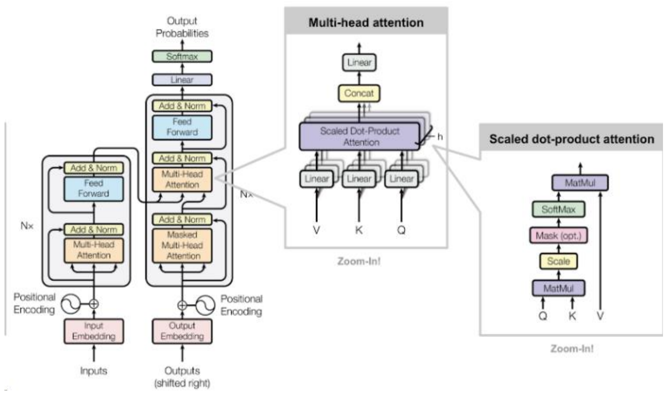


Fig. 10. This shows Architecture of BERT

**How Does Self Attention Work?**

Input representation: Every word in the input sequence is represented by a vector. These vectors are used to create three different kinds of vectors—Query (Q), Key (K), and Value (V) vectors—for every word using trainable linear transformations.

Scoring: The query vector of each word is multiplied by the key vector of every other word in the model to determine a score. This score determines the weight given to other input sequence segments while encoding a particular word.

Normalization: The scores are then lowered by the dimension squared of the Key vectors in order to guarantee stable training. To help determine the effect weight of the words, these scores are normalized using the softmax algorithm. Normalized into probabilities, the scores are all positive and sum up to one.

Weighted Sum: The normalized scores are multiplied by the value vectors. In this stage, the weights of the input words are expressed in a weighted manner according to how relevant each word is to the other words in the sentence. This process yields a new representation, which is the sum of these weighted Value vectors for each word, combining the contextual relevance of each word with its original meaning.

Output: The self-attention layer generates a fresh set of vectors, each of which is a weighted sum of the vectors it received. These outputs could subsequently be sent to the network's higher tiers for additional processing. Reword this without altering the paragraph's meaning.

**How Does Self Attention Work?**

Using 16 classes of BERT (Bidirectional Encoder Representations from Transformers), the create_model function builds a deep learning model for classification tasks. To construct and set up the model, TensorFlow and the Transformers library are used. Below is an explanation of every element that makes up the architecture of the model:

Input Layer:

- input_word_ids: Sequences of token IDs are fed into this layer, which functions as the model's input. maxlen, which establishes the maximum length of the input sequences, determines the form of the input. The standard data type for indices, tf.int32 (32-bit integer), is given.

BERT Layer:

- bert_layer: The Hugging Face Transformers library's pre-trained BERT model (bert-base-uncased) is loaded by the model. With from_pt=True, it is configured to convert from PyTorch checkpoints. BERT receives sequences of token IDs as input and produces a series of embedding vectors for every token.

- bert_outputs: The BERT layer generates a tensor, where each token in the sequence is indexed in the first dimension, and each token's encoded features are stored in the second dimension. Since BERT models can generate many layers of output, we extract the outputs from the final layer using the notation [0].

Dense Layer for Classification:

- Bert_outputs[:,0,:] is used to select the BERT layer's output corresponding to the initial token ([CLS] token), which combines data for classification. Throughout all input examples, this slicing operation selects the embedding for the [CLS] token.

- A Dense layer with 16 units and a softmax activation function receives this embedding after that. The logits for each of the 16 classes in your classification problem are converted into probabilities by the softmax function, which matches these 16 units to the 16 classes.

Model Configuration:

- A TensorFlow model is configured with the inputs and outputs. The input_word_ids layer is the value of the inputs parameter, while the output of the Dense layer (pred) is the value of the outputs parameter.

- Loss Function: The categorical_crossentropy loss function is used when compiling the model, and it is suitable for multi-class classification problems in which the target labels are given in a format that is singularly encoded.

- Optimizer: A learning rate of 0.00001 is applied while using the Adam optimizer. Because of its capacity for flexible learning rate, Adam is a well-liked option for deep learning model training.

- Metrics: A statistic called accuracy is used to assess how well the model performs both during training and validation.

For a multi-class classification task, this function builds a BERT-based model that is specially customized. With the help of a basic neural network classifier and the potent feature extraction capabilities of BERT, it can forecast class probabilities. We apply the normal procedure in BERT-based models for sentence-level tasks to the categorization of the [CLS] token using its embedding.

Various BERT models were tested, including tiny, medium, tiny, mini, small, and BERT-based. There is less space needed for this compact BERT variant. Bert-base-uncased was our last attempt, and it produced an accuracy of 69%.

### 2.3.3 Difference between our approach and the paper that we used for our references.

The papers we used for references calculated the true positive based on a different approach. They used an approach that determines accuracy by comparing traits across four pairs of eight traits (E-I, S-N, T-F, J-P), our approach mandates an exact match across all four traits for a prediction to be deemed correct.

