Machine Learning Approach to Personality Type Prediction From Social Media Posts Using Sentence Transformers

Project & Thesis-I

CSE 4100

A thesis Report

Submitted in partial fulfillment of the requirements for the Degree of

Bachelor of Science in Computer Science and Engineering

Submitted by

Risha Asfara	**180204001**
Rasel Ahmed	**190104093**
Tahira Ferdousi	**190104123**
Md. Ali Akber	**190104129**

Supervised by

Ms.Raqeebir Rab

Department of Computer Science and Engineering

Ahsanullah University of Science and Technology

Dhaka, Bangladesh

May 16,2023

ABSTRACT

Personality assessments are systematic evaluations designed to measure an individual's behavior and personality. The MBTI is a popular personality assessment tool, with millions of test takers worldwide. The exam is usually done online or on paper, and involve multiple-choice questions to gauge preferences in four areas: extraversion and introversion, sensing and intuition, thinking and emotion, and judging and perceiving. Machine learning models are now used to predict one's personality traits. The goal of this study is to automate the process of using machine learning models to predict a person's personality traits from social media postings by identifying the context of their posts with the aid of sentence transformers. Various text preprocessing techniques, data balancing techniques like random oversampling, and class weight balancing are used in this project. The context of the text posts is determined using Sentence-BERT (SBERT), a pre-trained model created especially for sentence-level embeddings. Finally, using a variety of machine learning methods, such as "K Nearest Neighbours," "Support Vector Machines," and "Logistic Regression," it is possible to predict people's personalities from text using the Myers-Briggs Type Indicator (MBTI).

Index Terms: Sentence-BERT (SBERT), K Nearest Neighbours, Support Vector Machines, Logistic Regression, Myers-Briggs Type Indicator (MBTI),Random Oversampling, Class Weight Balancing

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Social media platforms have developed into a veritable gold mine of user-generated content, providing insights into people's thoughts, emotions, and personality traits. A widely used method of classifying people into different personality types is the MBTI (Myers-Briggs Type Indicator). Combining these two elements, the development of an MBTI personality trait text classification machine learning model specifically trained on social media posts holds immense potential.

1.1 Motivation

A person's personality can be defined as the role they perform or the persona they present to the outside world. Personality assessments can help someone grow both personally and professionally. The applications of an MBTI personality trait text classification machine learning model to social media posts are extensive. The motivation behind developing an MBTI personality trait text classification machine learning model from social media posts stems from several factors. First, social media has become a prominent platform for self-expression and communication, enabling individuals to share their thoughts, experiences, and opinions freely. By tapping into this vast amount of user-generated textual data, we can gain deep insights into individuals' personality traits at scale. Then it can also aid in identifying potential correlations between specific personality types and mental health conditions or communication styles.

1.2 What is Machine Learning?

Machine learning is a way to teach computers how to learn and make decisions on their own by analyzing data, without being explicitly programmed for every specific task.Machine learning algorithms are designed to generalize from data, which means they can apply their learning to new, unseen data and make predictions or decisions based on that data. There are several types of machine learning algorithms, including:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

In supervised learning, the algorithm is trained on labeled data, where the input data is paired with corresponding output labels or targets. The algorithm learns to map the input data to the correct output based on the provided labels. This type of learning is commonly used for tasks such as classification.

1.3 What is Text Classification?

Text classification is an analytical procedure that takes any text content as input and assigns a label (or classification) from a specified set of class labels. Assigning the best class label automatically to a given text document based on its content is the aim of text categorization. The following phases are often included in the text classification process:

1. Data Preparation
2. Feature Extraction
3. Training Data Preparation
4. Model Selection and Training
5. Model Evaluation
6. Model Deployment

1.4 Problem Statement

We are interested in seeing if there is a big correlation between a person's online language use and their true personality in a world where communication is becoming more and more social media-centered. Every day, people share their ideas and emotions on social media. The posts' context varies depending on the personality of the author. Also, the personality of a person can be predicted from the context of his or her posts. Our aim is to use state-of-the-art model like SBERT is implemented to capture the context of individuals' posts and predict their personalities.

1.5 Proposed Methodology

To predict a person's MBTI personality trait, our first step would be to gather social media posts from individuals. We collected the MBTI dataset, which is freely accessible on Kaggle. Text data is unsuitable with machine learning models. Text data must be transformed into numerical vectors. Prior to that, the data needs to be cleaned up as it can contain URLs, images, and videos that are not very useful for determining someone's personality.To understand the text posts' context For each person's post, a state-of-the-art model like SBERT is used, which provides contextualized sentence embeddings.It leads to multiple machine learning models, such as SVM and LR.

Chapter 2

Background Study

2.1 What is MBTI (Myers-Briggs Type Indicator)?

Today, a variety of personality models, including the ”Big Five Personality Traits” model [1] and the ”Myers-Briggs Type Indicator (MBTI)” model [2], are used to describe personalities. Since its initial release in 1962, it has been found that ”MBTI” is more effective because it can be used in a wider range of disciplines. So, we selected the ”MBTI” personality model for this study due to its widespread use and the potential for it in a number of industries. The MBTI's fundamental principle is that each person's personality can be classified into only one of its 16 types. These classifications are based on four personality traits, each of which consists of two opposing preferences. The four dimensions are [3] :

1. Introversion (I) against extrovert (E): This component indicates the person's perceptual orientation. Extroverts are reputed to respond to direct and concrete environmental conditions. However, introverts focus inward on their internal and personal responses to their surroundings.

2. Intuition (N) against sensing (S): People who prefer sensing rely on that which can be perceived and are thought to be inclined toward reality. People who prefer intuition tend to rely more on their unconscious and nonobjective perceptual processes.

3. Feeling (F) against thinking (T): A prediction for thinking denotes the application of reason and logic to draw conclusions and make decisions. A preference for making judgments based on subjective considerations, including emotional responses to events, is represented by feeling.

4. Perception (P) against judgment (J): Briggs and Myers developed the judgment-perception 2 preference to show whether rational or illogical judgments predominate when a person interacts with the environment. The perceptive person uses the perceiving and

intuition processes, whereas the judging person uses a combination of thoughts and feelings when making decisions.

2.2 Classical Algorithms for Text Classification

- **Logistic Regression**: Based on the idea of probability, it is a predictive analysis method. When the dependent variable (target) is categorical, logistic regression is utilized [4].

- **Support Vector Machine:** Support Vector Machine, sometimes known as SVM, is a linear model used to solve classification and regression issues. The SVM concept is simple: A line or a hyperplane that divides the data into classes is produced by the algorithm. The SVM method identifies the points from both classes that are closest to the line. These points are called support vectors. The distance between the line and the support vectors is now calculated. The margin refers to this separation. Maximizing the margin is the goal. The best hyperplane is the one for which the gap is greatest. SVM aims to create decision boundaries that allow for the greatest feasible separation between the two groups [5].

- **Decision Tree:** A decision tree is a supervised learning algorithm that builds a tree-like model of decisions and their possible consequences. It recursively splits the input data based on the features to create a tree structure, where each internal node represents a decision based on a specific feature, and each leaf node represents a class label or outcome. Decision trees are intuitive, interpretable, and can handle both categorical and numerical features [6].

- **Random Forest:** Random forest is an ensemble learning method that utilizes multiple decision trees to make predictions. It creates an ensemble of decision trees, where each tree is trained on a random subset of the training data and features. During prediction, each tree in the random forest independently predicts the class label, and the final prediction is determined by majority voting or averaging the individual predictions. Random forests are known for their ability to handle high-dimensional data, reduce overfitting, and provide robust predictions [7].

