Text Classification: Group Assignment 1

Group: DSA_202101_9

Group Members:

Sheryl Shajan (300433896)

Lakshika Paiva (300323918)

Lenish Vaghasiya (300399630)

DTI/ELG 5125

Professor Arya Rahgozar

Faculty of Engineering, University of Ottawa

Table of Contents

1.0 Introduction to the Data

1.1 Introduction and Storyline

Background: Mental health is the foundation of our overall well-being, influencing how we think, feel, and interact with the world around us. It affects our ability to cope with stress, build relationships, and make decisions. Prioritizing mental health is essential for maintaining balance in our personal and professional lives, fostering resilience, and promoting physical health. By addressing mental health proactively, we can break stigma, encourage support, and ensure everyone has the opportunity to thrive emotionally, socially, and mentally.

As a result, we selected mental health as the theme of our analysis. Then, we selected five mental health disorders as the five categories for this assignment, namely,

- Clinical depression
- Bipolar disorder
- Anxiety disorder
- Post-traumatic stress disorder
- Schizophrenia

Challenge: Mental Health Research Scholars often use mental health research papers to conduct studies from popular databases like PubMed. However, the challenge is, there are many research papers in these categories. Scholars must manually search and read through abstracts before they select the relevant papers. This is a tedious and repetitive task that takes significant time and energy.

Solution: To ease their task, we, a team of data science experts, explore a suitable solution to programmatically extract abstracts of these mental health papers and classify them into given categories to help them in their work. We analyze the outcome of our work and present this report and results to Research Scholars in the mental health space.

1.2 Dataset Selection

For this classification assignment, we selected ‘abstracts’ of research papers published in mental health literature, as they contain a concise and precise summary of the subject (i.e., the mental health disorder). The extracted dataset consists of the category name (i.e., the label) and the abstract (200 partitions for each category, each containing 100 words).

PubMed, a popular, US based biomedical literature database was selected as the primary source to extract the research paper abstracts as it was publicly accessible and included large numbers of papers relevant to the mental health theme of this task.

It is important to note that we used Python as the programming language to perform all activities in the classification task, as it is widely used for data science.

2.0 Data Preprocessing and Cleansing

We developed a web scraping program to extract the abstracts from the research papers corresponding to each of the five categories, then preprocess and cleanse them.

These are the key steps carried out in data preprocessing:

- Extract the HTML content of a research paper linked to the category in PubMed database
- Extract the abstract from the research paper
- Cleans the text by removing the non-alphanumeric characters and the numbers which have no meaning, except for years beginning with '19' and '20'
- Splits the abstracts into separate words
- Randomly creates 200 partitions of abstracts containing 100 words each
- Labels the partitions with its corresponding category

The following figures from 1 to 6 present the data preprocessing code in the order they were executed.

- Several libraries were used for data preprocessing (refer figure 1) which we explain along with the data preprocessing code below. In figure 1, we import libraries.
    - We use the 'requests' library to fetch HTML content from PubMed, and the 'BeautifulSoup' library (from bs4) to parse the HTML and extract the abstract part of the web page.
- The function 'extract_abstract' (refer figure 1) takes the web link as an input, extracts the abstracts from the research paper in PubMed, and outputs the raw abstract as text.
- It connects to a research paper's webpage, searches for the "abstract" of the paper using the specific HTML structure of PubMed, and if found, it extracts the abstract text; otherwise, returns none.

Figure 1

- In figure 2, the function 'clean_text' (refer figure 2) takes the raw abstract as the input, and removes the non-alphanumeric characters and meaningless numbers, except for years starting with '19' and '20' (meaningless numbers were removed to ensure used features are generated).
- It uses the 're' library (i.e., regular expressions) for cleaning text by identifying patterns.

```
# Function to clean text by removing all non-alphanumeric characters and numbers except for years
def clean_text(text):
    # Remove all numbers except for years starting with '19' or '20'
    text = re.sub(r'\b(?!19\d{2}\b)(?!20\d{2}\b)\d+\b', '', text)  # Matches numbers not starting with '19xx' or '20xx' and
    # Remove all other non-alphanumeric characters
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return text
```