The paper “**Personality Type Based on Myers-Briggs Type Indicator with Text Posting Style by Using Traditional and Deep Learning**” used RNN. In the modeling process, embedding words are employed to represent indexes as dense vectors of fixed size, with a vocabulary length of 256. The decision to utilize "CONV1D" on "Recurrent Neural Networks" stems from its ability to move along a single axis, making it suitable for sequential data like text. To enhance the performance of the "Recurrent Neural Networks" model, a "Bi-directional Long Short-Term Memory (BI-LSTM)" with a size of 64 was incorporated. This addition enables the "Recurrent Neural Networks" to capture sequence information in both directions: backward (from future to past) and forward (from past to future). Also, we did more EDA by plotting different graphs as compared to this model. Also, we did not opt for removing stop words and doing lemmatization when using BERT because these can be useful for the BERT model. Also, we used CNN which can be seen in this section further.

The paper “**Personality Prediction Based on Contextual Feature Embedding SBERT**” used SBERT and different ML models. Data cleaning was almost the same as we did in our approach. SBERT employs a pretrained model, such as All-MPNetBaseV, to generate sentence embeddings instead of word embeddings. This method produces contextualized embeddings, wherein the embedding for a given sentence is influenced not only by the specific words or sentences but also by the surrounding context. Initially, the input sentence is tokenized into subwords using the Byte Pair Encoding (BPE) algorithm. Subsequently, the All-MPNetBaseV model processes these subword tokens through various layers of the transformer architecture, generating contextualized embeddings for each subword. These contextualized embeddings for all subwords are then combined into a fixed-length representation of the input sentence, typically by aggregating them using mean or maximum aggregation across all subword embeddings. In the case of the ALLMPNET-Base-V2 model, the fixed-length representation is a 768-dimensional vector.

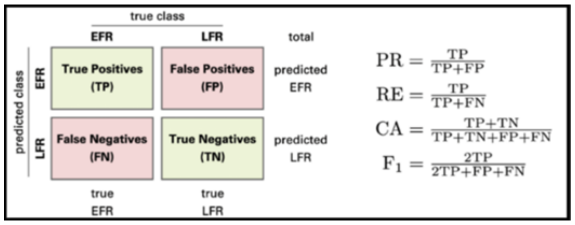
### 2.4 Difference in accuracy between our approach and the paper that we used for our references.

#### 2.4 1 OUR ACCURACY

| **Model** | **Evaluation Matrix** |
|---|---|
| CNN | • Precision: 0.48<br>• Recall: 0.56<br>• F1 Score: 0.51<br>• Accuracy:0.56 |
| BERT-small<br><br>(Params: 28M) | Accuracy: 0.645 |
| BERT- based-uncased.<br><br>(6 Classes – INFP, INFJ, INTP, INTJ, ENTP, ENFJ)<br><br>(Param: 110M) | Accuracy: 0.745 |
| BERT-based-uncased<br><br>(Params: 110M) | • Precision: 0.705<br>• Recall: 0.68<br>• F1 Score: 0.692<br>• Accuracy: 0.69 |

Table.1. Accuracy comparison of all the models used.

**NOTE:** We got high training accuracy but low testing accuracy because our dataset is highly imbalanced.



$$\mathrm{PR} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$$
$$\mathrm{RE} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$$
$$\mathrm{CA} = \frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}}$$
$$\mathrm{F}_1 = \frac{\mathrm{2TP}}{\mathrm{2TP+FP+FN}}$$

Fig. 9. This shows Evaluation Matrix, here F1 score, and accuracy is used.

### 2.4.2 PAPER ACCURACY

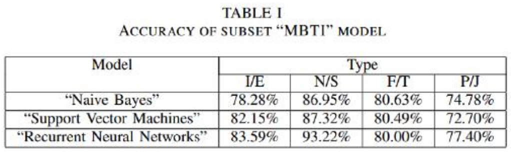


TABLE I
ACCURACY OF SUBSET "MBTI" MODEL

| Model | Type | | | |
|---|---|---|---|---|
| | I/E | N/S | F/T | P/J |
| "Naive Bayes" | 78.28% | 86.95% | 80.63% | 74.78% |
| "Support Vector Machines" | 82.15% | 87.32% | 80.49% | 72.70% |
| "Recurrent Neural Networks" | 83.59% | 93.22% | 80.00% | 77.40% |

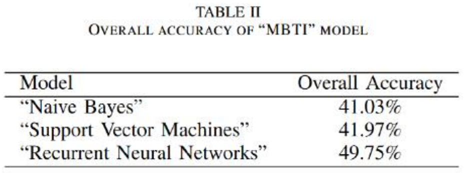


TABLE II
OVERALL ACCURACY OF "MBTI" MODEL

| Model | Overall Accuracy |
|---|---|
| "Naive Bayes" | 41.03% |
| "Support Vector Machines" | 41.97% |
| "Recurrent Neural Networks" | 49.75% |

Fig.10 Accuracy in reference paper: **"Personality Type Based on Myers-Briggs Type Indicator with Text Posting Style by Using Traditional and Deep Learning."**

This is the accuracy they got in the paper "**Personality Type Based on Myers-Briggs Type Indicator with Text Posting Style by Using Traditional and Deep Learning**". As one can see that when they used pairing of traits, they are getting more average accuracy for the models, but they got less overall accuracy. They have less accuracy than our model, which is approximately 70%. The same split was used, 75-25% for the train test respectively that we also used in our model.