2.3 Handling Imbalanced Data

Imbalanced datasets can pose challenges in text classification and may lead to biased or inaccurate models.The two most popular data balancing techniques available are oversampling and undersampling. Undersampling involves discarding a significant portion of the majority class instances, which can result in the loss of valuable information. By reducing the training data for the majority class, the classifier may not capture the full range of patterns and variations in that class, leading to a less accurate model.On the other hand Random Oversampling includes selecting random examples from the minority class with replacement and supplementing the training data with multiple copies of this instance, hence it is possible that a single instance may be selected multiple times [8].

2.4 Tokenization

Tokenization involves identifying and dividing the text into pertinent chunks. The fundamental building blocks for further NLP tasks like text classification are provided by these units. Tokenization aims to convert unstructured text input into a structured format for computational analysis [9].

2.5 Lemmatization

A lemma is the base or canonical form of a word, and lemmatization is a linguistic and natural language processing (NLP) technique that attempts to change words into lemma form. Lemmatization reduces inflected or derived words to a standard form to improve text data analysis and comprehension [10].

2.6 Hyperparameters

Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. The prefix 'hyper' suggests that they are 'top-level' parameters that control the learning process and the model parameters that result from it. Support Vector Machine (SVM) have a number of hyperparameters that can be adjusted to enhance the SVM model's functionality. Here are some typical SVM hyperparameters [11]:

- **Kernel :** kernel function used to transform the input data into a higher-dimensional feature space. The kernel function plays a crucial role in SVMs as it defines the similarity measure between data points and enables the creation of non-linear decision boundaries. The kernel function can take other values such as linear, poly, rbf or sigmoid,

- **Regularization Parameter :**

  Regularization is a technique used to prevent overfitting, which occurs when a model becomes too complex and starts to fit the training data too closely, resulting in poor generalization to unseen data.
  Higher 'C' value: The SVM will try to classify all training examples correctly, potentially leading to a more complex decision boundary. This can be useful when the training data is noisy or contains overlapping classes.
  Lower 'C' value: The SVM will aim for a larger margin and tolerate more misclassified examples. It promotes a simpler decision boundary and helps to avoid overfitting. This can be effective when the training data is clean and well-separated.
  Choosing the right 'C' value: The optimal 'C' value depends on the specific dataset and problem at hand. It is typically determined through hyperparameter optimization techniques like grid search or Bayesian optimization. Cross-validation is often used to assess the model's performance for different 'C' values and select the one that generalizes the best.

- **Gamma (Kernel Coefficient):** The 'gamma' parameter in Support Vector Machine (SVM) models determines the influence of each training example on the decision boundary. It is a hyperparameter that affects the flexibility and sensitivity of the SVM model to the training data.Gamma can take the value of scale, auto, or a float value.

- **Degree (Polynomial Kernel):** If the SVM uses a polynomial kernel, the degree hyperparameter determines the degree of the polynomial function. Higher degree values can capture more complex relationships in the data but also increase the model's complexity.

2.7 Hyperparameter-tuning with Bayesian Optimization :

Hyperparameter-tuning is the process of searching the most accurate hyperparameters for a dataset with a Machine Learning algorithm. Bayesian optimization's main goal is to simulate the unknown objective function that correlates hyperparameter adjustments to related performance values. A probabilistic model (typically a Gaussian Process) is updated to represent the connection between hyperparameters and performance by iteratively choosing hyperparameter configurations to evaluate based on previous observations [12].

Chapter 3

Literature Review

In their project, Ontoum, Sakdipat, and Jonathan H. Chan [13] utilized the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology to guide their learning process. To expedite development and minimize the development cycle, they combined CRISP-DM with an agile methodology, which is known for its rapid and iterative approach to software development.To enhance the accuracy of their classification task, the researchers decided to implement four binary classifications instead of a more complex 16-class multi-class classification. This approach aimed to simplify the task and improve the precision of their predictions.The initial step in their data preprocessing phase involved cleaning the text data. They utilized Python's NLTK (Natural Language Toolkit) to remove stop words, which are commonly used but carry little semantic value. This step helped reduce noise in the text data, improving the quality of the subsequent analysis.Following the cleaning step, the researchers performed lemmatization to normalize the words in the text. Lemmatization is a technique that transforms words into their base or dictionary form, reducing different variations of words to a common representation. This process helps to ensure consistency and improve the accuracy of subsequent analyses.For the Naive Bayes and Support Vector Machine Classifier models, tokenization was applied after lemmatization. The transformed words were tokenized using NLTK's word tokenizer, breaking the text into individual tokens or words. The text was then converted into a frequency representation using techniques such as Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). These techniques aimed to capture the importance and presence of words in the text, enabling the models to learn from these patterns.In the case of Recurrent Neural Networks (RNN), a Keras word tokenizer was used for tokenization after lemmatization. This allowed the RNN model to process the text in a sequential manner, considering the order and context of the words. Moving to the modeling phase, the researchers trained Naive Bayes and Support Vector Machines classifiers on the preprocessed data. These classifiers achieved overall accuracies of 41.03% and 41.97% respectively, indicating their effectiveness in predicting

the desired outcomes.To further improve the performance of the Recurrent Neural Networks model, the researchers introduced a CONV1D layer, which enhanced the model's ability to capture local patterns in the text. Additionally, they added a Bi-directional Long Short-Term Memory (BI-LSTM) layer, which allowed the model to capture long-term dependencies and context. This modified RNN model achieved an overall accuracy of 49.75%, indicating its improved performance compared to the other models.However, despite the good accuracy achieved by the four binary classifications in each model, the researchers noted a general weakness in accurately classifying all four dimensions of the MBTI (Myers-Briggs Type Indicator). This suggests that there is still room for improvement in the models' ability to accurately classify all aspects of personality traits, highlighting a potential area for future research and development.

The major goal of the experiment [14] conducted by Cui, Brandon, and Calvin Qi was to identify the machine learning and deep learning models that are most successful in correctly predicting a person's personality from their social media posts. The feature selection process involved featurizing the posts using a "bag of words" approach. The size of the bag, denoted as B, was tuned as a hyperparameter, and it was found that the most effective value for B was 50,000. Additional features such as bigrams, skip grams, part of speech tags, and capital letter count were also considered in the featurization process.The initial model used was a multiclass Softmax classifier, aiming to predict all 16 personality types. The baseline performance of this model showed a training accuracy of 19% and a test accuracy of 17%, which outperformed randomly choosing classes by 6.25%. However, in order to improve accuracy, the approach was modified to use four binary classifiers instead of a 16-class multi-class classifier.Naive Bayes was employed to predict the binary classes of (Extrovert/Introvert, Sensing/Intuition, Thinking/Feeling, Judging/Perceiving), achieving accuracies of 75%, 84.5%, 62.4%, and 60.3%, respectively. The overall accuracy of the Naive Bayes model improved to 26%, a significant improvement compared to the Softmax baseline of 17%.Support Vector Machines (SVM) were also utilized, using the hinge loss and aiming to maximize margin through L2 regularization. The hyperparameters were carefully tuned, and minibatch Stochastic Gradient Descent was employed until convergence. An ablative error analysis revealed that the preprocessing step had the largest impact on classifier accuracy. Therefore, focusing on improving the preprocessing steps was identified as a potential avenue for achieving better results. Deep learning techniques were explored, employing a multi-layer long-short-term memory (LSTM) recurrent neural network as the encoder and a 3-layer feed-forward neural network with rectified linear unit (ReLU) activations as the decoder. The decoder provided the probability of each class through a softmax function. The 16-class classifiers achieved an accuracy of 23%, while the four binary classifiers showed an accuracy of 38%.The paper also discussed a comparison of various deep learning parameters, examining their impact on the development and test accuracy of

the models. Furthermore, it suggested potential future work, such as exploring alternative mechanisms for representing word embeddings, including char and k-char word embeddings, and utilizing pretrained GloVe vectors.Overall, the study aimed to identify the most effective machine learning and deep learning models for accurately predicting personality traits from social media posts, and it highlighted the importance of feature selection, preprocessing steps, and parameter optimization in achieving better classification performance. The future work suggestions indicate the potential for further improvements and advancements in the field of personality trait prediction using text classification models.