Figure 2

- In figure 3, the function ‘partition_text’, takes the cleaned abstracts as a text input and processes it to create and output 200 partitions per category, each having 100 words.
- It uses ‘random’ library to create randomized starting points for each partition in an abstract.
- We also created a similar dataset with 75 words per partition for performance comparison.

```python
# Function to partition text into exactly 200 partitions with specified words per partition
def partition_text(text, num_partitions=200, words_per_partition=100):
    words = text.split()
    partitions = []
    start_index = random.randint(0, max(0, len(words) - num_partitions * words_per_partition))
    for i in range(num_partitions):
        start = start_index + i * words_per_partition
        end = start + words_per_partition
        partition = " ".join(words[start:end])
        if len(partition.split()) == words_per_partition:  # Ensure partition has exactly 100 words
            partitions.append(partition)
        if len(partitions) >= num_partitions:
            break
    return partitions
```

Figure 3

- The program uses the ‘ThreadPoolExecutor’ library to speed up web scraping by downloading multiple abstracts, simultaneously (refer figure 4 and 5).
- The function ‘scrape_pubmed’ runs the program and calls the other functions within it to provide the cleaned abstracts as the output. In total, it provides 200 partitions per category.

```python
# Function to scrape PubMed for a specific query
def scrape_pubmed(query, max_results=200):
    url = f"https://pubmed.ncbi.nlm.nih.gov/?term={query.replace(' ', '+')}&size=200"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")

    abstracts = []
    results = soup.find_all("article", class_="full-docsum")

    links = ["https://pubmed.ncbi.nlm.nih.gov" + article.find("a", class_="docsum-title")["href"] for article in results]

    with ThreadPoolExecutor(max_workers=10) as executor:
        for abstract in executor.map(extract_abstract, links):
            if abstract:  # Only include articles with an abstract
                clean_abstract = clean_text(abstract)
                abstracts.append(clean_abstract)
            if len(abstracts) >= max_results:
                break

    return abstracts
```

Figure 4

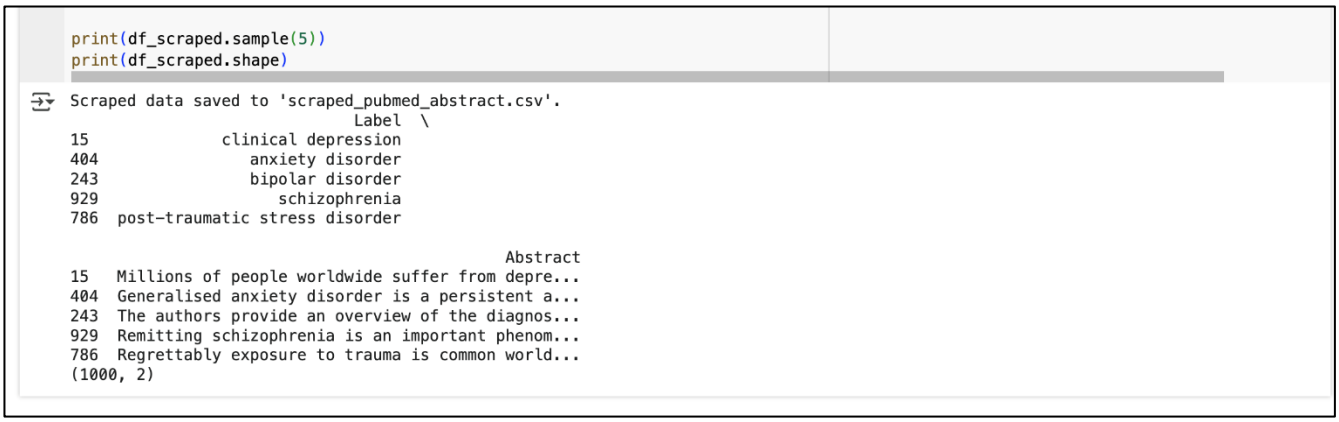
```python
# Scrape data for each category
scraped_data = []
# Define categories
categories = {
    "Category1": "clinical depression",
    "Category2": "bipolar disorder",
    "Category3": "anxiety disorder",
    "Category4": "post-traumatic stress disorder",
    "Category5": "schizophrenia"
}
```

```python
for idx, (label, category) in enumerate(categories.items()):
    data = []
    while len(data) < 200:
        data.extend(scrape_pubmed(category, max_results=200 - len(data)))
    partitions = []
    for abstract in data:
        partitions.extend(partition_text(abstract))
        if len(partitions) >= 200:
            break
    partitions = partitions[:200]  # Ensure exactly 200 partitions per category
    for partition in partitions:
        if partition.strip():  # Ensure partition is not empty
            scraped_data.append({"Label": category, "Abstract": partition})

# Save the scraped data to a CSV
df_scraped = pd.DataFrame(scraped_data, columns=["Label", "Abstract"])
df_scraped.to_csv("scraped_pubmed_abstract.csv", index=False)

print("Scraped data saved to 'scraped_pubmed_abstract.csv'.")
```