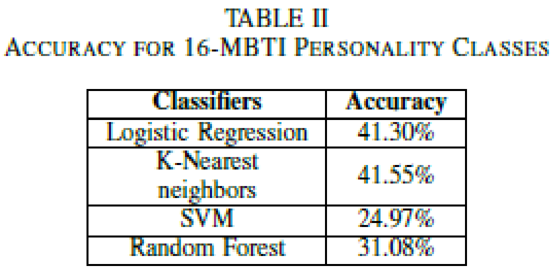
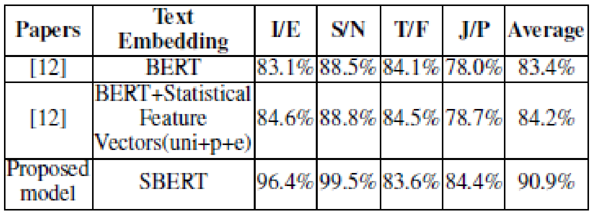


TABLE II
ACCURACY FOR 16-MBTI PERSONALITY CLASSES

| Classifiers | Accuracy |
|---|---|
| Logistic Regression | 41.30% |
| K-Nearest neighbors | 41.55% |
| SVM | 24.97% |
| Random Forest | 31.08% |

Fig .11. Accuracy for Paper: "**Personality Prediction Based on Contextual Feature Embedding SBERT**"



| Papers | Text Embedding | I/E | S/N | T/F | J/P | Average |
|---|---|---|---|---|---|---|
| [12] | BERT | 83.1% | 88.5% | 84.1% | 78.0% | 83.4% |
| [12] | BERT+Statistical Feature Vectors(uni+p+e) | 84.6% | 88.8% | 84.5% | 78.7% | 84.2% |
| Proposed model | SBERT | 96.4% | 99.5% | 83.6% | 84.4% | 90.9% |

Fig. 12. Accuracy Table "2" for paper: "**Personality Prediction Based on Contextual Feature Embedding SBERT.**"

For the SBERT they got more accuracy when they did pair of traits. Overall accuracy is not written by them for SBERT, but it should be near 65%.

## 3. ANALYSIS

### 3.1 What did we do well?

Our approach demonstrated notable strengths in achieving a 70% accuracy rate in predicting MBTI personality types. Unlike many existing studies that calculate accuracy based on the similarity of traits across the four pairs of the eight traits (E-I, S-N, T-F, J-P). For example, suppose the true value of the personality trait is INTJ but it predicted INTP so in the accuracy it will show an increase of accuracy for pairs E-I, S-N, T-F. Our method required an exact match across all four traits for a prediction to be considered a true positive. This stringent criterion underscores the robustness of our model's predictions. Still, we got a good accuracy when we used BERT. Also for the CNN approach, we handled the imbalance data well by using oversampling techniques like random oversampling and SMOTE and got an accuracy of 55%.

### 3.2 What could we have done well?

While our approach yielded promising results, there are areas where we could enhance our methodology. Firstly, addressing the imbalanced dataset more effectively could improve the model's performance. Exploring additional techniques beyond oversampling may offer more nuanced solutions to tackle this challenge. Additionally, incorporating larger language models with increased parameters might further optimize accuracy. Despite attempts to implement such models, we encountered 'resource exhausted' errors on platforms like Kaggle, indicating a need for more efficient resource management or alternative solutions to handle such large models.

### 3.3 What is left for future work?

Moving forward, our future work aims to enhance the model's performance by balancing the dataset with authentic data from minority personality traits, potentially leading to improved accuracy. We also plan to streamline our workflow by developing a comprehensive pipeline that facilitates efficient data preprocessing, model training, and evaluation. Furthermore, we envision developing a user-friendly website where individuals can input sentences, and our model will predict their MBTI personality type, thereby extending the accessibility and applicability of our research to a broader audience.

## 4. CONCLUSION

Despite drastically unbalanced datasets, large language models have transformed natural language processing (NLP) by exhibiting remarkable accuracy in class prediction across sixteen categories. Nevertheless, we ran into issues when we tried to train our model on the Kaggle platform using big BERT parameters. In response, we tested using smaller batch sizes and fine-tuned BERT models from Hugging Face, albeit at the expense of longer training times. Because bootstrapping approaches need a large amount of training time, even for a dataset with only 6000 rows, we decided against using them. Similarly, oversampling would have increased the number of rows in our data to 25000 without appreciably increasing accuracy Going on, we want to improve dataset balance by adding real Twitter data with multiple personalities represented in it. We also intend to investigate the use of several Large Language Models (LLMs) with more parameters in order to possibly improve accuracy. Unfortunately, collecting data is difficult since some personality types are few.