In their research [15], Pradhan, Tejas, et al. utilized a labeled dataset from a social network to develop a machine learning model for predicting personality types. The ultimate goal was to deploy this model on a website as a personality predictor. The methodology employed in this study consisted of four main steps: text mining, machine learning model development for classification, deep learning model development for classification, and the creation of a website for personality prediction.To preprocess the dataset, several rounds of data cleaning were conducted. This involved removing unnecessary punctuation, emoticons, and stop words using regular expressions in Python and the NLTK text processing library. In the feature selection stage for machine learning, it was essential to assign weights to words based on their frequency of use to classify personalities accurately. The count vectorizer and TF-IDF techniques were employed to convert the unstructured text into vectors, providing a structured representation of the data. The performance of the machine learning models yielded approximately 60% accuracy. However, the Naive Bayes classifier demonstrated only 32% accuracy due to high bias and the challenge of multi-class classification. To address this, a random forest classifier was trained, achieving an accuracy of 36.03%. Additionally, the linear support vector machine algorithm achieved an accuracy of 57.9%. For the deep learning approach, feature selection played a crucial role. An embedding layer was used to transform each word into a 1-dimensional vector, which was then fed into the CNN layer as input. Weights and biases were assigned to these input vectors based on features and max-pooling techniques. This approach achieved around 81.4% accuracy in multi-label classification. Instead of predicting 16 classes for the 16 personality types, four binary classifiers were utilized to enhance precision, recall, and accuracy.The study acknowledged certain limitations. The model exhibited bias since some personality types were more common than others in the dataset. To mitigate this, the researchers proposed scraping more information about the minority personality types to address the imbalance. Additionally, they suggested exploring the use of an RNN model to improve the overall accuracy of the predictions.In summary, Pradhan, Tejas, et al. implemented a comprehensive methodology involving text mining, machine learning, and deep learning techniques to develop a personality prediction model. Their results demonstrated significant progress in accurately predicting personality types, with potential areas for further improvement and

future research.

In their study [16], Bharadwaj, Srilakshmi, et al. utilized a dataset consisting of 8,660 individuals' Twitter posts, with each individual contributing a total of 49 posts. The researchers focused on preprocessing the data to ensure its quality and relevance for further analysis.During the data preprocessing stage, the tweets were carefully cleaned to remove any links, numbers, or punctuation marks. Additionally, lemmatization and stemming techniques were applied using WordNet Lemmatizer and Lancaster Stemmer, respectively, to standardize the words in the sentences. Tweet Tokenizer was employed to tokenize the tweets, breaking them down into individual words. The top 1,500 most frequent words in the tweets were selected to represent the overall content.To generate the feature vector, multiple sources of information were combined. TF-IDF (Term Frequency-Inverse Document Frequency), EmoSenticNet, LIWC (Linguistic Inquiry and Word Count), and ConceptNet features were integrated into a comprehensive representation of the tweets. EmoSenticNet, a sentiment analysis model, was used to analyze the emotional tone of the text. LIWC, on the other hand, focused on analyzing the linguistic content and identifying language patterns associated with psychological and social processes. ConceptNet, an open-source semantic network, provided information about word meanings and relationships. The resulting feature vector had a size of 1,850, incorporating various aspects of the tweets. To reduce the dimensionality of the feature space and improve computational efficiency, the researchers employed Singular Value Decomposition (SVD) and reduced the total number of features to 50.In the training phase, the four MBTI (Myers-Briggs Type Indicator) classes were individually treated by the training module. Each class comprised two traits, and binary values of 0 or 1 were assigned to represent the presence or absence of these traits. Based on the feature vectors, Support Vector Machines (SVM) and Naive Bayes classifiers were trained. Additionally, a neural network with softmax output was trained using the data.To evaluate the performance of the models, three metrics, including accuracy, were employed. Across all MBTI classes, SVM consistently outperformed other classifiers. Specifically, it was observed that SVM yielded the best results when using the LIWC and TF-IDF feature vector combination without feature reduction for the I/E (Introversion/Extraversion) and S/N (Sensing/Intuition) traits. Moreover, the performance of SVM was found to improve when incorporating ConceptNet and EmoSenticNet features and reducing the number of features, indicating the importance of these additional factors in enhancing the classification accuracy.Through their comprehensive approach, Bharadwaj, Srilakshmi, et al. successfully leveraged various feature selection techniques and machine learning models to analyze and classify individuals' Twitter posts based on their MBTI traits. The results highlighted the effectiveness of SVM in this context and emphasized the significance of incorporating diverse features for improved performance.

In their study [17], Brindha, S., K. Prabha, and Sandeep Sukumaran aimed to evaluate

the classification accuracy of various algorithms for distinct datasets. The researchers examined the performance of K Nearest Neighbor (KNN), Naive Bayes, Support Vector Machine (SVM), Decision Tree, and Logistic Regression algorithms.K Nearest Neighbor is a widely used pattern recognition algorithm known for its effectiveness. However, it has some limitations. One challenge is the calculation complexity associated with using all training data. Finding similarities between samples takes more time when there are fewer similarities present. Additionally, KNN requires recalculating if there is even a small change in the training data since it solely relies on that data. Another limitation is the lack of weight difference between samples, which may affect classification accuracy.Naive Bayes is a quick and scalable algorithm commonly used for reference scoring and model building. However, it faces an issue when there is a certain attribute value that has no instances of a class label, potentially impacting its performance. Support Vector Machine (SVM) transforms training data into a top-dimensional space using a nonlinear mapping. Although SVMs can train slowly, they excel at forming composite nonlinear decision boundaries. Decision Tree algorithms have shown promise in solving land cover mapping issues. They are easy to interpret due to the transparent decision tree structure. However, overfitting can be a drawback, especially when dealing with a large number of dictionary entries. Despite this, decision trees generally perform well in categorization tasks.Logistic Regression models are typically used to determine the probability of class membership for one of the two categories in a dataset. They are commonly employed in classification tasks. The researchers utilized datasets DS1 and DS2, varying in sample sizes from 200 to 5000 records, to evaluate the performance of KNN, Naive Bayes, SVM, Decision Tree, and Logistic Regression algorithms. For DS1, the error rate of KNN, SVM, and Decision Tree algorithms increased as the sample size increased. In contrast, the error rate decreased for Naive Bayes and Logistic Regression algorithms with increasing sample sizes. In the case of DS2, the error rate increased as the sample size increased, although the rate of increase was less noticeable for Naive Bayes and Logistic Regression compared to other methods. Hence, it can be concluded that a larger sample size does not necessarily guarantee a lower error rate.Regarding the evaluation metrics, SVM outperformed all other algorithms in terms of precision, recall, and F1 measure. The results indicated that SVM yielded the highest classification performance among the tested algorithms.In summary, Brindha, S., K. Prabha, and Sandeep Sukumaran's study provided valuable insights into the classification accuracy of different algorithms for distinct datasets. The findings shed light on the strengths and limitations of each algorithm and highlighted SVM's superior performance in terms of precision, recall, and F1 measure.