Figure 5

- It searches PubMed for a specific keyword (e.g., "clinical depression"), and collects links to papers from the search results. Then, it uses extract_abstract to get abstracts from the links.
- Thereafter, it cleans the abstracts and limits the results to the specified maximum (200).
- The dictionary (categories) defines 5 mental health disorders as search terms (figure 5).
- For each category:
    - PubMed is scraped for 200 abstracts. Each abstract is split into 200 smaller text partitions, each with 100 words.
    - These partitions are labeled with the disorder's name and added to a list.

- At the end of figure 5, we save the data into a .CSV file.
- We use ‘pandas’ library for organizing and saving data in a DataFrame and exporting it into a .CSV file.
- All the labeled partitions are compiled into a table (DataFrame) with two columns:
    - Label: The disorder's name (e.g., "clinical depression").
    - Abstract: The 100-word text partition.
- The table is saved as a CSV file (scraped_pubmed_abstract.csv) and downloaded.
- Figure 6 prints the first 5 rows of the saved data and the total number of rows (refer figure 6).



```
print(df_scraped.sample(5))
print(df_scraped.shape)
```

```
Scraped data saved to 'scraped_pubmed_abstract.csv'.
                              Label  \
15              clinical depression
404                anxiety disorder
243                bipolar disorder
929                   schizophrenia
786  post-traumatic stress disorder

                                              Abstract
15   Millions of people worldwide suffer from depre...
404  Generalised anxiety disorder is a persistent a...
243  The authors provide an overview of the diagnos...
929  Remitting schizophrenia is an important phenom...
786  Regrettably exposure to trauma is common world...
(1000, 2)
```

Figure 6

As the output of this preprocessing and cleansing step, we produce two files which have 100 words per partition and 75 words per partition. Our default selection is 100 words per partition; however, we produced the other file (i.e., 75 words per partition) to make a comparison of the accuracy of our predictions which is described in a later step (i.e., section 8).

Output:

- scraped_pubmed_abstract_100.csv (default)
- scraped_pubmed_abstract _75.csv

3.0 Feature Engineering

3.1 Bag of Words vs TF-IDF Evaluation

Bag of words featurization quantifies the frequency of words in text documents for processing in machine learning models. Its variation Term Frequency-Inverse Document Frequency (TF-IDF) generates models that further account for word frequency across a corpus of documents (Murel and Kavlakoglu, 2024).

We experimented with Bag of Words and TF-IDF techniques for feature engineering and chose to go ahead with TF-IDF for the following reasons:

- It can handle stop words (in English), and rare words with higher weights to them
- It is more suitable for large vocabulary, and it is in a normalized form
- Mental health research papers often have domain-specific terminology (e.g., 'schizophrenia' or 'cognitive behavior'). In a technical domain like this, TF-IDF performs better because it emphasizes these unique terms, improving classification
- Research abstracts often contain common words (like 'study' or 'method'). TF-IDF reduces their impact by assigning lower weights.

3.2 Implementation of TF-IDF

The following codes illustrates how we 'feature engineer', or in other words convert the abstracts into the TF-IDF format.

- First, all required libraries are installed and imported into the environment (i.e., Google Collab). Enable resources when needed.

```
[ ]  import requests
     from bs4 import BeautifulSoup
     import pandas as pd
     from concurrent.futures import ThreadPoolExecutor
     import re  # Import the re module for regular expressions
     import random  # Import the random module

     import nltk
     from nltk.corpus import stopwords
     from nltk.tokenize import word_tokenize
     from nltk.stem import WordNetLemmatizer
     import pandas as pd

     from sklearn.feature_extraction.text import TfidfVectorizer

     # Download necessary NLTK resources
     #nltk.download('punkt')
     #nltk.download('stopwords')
     #nltk.download('wordnet')
     #nltk.download('punkt_tab')

     from sklearn.pipeline import Pipeline
     from sklearn.model_selection import GridSearchCV, cross_val_score
     from sklearn.svm import SVC
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.naive_bayes import MultinomialNB
     from xgboost import XGBClassifier
     from sklearn.linear_model import SGDClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import classification_report
     from sklearn.model_selection import train_test_split
```

Figure 7

- As this is the beginning of the code, we already discussed some of them in the previous preprocessing step.

- We use the ‘nltk’ library to preprocess the text by tokenizing abstracts into words. This also allows us to perform lemmatization and remove stop words.

```
[ ] df_scraped = pd.read_csv('scraped_pubmed_abstract_100.csv')

    df_scraped.shape

(1000, 2)

[ ] df_scraped.head()
```

	Label	Abstract
0	clinical depression	Although anxiety and depression have been cons...
1	clinical depression	Major depression is one of the most prevalent ...
2	clinical depression	ObjectiveAdolescence is a formative and turbul...
3	clinical depression	2001 to 2020 was CI Point prevalence for major...
4	clinical depression	Major depressive disorder MDD is considered a ...

Figure 7

- In figure 7 above, we import the pre-prepared text file with partitions for faster processing. This is converted into a data frame using the ‘Pandas’ library.

- In figure 8, the function preprocess_text cleans and simplifies the abstracts so it is easier for the machine to understand.

Text Preprocessing and Vectorizing

```
[42] nltk.download('punkt_tab')

    # Initialize lemmatizer and stopwords
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    def preprocess_text(text):
        # Tokenize the text
        words = word_tokenize(text.lower())
        # Remove stop words and non-alphanumeric tokens
        filtered_words = [lemmatizer.lemmatize(word) for word in words if word.isalpha() and word not in stop_words]

        return " ".join(filtered_words)

    # Example of transforming text using TF-IDF
    def transform_to_tfidf(partitions):
        vectorizer = TfidfVectorizer(max_features=1000, ngram_range=(1, 2))  # Bigrams (1,2) and top 1000 features
        tfidf_matrix = vectorizer.fit_transform(partitions)
        return tfidf_matrix, vectorizer

[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Figure 8

- In figure 8, the abstracts are tokenized, in which it breaks down the input abstracts into smaller chunks, specifically words, and converts everything to lowercase for consistency.
- Then it removes ‘stop words’, like *the, is, and, etc.*, which don’t carry much meaning. Further, it removes anything that's not a word (e.g., numbers or punctuation).
- Thereafter, the words in abstracts are simplified to their base form. For example, *studying* becomes *study*, and *better* becomes *good*. This helps group similar words together.
- The cleaned-up words are then joined back together as a single string.