Chapter 4

Methodology

4.1 Overview

The overall methodology of our proposed task is shown in fig. 4.1

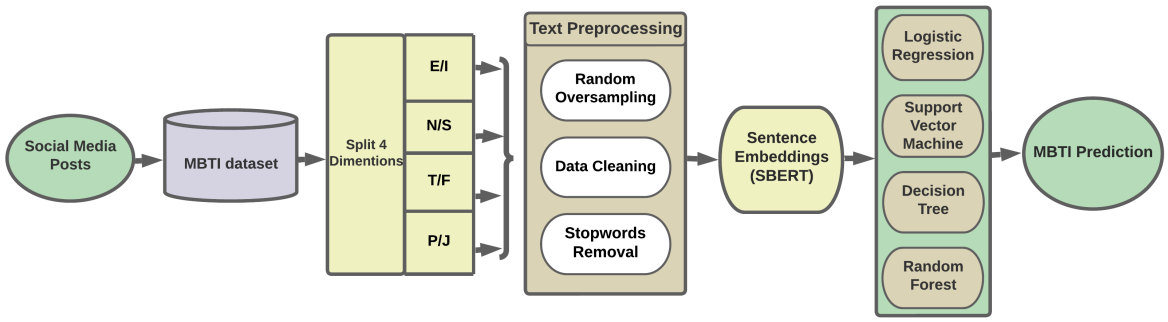

Figure 4.1: Proposed Methodology for MBTI Personality Classification

1. **Data Collection :** The publicly available Myers-Briggs Personality Type Dataset is acquired from Kaggle [18].

2. **Data Augmentation :** Divide the dataset into four dimensions in order to execute four independent binary classifications, which together make up a single MBTI personality type.

3. **Random Oversampling :** We can notice a significant data imbalance in each dimension after partitioning the dataset into the four MBTI type dimensions. Therefore, random over sampling is used to address the problem.

4. **Data Preprocessing :** This part does data cleansing and stop word removal.

5. **Feature extraction :** To extract the contextualized feature from the social media posts, a trained Sentence-BERT model is used.

6. **Classification :** We used four different classifiers, including Logistic Regression, Support Vector Machine, Decision Tree, and Random Forest, to carry out the four separate dimensions' classification task that we derived from the MBTI types.

7. **Bayesian Optimization :** To enhance performance, the Support Vector Machine's hyperparameters are tuned via Bayesian optimization.

8. **Performance Evaluation :** The outcome of each model is computed using a separate performance metric and plotted for comparison.

4.2 Data Collection and Visualization:

Along with the user's MBTI personality type, the dataset consists of last 50 things each user has posted (Each entry separated by "|||" (3 pipe characters)) for a total of 8675 users. The labels are obtained by allowing the user to provide their MBTI type as account information. The messages are taken from the PersonalityCafe online forum, a venue for all kinds of talks and discussions. The visual representation is shown in fig. 4.2

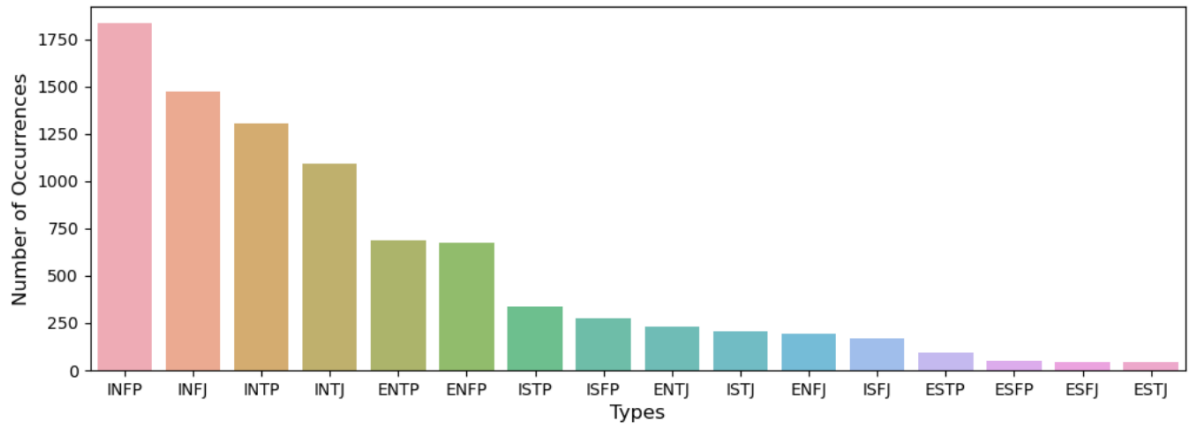

Figure 4.2: Visual Representation of all 16 MBTI types in the dataset

4.3 Data Augmentation:

According to [15], with 16 classes, the accuracy could not be improved because of the large number of classes to predict. Instead of predicting 16 classes for 16 personality types, four binary classifiers were used in order to get better precision, recall, and accuracy. So we added four additional columns (is_introvert,is_sensing, is_thinking, and is_judging) to the

dataset that were labeled according to how respondents responded to the ”MBTI” model’s four dimensions in order to perform multilabel classification. The process is intended to increase accuracy [13]. When is_introvert is set to 1, it indicates that the respondent is an introvert, while when it is set to 0, it indicates that they are an extrovert. Similar rules apply to the other three columns that identify which dimension of the ”MBTI” model a respondent belongs to.

4.4 Random Oversampling :

The dataset is drastically skewed, with an introvert count of 6676 versus an extrovert count of 1999. Also, the following numbers contrast intuition (7478) with sensing (1197), feeling (4694) with thinking (3981), and perception (5241) with judgment (3434). The visual representation is shown in fig. 4.3. Classification algorithms may learn to predict the majority

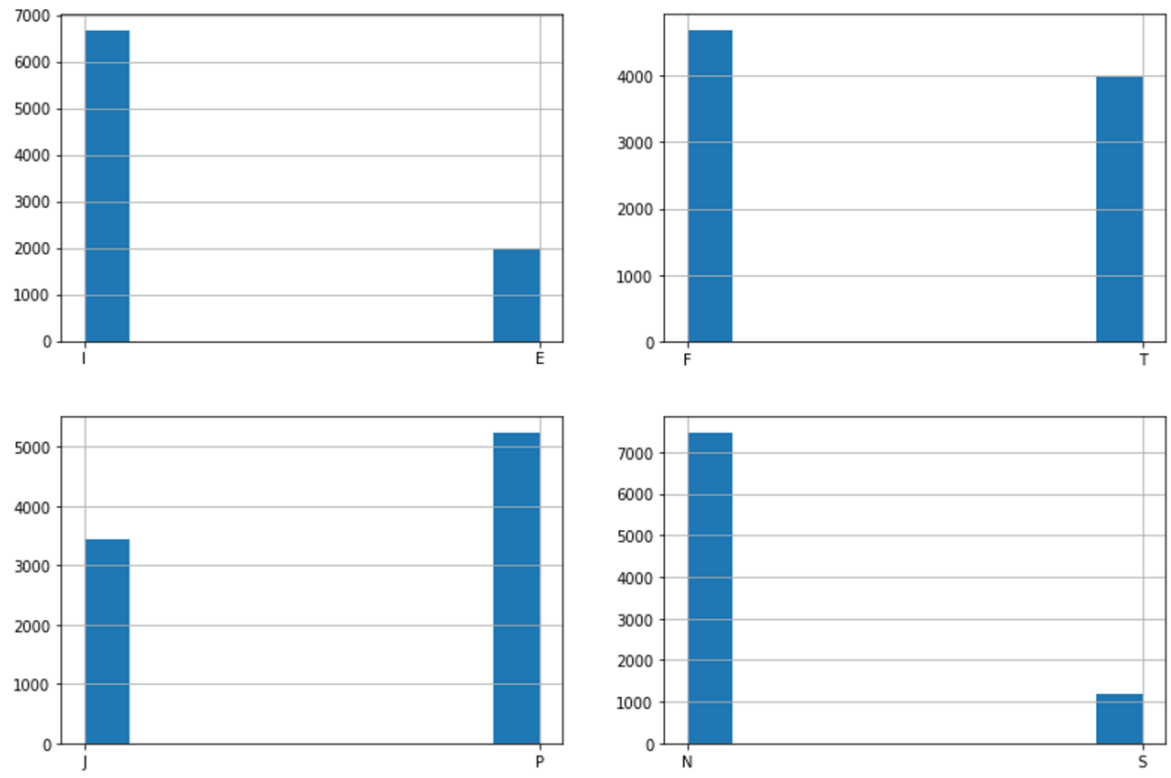

Figure 4.3: Diagram showing the proportions of each "MBTI" dimension

class more correctly since it predominates the training set due to an unbalanced dataset. This causes the minority class to do poorly. Undersampling, oversampling, and balancing class weights are just a few of the strategies used to remedy the imbalance between classes. The random oversampling strategies worked well in our case, despite the fact that we also experimented with the class weights balancing scheme and undersampling. Random oversampling is the easiest type of oversampling because it essentially duplicates minority cases

to increase the imbalance proportion. The performance of the machine learning classifier for effective predictions was significantly enhanced by this duplication of minority class enhancement [17]. After random over-sampling:

- Introversion(I) : 6676 / Extraversion(E) : 6676.
- Sensing(S) : 7478 / Intuition(N) : 7478.
- Thinking(T) : 4694 / Feeling(F) : 4694.
- Judging(J) : 5241 / Perceiving(P) : 5241.