```
df_scraped['Processed_Text'] = df_scraped['Abstract'].apply(preprocess_text)
print(df_scraped[['Label', 'Processed_Text']].head())
```

```
                Label                                     Processed_Text
0  clinical depression  although anxiety depression considered two dis...
1  clinical depression  major depression one prevalent debilitating pe...
2  clinical depression  objectiveadolescence formative turbulent phase...
3  clinical depression  ci point prevalence major depressive disorder ...
4  clinical depression  major depressive disorder mdd considered serio...
```

Figure 9

- Figure 9 shows that only the abstracts in the relevant column are processed in this manner. However, the labels remain the same.
- For example, ‘although anxiety and depression are considered as two...’ becomes ‘although anxiety depression considered two...’

- In figure 10, TF-IDF converts the processed abstracts into a numerical format that machine learning algorithms can work with. TF-IDF assigns numbers to words based on how important they are within an abstract and across multiple abstracts.
    - TF-IDF Vectorizer looks at the cleaned abstracts and counts how often words (or pairs of words, called bigrams) appear. Then, each abstract is turned into a row of numbers, where each column represents a word or bigram and its importance.
- The transform_to_tfidf function is applied to the Processed_Text column, converting all abstracts into this numerical matrix.
- The resulting matrix is ready for our machine learning task: classification of abstracts.
- Figure 10 prints and illustrates the feature names generated by TF-IDF.

```
# Transform the preprocessed text into TF-IDF features
tfidf_matrix, vectorizer = transform_to_tfidf(df_scraped['Processed_Text'])

print(vectorizer.get_feature_names_out())
```

```
 'pharmacologic' 'pharmacological' 'pharmacological intervention'
 'pharmacotherapy' 'phase' 'phenomenology' 'phenomenon' 'phenotype'
 'phobia' 'physical' 'physical mental' 'place' 'placebo' 'play' 'point'
 'pooled' 'poor' 'population' 'positive' 'possible' 'post' 'posttraumatic'
 'posttraumatic stress' 'potential' 'potentially' 'practice' 'preclinical'
 'predictor' 'prepubertal' 'prescribing' 'presence' 'present'
 'present study' 'presentation' 'presented' 'prevalence' 'prevalence rate'
 'prevalent' 'preventing' 'preventing weight' 'prevention' 'previous'
 'previously' 'primary' 'primary care' 'primary outcome' 'problem'
 'process' 'profile' 'progress' 'progression' 'proinflammatory'
 'proportion' 'proportion time' 'proposed' 'protein' 'provide' 'provided'
 'provides' 'psychiatric' 'psychiatric disorder' 'psychiatry'
 'psychoeducation' 'psychological' 'psychology' 'psychopathological'
```

Figure 10

Once feature engineering is complete, the output is used in the modelling for classification (next section), as models can only process abstracts transformed in such a manner.

4.0 Modelling for Classification

We used the following 6 algorithms to implement models for classification:

- **Classical Models:**
  - Support Vector Machine (SVM)
  - Random Forest
  - MultinomialNB
  - XGBoost
  - K-Nearest Neighbour (kNN)
  - Stochastic Gradient Descent (SGD)
- **Deep Learning Model:** Bidirectional Encoder Representations from Transformers (BERT)

Thereafter we compare the performance of each of these models which each other.

4.1 Application of Classical Machine Learning Models for Classification

The code in figure 11 defines a list of machine learning models that will be tested. These models are briefly mentioned below and why they were chosen.

- **Support Vector Machine (SVC):** Good for classification, finds the best boundary between categories.
- **Random Forest Classifier:** Uses many decision trees to make predictions, reducing errors.
- **Multinomial Naïve Bayes:** Works well with text classification and categorical data.
- **XGBoost:** A powerful, fast, and optimized decision-tree-based model.
- **k-Nearest Neighbors (KNN):** Compares new data points to existing ones to determine classification.
- **Stochastic Gradient Descent (SGD):** A fast, simple model for large datasets that updates weights step by step.

In figure 11, we define the hyperparameters for each classification model to set up the tuning options and optimize performance.

- SVC:
  - C: Controls how strict the model is (higher = more complex).
  - kernel: Defines the method to find decision boundaries (linear or RBF).
  - gamma: Adjusts how much influence a single data point has.
- Random Forest:
  - n_estimators: Number of decision trees in the forest.
  - max_depth: Maximum depth of each tree (None means unlimited).
  - min_samples_split: Minimum samples needed to split a node.
- Naive Bayes:
  - alpha: A smoothing factor that prevents zero probabilities.
- K-Nearest Neighbours
  - n_neighbors: Number of nearest data points to consider for classification.
  - weights: Defines how much influence each neighbor has.
  - metric: Defines how distance is measured (Euclidean or Manhattan).
- SGD:

- loss: Defines the type of loss function used (hinge, log_loss, etc.).
- penalty: Defines the type of regularization (to prevent overfitting).
- alpha: Learning rate to control weight updates.
- max_iter: Maximum number of iterations to optimize the model.