4.5 Preprocessing :

1. **Data Cleaning :** Numerous URLs and different punctuation marks were present in the data, which needed to be cleaned up and eliminated. The unnecessary punctuation, pictures, and emojis were all deleted over several rounds of the data cleaning process. Since these don't carry much significant value that can capture a person's personality.
2. **Tokenization and Stop Words removal :** Preprocessing activities like tokenization is performed internally by SBERT model.Tokenizing is the process of breaking a text down into smaller units, called tokens. Tokens are also referred to as individual words or significant textual elements. But stopwords removal have all been carried out using predefined lists of stop words from the Spacy library. Each token in the sentences are lowercased after stopwords removal.

4.6 Feature Extraction :

The process of feature engineering plays a crucial role in machine learning by selecting and extracting relevant features from raw data. In the domain of Natural Language Processing (NLP), feature engineering involves transforming unstructured text data into a numerical representation that can be effectively utilized by machine learning algorithms.

1. **Sentence Embedding:** In NLP, BERT (Bidirectional Encoder Representations from Transformers) is a widely used model for translating sentences into vector representations. However, the vector space produced by BERT is not optimized for cosine similarity or other popular similarity measures. To overcome this limitation, Sentence-BERT (SBERT) was introduced. SBERT provides a computationally efficient solution and outperforms BERT in terms of clustering large sets of sentences. For example,

hierarchical clustering of 10,000 sentences with BERT takes approximately 65 hours, while SBERT reduces the effort to merely 5 seconds [4].

2. **SBERT Model ALL-MPNET-BASE-V2:** SBERT modifies the BERT model, initially designed for generating contextual word embeddings, to produce numerical representations of text data. Instead of word embeddings, SBERT leverages a model known as "All-MPNetBaseV2" to generate sentence embeddings. This approach focuses on contextualized embeddings, wherein the embedding for a given word or sentence considers not only the specific word or sentence but also its surrounding context.

3. **Tokenization and Embedding Construction:** To create sentence embeddings using the "ALL-MPNET-BASE-V2" model, the input sentence undergoes tokenization using the Byte Pair Encoding (BPE) algorithm. This process converts the sentence into a series of subwords. The subword tokens are then processed through multiple layers of the transformer architecture, which results in contextualized embeddings for each subword.

4. **Integration of Subword Embeddings:** To obtain a fixed-length representation of the input sentence, the contextualized embeddings for all subwords are integrated. This can be achieved by taking the maximum pooling across all subwords embeddings. The integrated representation is a 768-dimensional vector for the ALLMPNET-Base-V2 model.

4.7 Classification:

1. **Logistic Regression :**
   - Built-in module used : sklearn.linear_model.LogisticRegression
   - Parameters : Default (penalty='l2', solver='lbfgs', C=1.0, etc.)
2. **K-nearest Neighbor :**
   - Built-in module used: sklearn.neighbors.KNeighborsClassifier
   - Parameters: Default (n_neighbors=5, weights='uniform', algorithm='auto', etc.)
3. **Decision Tree :**
   - Built-in module used: sklearn.tree.DecisionTreeClassifier
   - Parameters: Default (criterion='gini', splitter='best', max_depth=None,min_samples_split=2, etc.)
4. **Support Vector Machine :**

- Built-in module used: sklearn.svm.SVC
- Parameters: Default (C=1.0, kernel='rbf', degree=3, gamma='scale', etc.)
- Bayesian optimization : Hyperparameter tuning with cross validation is performed to get the best hyperparameter combination. Keeping the accuracy as the evaluation metric best combination of the hyperparameters are obtained and are shown in Table 4.1.

Binary Classification	**Regularization**	**Kernel**	**Degree for Polynomial Kernel**	**Gamma Value**
I/E	100	poly	4	scale
S/N	22.09	poly	4	scale
T/F	18.04	rbf	3	scale
J/P	18.67	poly	4	scale

Table 4.1: Best Hyperparameter Combination for SVM.

Chapter 5

Performance Analysis

5.1 Evaluation Metrics

1. **Confusion Matrix -** Confusion matrices are frequently used in binary classification issues to map the expected class against the true class and produce true positives, true negatives, false positives, and false negatives [19].

   (a) True Positive (TP): This represents the cases where the model predicted a positive class correctly. In other words, the model predicted a positive outcome, and the actual outcome was also positive.

   (b) True Negative (TN): This represents the cases where the model predicted a negative class correctly. It means the model predicted a negative outcome, and the actual outcome was also negative.

   (c) False Positive (FP): This represents the cases where the model predicted a positive class incorrectly. The model predicted a positive outcome, but the actual outcome was negative.

   (d) False Negative (FN): This represents the cases where the model predicted a negative class incorrectly. The model predicted a negative outcome, but the actual outcome was positive.

2. **Precision -** Precision is the ratio between correctly predicted positive classifications to the total predicted positive classifications.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{5.1}$$

3. **Recall -** Recall is the ratio between correctly predicted positive classifications to all positive and negative in the category.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{5.2}$$

4. **Accuracy -** Accuracy measures the performance of a classifier by finding the ratio of correct classifications against all classifications.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Positive} + \text{False Negative} + \text{True Negative}} \tag{5.3}$$

5. **F1 Score -** The F1 score provides a balanced assessment of a model's performance by combining precision and recall into one metric.

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

5.2 Results

We used four distinct binary classifiers on four MBTI type dimensions. In each of the four scenarios, Decision Tree and SVM perform better in terms of accuracy, precision, and other metrics than Logistic Regression and KNN. According to [17] SVM performs better on text classification and Hyperparameter tuning can improve performance in the classification task. So, we used bayesian optimization with cross validation for hyperparameter tuning and it clearly improved the accuracy, precision, recall and F1 schore shown in table 5.1 , 5.1 , 5.2 and 5.4

Model Name	Accuracy	Precision	Recall	F1 Score
Logistic Regression	72.5%	73.7%	70.1%	71.9%
KNN	72.1%	68.6%	81.5%	74.5%
Decision Tree	86.4%	80.6%	95.9%	87.6%
SVM	79.6%	78.9%	80.7%	79.8%
SVM (Hyper Parameter Tune)	92.06%	88.65%	96.48%	92.40%

Table 5.1: Extroverts/Introverts

Model Name	Accuracy	Precision	Recall	F1 Score
Logistic Regression	75.26%	74.88%	76.11%	75.47%
KNN	80.38%	73.83%	94.15%	82.76%
Decision Tree	91.50%	85.92%	99.51%	92.31%
SVM	85.20%	83.18%	88.25%	85.64%
SVM (Hyper Parameter Tune)	96.79%	94.47%	99.39%	96.87%

Table 5.2: Sensors/Intuitives

Model Name	Accuracy	Precision	Recall	F1 Score
Logistic Regression	77.02%	76.47%	78.12%	77.27%
KNN	70.85%	70.08%	72.79%	71.41%
Decision Tree	75.69%	72.80%	80.67%	76.03%
SVM	79.10%	77.50%	82.04%	79.70%
SVM (Hyper Parameter Tune)	83.81%	82.23%	86.26%	84.19%

Table 5.3: Thinkers/Feelers

Model Name	Accuracy	Precision	Recall	F1 Score
Logistic Regression	69.3%	68.9%	70.3%	69.6%
KNN	63.8%	62.3%	69.6%	65.8%
Decision Tree	74.5%	70.9%	82.8%	76.5%
SVM	73%	72.1%	74.9%	73.5%
SVM (Hyper Parameter Tune)	81.54%	80.09%	83.98%	81.99%

Table 5.4: Judgers/Perceivers

The Confusion Matrix for SVM results are displayed in figure 5.1. We can see that the True positive and True negative values are high for each MBTI type dimension, which is desirable for a good classification model.

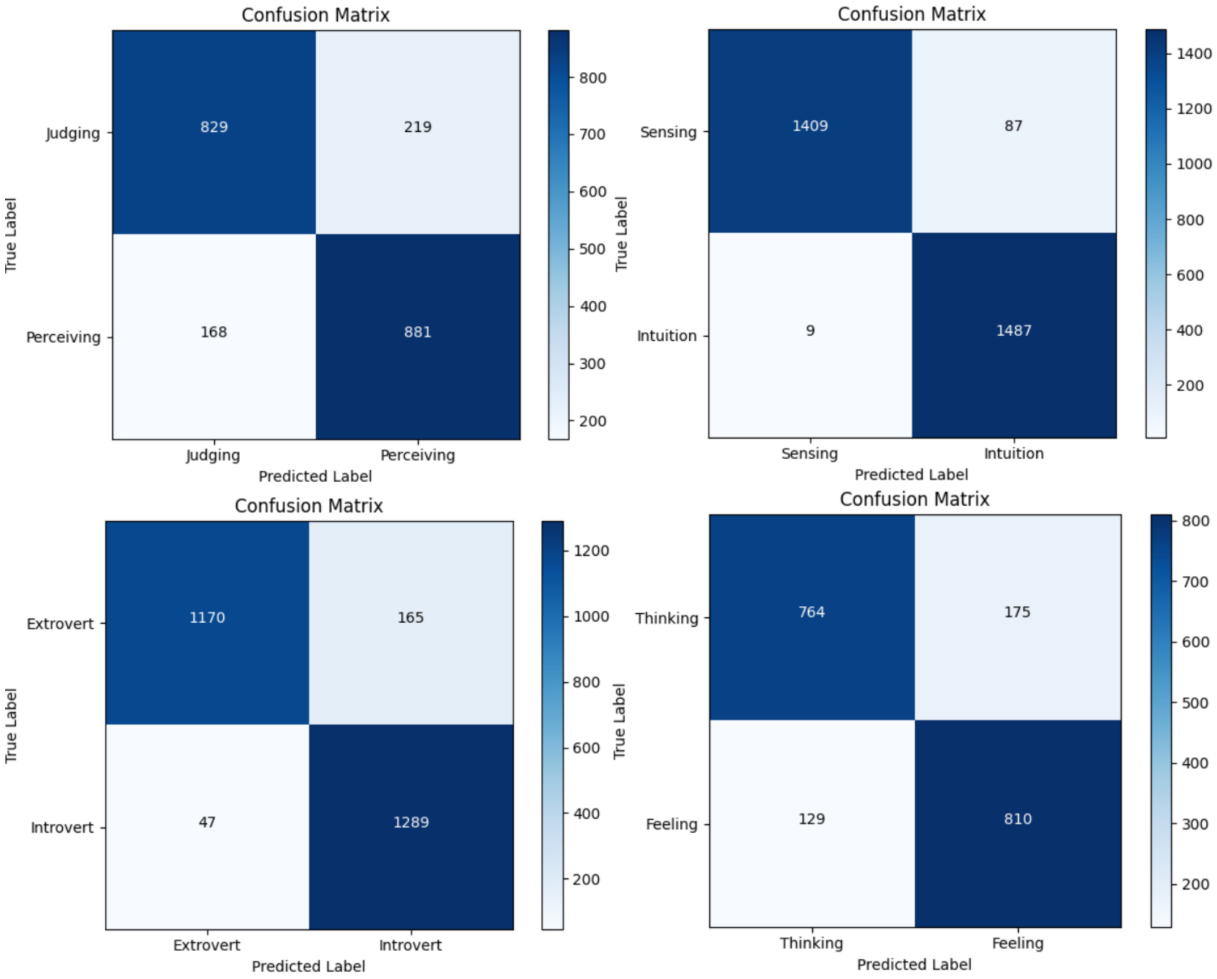

Figure 5.1: Confusion Matrix for SVM

5.3 Comparison

The experimental in table 5.5 shows that our suggested model outperforms the model [20] that uses BERT for text embeddings.Additionally, it outperforms models that uses Bag of Word (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) in [15] and count vectorizer and TF-IDF in [14].

Papers	**Text Embedding**	**I/E**	**S/N**	**T/F**	**J/P**	**Average**
[20]	BERT	83.1%	88.5%	84.1%	78.0%	83.4%
[20]	BERT + Statistical Feature Vectors(uni+p+e)	84.6%	88.8%	84.5%	78.7%	84.2%
[14]	TF-IDF + Count Vectorizer	89.5%	89.8%	69.0%	67.6%	78.97%
Out proposed model	SBERT	92.06%	96.79%	83.81%	81.54%	88.5%

Table 5.5: Accuracy Comparison

Chapter 6

Conclusion and Future Work

The goal of this paper was to create a personality prediction profile using text (social media postings) written by the user. We employed the "Support Vector Machine" machine learning technique, which is widely regarded as the best model for text classification. To estimate applicants' personality types, we used the "MBTI" dataset. We used random oversampling on our dataset to balance the skewed data. The sentences were then preprocessed (punctuation marks, emojis, and images were removed). For feature extraction, we employed the "SBERT" sentence transformer model for embedding. The performances of our model outperforms [20] which uses state-of-the-art model(BERT) for MBTI prediction.

Moving forward, we can set up a website that individuals from all around the world can use to correctly forecast their personality type. "Convolutional Neural Networks", a deep learning technique, can improve accuracy in comparison to other machine learning models. Consequently, applying deep learning models could be a way to enhance our outcomes. Also, various resources like LIWC (extracting linguistic features), EmoSenticNet (expresses emotions) and ConceptNet can be used as features for our model.

Chapter 7

Source Code

7.1 Basic Imports

```
import pandas as pd
import numpy as np
# PyViz
import seaborn as sns
# %matplotlib inline
import matplotlib.pyplot as plt
# NLP
import nltk
import re
from mpl_toolkits.mplot3d import Axes3D
from textblob import TextBlob
import spacy
from nltk import word_tokenize
import sys
import spacy
from nltk import word_tokenize
import nltk
nltk.download('averaged_perceptron_tagger')
#Basics
import pandas as pd
import numpy as np
from pathlib import Path
import os
import seaborn as sns
import nltk
import re
import matplotlib.pyplot as plt
# %matplotlib inline
#SKLearn
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split, GridSearchCV,
   StratifiedKFold, cross_val_score
#Metrics
import sklearn
from sklearn.metrics import r2_score, mean_squared_error,
   mean_absolute_error, accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,
    multilabel_confusion_matrix, confusion_matrix
from sklearn.metrics import classification_report
#Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
   GradientBoostingClassifier
import xgboost as xgb
# preprocessing
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import SelectKBest
from sklearn.compose import ColumnTransformer
# class imbalance
from imblearn.pipeline import make_pipeline as imb_make_pipeline
from imblearn.under_sampling import RandomUnderSampler
# algorithms/models
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.svm import LinearSVC, SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
# model evaluation
from imblearn.metrics import classification_report_imbalanced
from imblearn.metrics import geometric_mean_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import roc_auc_score
# performance check
import time
import warnings
warnings.filterwarnings("ignore")
# sparse to dense
from sklearn.base import TransformerMixin
```

```
73
74 import pandas as pd
75 import numpy as np
76 import locale
77 import locale
78 import spacy
79 import string
80 !pip install scikit-optimize
81 import pandas as pd
82 import numpy as np
83 import matplotlib.pyplot as plt
84 from sklearn.feature_extraction.text import TfidfVectorizer
85 from sklearn.model_selection import cross_val_score, StratifiedKFold
86 from sklearn.svm import SVC
87 from sklearn.metrics import accuracy_score, precision_score,
       recall_score, f1_score, confusion_matrix
88 from skopt import BayesSearchCV
89 from sklearn.model_selection import cross_val_score
90 from sklearn.model_selection import StratifiedKFold
91 from sklearn.linear_model import LogisticRegression
92 from sklearn import metrics
93 import pandas as pd
94 from sklearn.model_selection import cross_val_score
95 from sklearn.model_selection import StratifiedKFold
96 from sklearn.model_selection import train_test_split
97 from sklearn.ensemble import RandomForestClassifier
98 from sklearn import metrics
99 from sklearn.metrics import confusion_matrix
100 from sklearn.model_selection import cross_val_score
101 from sklearn.model_selection import StratifiedKFold
102 from sklearn.model_selection import train_test_split
103 from sklearn.neighbors import KNeighborsClassifier
104 from sklearn import metrics
105 from sklearn.metrics import confusion_matrix
106 from sklearn.model_selection import cross_val_score
107 from sklearn.model_selection import StratifiedKFold
108 from sklearn.model_selection import train_test_split
109 from sklearn.tree import DecisionTreeClassifier
110 from sklearn import metrics
111 from sklearn.metrics import confusion_matrix
112 from sklearn.model_selection import cross_val_score
113 from sklearn.model_selection import StratifiedKFold
114 from sklearn.svm import SVC
115 from sklearn import metrics
```

Source Code 7.1: Importing Basic Libraries and Models of Python.

7.2 Data Augmentation and Oversampling

```
class DenseTransformer(TransformerMixin):
    def fit(self, X, y=None, **fit_params):
        return self

    def transform(self, X, y=None, **fit_params):
        return X.todense()
from google.colab import drive
drive.mount('/content/drive/')
df2 = pd.read_csv("/content/drive/MyDrive/Dataset/Original_MBTI.csv")
df2.head()
# to handle the class imbalance better, converting the 16 classes into 4 more balanced classes
df2["is_Thinking"] = df2["type"].apply(
    lambda x: 1 if x[2] == "T" else 0
)
# rearranging the dataframe columns
df_combined = df2[
    ["is_Thinking", "posts"]
]
df_combined.head()
class_counts = df_combined['is_Thinking'].value_counts()
print(class_counts)
is_ex1,is_In0 =df_combined.is_Thinking.value_counts()
df_ex1 = df_combined[df_combined['is_Thinking']==1]
df_in0 = df_combined[df_combined['is_Thinking']==0]
is_ex1,is_In0

# Oversample 1-class and concat the DataFrames of both classes
df_ex1_under = df_ex1.sample(is_ex1, replace=True)
df_test_over = pd.concat([df_ex1_under, df_in0], axis=0)
print('Random over-sampling:')
print(df_test_over.is_Thinking.value_counts())
df_combined.to_csv('MBTI789.csv', index=False)
# shuffle the rows of the DataFrame
shuffled_df = df_test_over.sample(frac=1, random_state=42)  # frac=1 means shuffling the whole dataset
# reset the index of the shuffled DataFrame
shuffled_df = shuffled_df.reset_index(drop=True)
# save the shuffled DataFrame to a new CSV file
shuffled_df.to_csv('shuffled_dataset_sensing.csv', index=False)
```

Source Code 7.2: Loading Dataset and Augmenting the Dataset.

7.3 SBERT Model

```
def getpreferredencoding(do_setlocale = True):
    return "UTF-8"
locale.getpreferredencoding = getpreferredencoding
!pip install -U sentence-transformers -q
np.random.seed(2022)
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-mpnet-base-v2')
```

Source Code 7.3: Loading Pretrained SBERT Model.

7.4 Text Preprocessingl

```
nlp = spacy.load("en_core_web_sm")
stop_words = nlp.Defaults.stop_words
print(stop_words)
punctuations = string.punctuation
print(punctuations)
#removes links and special characters using simple regex statements.
def clean_phrase(phrase):
  return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w
   +:\/\/\S+)", " ", phrase).split())
# Creating our tokenizer function
def spacy_tokenizer(sentence):
    # Creating our token object, which is used to create documents
   with linguistic annotations.
    doc = nlp(sentence)

    # print(doc)
    # print(type(doc))
    # Lemmatizing each token and converting each token into
   lowercase
    mytokens = [ word.lemma_.lower().strip() for word in doc ]

    # print(mytokens)
    # Removing stop words
    mytokens = [ word for word in mytokens if word not in stop_words
    and word not in punctuations ]

    sentence = " ".join(mytokens)
    # return preprocessed list of tokens
    return sentence
shuffled_df['tokenize'] = shuffled_df['posts'].apply(spacy_tokenizer
   )
shuffled_df.head()
```

```
shuffled_df['embeddings'] = shuffled_df['tokenize'].apply(model.
    encode)
shuffled_df.head()
```

Source Code 7.4: Data Cleaning and Stopwords Removal.

7.5 Classical Machine Learning Models and Performance Evaluations

```
X = shuffled_df['embeddings'].to_list()
y = shuffled_df['is_Thinking'].to_list()
#LR with k fold
LR = LogisticRegression()
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  #
    Using 5-fold cross-validation
accuracy_scores = cross_val_score(LR, X, y, cv=skf, scoring='
    accuracy')
precision_scores = cross_val_score(LR, X, y, cv=skf, scoring='
    precision')
recall_scores = cross_val_score(LR, X, y, cv=skf, scoring='recall')
f1_scores = cross_val_score(LR, X, y, cv=skf, scoring='f1')
print("Logistic Regression Accuracy:", accuracy_scores.mean())
print("Logistic Regression Precision:", precision_scores.mean())
print("Logistic Regression Recall:", recall_scores.mean())
print("Logistic Regression F1 score:", f1_scores.mean())
# Confusion matrix using the last fold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, stratify=y, random_state=42)
LR.fit(X_train, y_train)
predicted = LR.predict(X_test)
cm = confusion_matrix(y_test, predicted)
print("Confusion Matrix:")
print(cm)

#KNN with k fold
KNN = KNeighborsClassifier(n_neighbors=5)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  #
    Using 5-fold cross-validation
accuracy_scores = cross_val_score(KNN, X, y, cv=skf, scoring='
    accuracy')
precision_scores = cross_val_score(KNN, X, y, cv=skf, scoring='
    precision')
recall_scores = cross_val_score(KNN, X, y, cv=skf, scoring='recall')
f1_scores = cross_val_score(KNN, X, y, cv=skf, scoring='f1')
print("KNN Accuracy:", accuracy_scores.mean())
print("KNN Precision:", precision_scores.mean())
```

```
print("KNN Recall:", recall_scores.mean())
print("KNN F1 score:", f1_scores.mean())
# Confusion matrix using the last fold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, stratify=y, random_state=42)
KNN.fit(X_train, y_train)
predicted = KNN.predict(X_test)
cm = confusion_matrix(y_test, predicted)
print("Confusion Matrix:")
print(cm)

#Random Forest with K fold
random_forest = RandomForestClassifier()
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  #
    Using 5-fold cross-validation
accuracy_scores = cross_val_score(random_forest, X, y, cv=skf,
    scoring='accuracy')
precision_scores = cross_val_score(random_forest, X, y, cv=skf,
    scoring='precision')
recall_scores = cross_val_score(random_forest, X, y, cv=skf, scoring
    ='recall')
f1_scores = cross_val_score(random_forest, X, y, cv=skf, scoring='f1
    ')
print("Random Forest Accuracy:", accuracy_scores.mean())
print("Random Forest Precision:", precision_scores.mean())
print("Random Forest Recall:", recall_scores.mean())
print("Random Forest F1 score:", f1_scores.mean())
# Confusion matrix using the last fold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, stratify=y, random_state=42)
random_forest.fit(X_train, y_train)
predicted = random_forest.predict(X_test)
cm = confusion_matrix(y_test, predicted)
print("Confusion Matrix:")
print(cm)

#Decision Tree with K fold
decision_tree = DecisionTreeClassifier()
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  #
    Using 5-fold cross-validation
accuracy_scores = cross_val_score(decision_tree, X, y, cv=skf,
    scoring='accuracy')
precision_scores = cross_val_score(decision_tree, X, y, cv=skf,
    scoring='precision')
recall_scores = cross_val_score(decision_tree, X, y, cv=skf, scoring
    ='recall')
f1_scores = cross_val_score(decision_tree, X, y, cv=skf, scoring='f1
```

```
    ')
print("Decision Tree Accuracy:", accuracy_scores.mean())
print("Decision Tree Precision:", precision_scores.mean())
print("Decision Tree Recall:", recall_scores.mean())
print("Decision Tree F1 score:", f1_scores.mean())
# Confusion matrix using the last fold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, stratify=y, random_state=42)
decision_tree.fit(X_train, y_train)
predicted = decision_tree.predict(X_test)
cm = confusion_matrix(y_test, predicted)
print("Confusion Matrix:")
print(cm)

#SVM with k fold
SVM = SVC()
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  #
    Using 5-fold cross-validation
accuracy_scores = cross_val_score(SVM, X, y, cv=skf, scoring='
    accuracy')
precision_scores = cross_val_score(SVM, X, y, cv=skf, scoring='
    precision')
recall_scores = cross_val_score(SVM, X, y, cv=skf, scoring='recall')
f1_scores = cross_val_score(SVM, X, y, cv=skf, scoring='f1')
print("SVM Accuracy:", accuracy_scores.mean())
print("SVM Precision:", precision_scores.mean())
print("SVM Recall:", recall_scores.mean())
print("SVM F1 score:", f1_scores.mean())
# Confusion matrix using the last fold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, stratify=y, random_state=42)
SVM.fit(X_train, y_train)
predicted = SVM.predict(X_test)
cm = confusion_matrix(y_test, predicted)
print("Confusion Matrix:")
print(cm)
```

Source Code 7.5: Performing ML models and Performance Evaluation.

7.6 Hyperparameter Tuning

```
# Create an SVM classifier
SVM = SVC()
# Define the hyperparameter search space
param_space = {
    'C': (0.1, 100.0, 'log-uniform'),  # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'],  # Kernel type
```

```
    'degree': (2, 4),  # Degree for the polynomial kernel
    'gamma': ['scale', 'auto']  # Gamma value
}
# Perform Bayesian optimization with cross-validation
bayes_cv = BayesSearchCV(SVM, param_space, scoring='accuracy', cv=5,
    n_jobs=-1)
bayes_cv.fit(X, y)
# Get the best parameters and the corresponding model
best_params = bayes_cv.best_params_
best_model = bayes_cv.best_estimator_
# Print the best parameters
print("Best Parameters:", best_params)
# Perform k-fold cross-validation with the best model
k = 5  # Number of folds
skf = StratifiedKFold(n_splits=k, shuffle=True)
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
for train_index, val_index in skf.split(X, y):
    X_train, X_val = np.array(X)[train_index], np.array(X)[val_index
    ]
    y_train, y_val = np.array(y)[train_index], np.array(y)[val_index
    ]
    best_model.fit(X_train, y_train)
    y_pred = best_model.predict(X_val)
    accuracy_scores.append(accuracy_score(y_val, y_pred))
    precision_scores.append(precision_score(y_val, y_pred))
    recall_scores.append(recall_score(y_val, y_pred))
    f1_scores.append(f1_score(y_val, y_pred))
mean_accuracy = np.mean(accuracy_scores)
mean_precision = np.mean(precision_scores)
mean_recall = np.mean(recall_scores)
mean_f1 = np.mean(f1_scores)
print("SVM Accuracy:", mean_accuracy)
print("SVM Precision:", mean_precision)
print("SVM Recall:", mean_recall)
print("SVM F1 Score:", mean_f1)
```

Source Code 7.6: Bayesian Optimization Hyperparameter Tuning with Cross Validation for SVM and Performance Evaluation.

References

[1] B. De Raad and B. Mlacic., "Big five factor model, theory and structure.," in *International encyclopedia of the social behavioral sciences 2*, pp. 559–566, 2015.

[2] G. J. Boyle, "Myers-briggs type indicator (mbti): some psychometric limitations.," in *Australian Psychologist 30.1*, pp. 71–74, 1995.

[3] D. J. Pittenger, "Measuring the mbti... and coming up short.," in *Journal of Career Planning and Employment 54.1*, pp. 48–52, 1993.

[4] "Logistic regression — detailed overview.." https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc. Accessed: 2023-05-11.

[5] "Support vector machines(svm)—an overview.." https://medium.com/towards-data-science/https-medium-com-pupalerushikesh-svm-f4b428 Accessed: 2023-05-11.

[6] "Decision trees explained easily.." https://chirag-sehra.medium.com/decision-trees-explained-easily-28f23241248. Accessed: 2023-05-11.

[7] "Understanding random forest.." https://towardsdatascience.com/understanding-random-forest-58381e0602d2. Accessed: 2023-05-11.

[8] "Oversampling and undersampling.." https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf. Accessed: 2023-05-02.

[9] "Tokenization in nlp." https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/. Accessed: 2023-05-02.

[10] "Lemmatization in nlp." https://towardsdatascience.com/stemming-vs-lemmatization-in-nlp-dea008600a0. Accessed: 2023-05-03.

[11] "Support vector machine (svm) hyperparameter tuning." https://medium.com/grabngoinfo/support-vector-machine-svm-hyperparameter-tuning Accessed: 2023-05-03.

[12] "Baysian optimization." https://www.analyticsvidhya.com/blog/2021/05/bayesian-optimization-bayes_opt-or-hyperopt/#:~:text=Hyperparameter%2Dtuning%20is%20the%20process,we%20find%20the%20best%20accuracy. Accessed: 2023-05-04.

[13] S. Ontoum and J. H. Chan., "Personality type based on myers-briggs type indicator with text posting style by using traditional and deep learning.," in *arXiv preprint arXiv:2201.08717*, 2022.

[14] B. Cui and C. Qi., "Survey analysis of machine learning methods for natural language processing for mbti personality type prediction.," in *Final Report Stanford University*, 2017.

[15] e. a. Pradhan, Tejas, "Analysis of personality traits using natural language processing and deep learning.," in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA).*, 2020.

[16] e. a. Bharadwaj, Srilakshmi, "Persona traits identification based on myers-briggs type indicator (mbti)-a text classification approach.," in *2018 international conference on advances in computing, communications and informatics (ICACCI). IEEE*, 2018.

[17] K. P. Brindha, S. and S. Sukumaran., "A survey on classification techniques for text mining.," in *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS). Vol. 1. IEEE,*, 2016.

[18] M. J. ·, "(MBTI) Myers-Briggs Personality Type Dataset." Kaggle.

[19] "Understanding confusion matrix." https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. Accessed: 2023-05-04.

[20] S. Majima and K. Markov., "Personality prediction from social media posts using text embedding and statistical features," in *2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS). IEEE*, 2022.

Generated using Undegraduate Thesis LaTeX Template, Version 1.4. Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka, Bangladesh.

This thesis was generated on Monday 15th May, 2023 at 7:21pm.