```
# Define models
models = [
    SVC(),  # Support Vector Machine
    RandomForestClassifier(random_state=42),  # Random Forest Classifier
    MultinomialNB(),  # Naïve Bayes
   # XGBClassifier(use_label_encoder=False, eval_metric='logloss'),  # XGBoost
    KNeighborsClassifier(),  # k-Nearest Neighbors
    SGDClassifier(random_state=42)  # Stochastic Gradient Descent
]

# Define corresponding parameter grids
parameters = [
    # SVM
    {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    },
    # Random Forest
    {
        'n_estimators': [50, 100, 200],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5, 10]
    },
    # Naïve Bayes
    {
        'alpha': [0.1, 1.0, 10.0]
    },
        # XGBoost
        {
            'n_estimators': [50, 100, 200],
            'learning_rate': [0.01, 0.1, 0.2],
            'max_depth': [3, 5, 10]
        },
        {#KNN
            'n_neighbors': [3, 5, 10],
            'weights': ['uniform', 'distance'],
            'metric': ['euclidean', 'manhattan']
        },
        # SGD
        {
            'loss': ['hinge', 'log_loss', 'modified_huber'],
            'penalty': ['l2', 'l1', 'elasticnet'],
            'alpha': [0.0001, 0.001, 0.01],
            'max_iter': [1000, 2000]
        }
    ]
```

Figure 11

- In figure 12, we split the dataset into training (80%) and test (20%) datasets.
- Here, df_scraped['Label'] contains the actual categories (labels) of the abstracts.

```
[45] labels = df_scraped['Label']
     X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix, labels, test_size=0.2, random_state=42)
```

Figure 12

- In figure 13 below, this function ‘grid search best’ runs an automated search to find the best settings (hyperparameters) for a given model using GridSearchCV. It uses ten-fold cross-validation (cv=10), which means it tests the model 10 times with different parts of the training data to get a more reliable result (to prevent overfitting).
- The *scoring='accuracy'* measures how well the model predicts correctly.
- Then it saves the best version of each model.
- Finally, as the output it shows the best settings, and the highest accuracy score found (figure 14).

```
[ ]
    grid_search_best = []

    # Iterate through models and parameters
    for model, params in zip(models, parameters):
        print(f"Running GridSearch for {model.__class__.__name__}...")

        # Perform grid search
        grid_search = GridSearchCV(
            model,
            param_grid=params,
            cv=10,  # 10-fold cross-validation
            scoring='accuracy',
            verbose=2,
            n_jobs=-1
        )

        # Fit the grid search
        grid_search.fit(X_train, y_train_encoded)

        # Append the best model
        grid_search_best.append((model.__class__.__name__, grid_search.best_estimator_))

        # Output results
        print(f"Best parameters for {model.__class__.__name__}: {grid_search.best_params_}")
        print(f"Best cross-validation score for {model.__class__.__name__}: {grid_search.best_score_}\n")
```

Figure 13

```
Running GridSearch for SVC...
Fitting 10 folds for each of 12 candidates, totalling 120 fits
Best parameters for SVC: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Best cross-validation score for SVC: 0.9825000000000002

Running GridSearch for RandomForestClassifier...
Fitting 10 folds for each of 27 candidates, totalling 270 fits
Best parameters for RandomForestClassifier: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 100}
Best cross-validation score for RandomForestClassifier: 0.9812500000000002

Running GridSearch for MultinomialNB...
Fitting 10 folds for each of 3 candidates, totalling 30 fits
Best parameters for MultinomialNB: {'alpha': 10.0}
Best cross-validation score for MultinomialNB: 0.975

Running GridSearch for XGBClassifier...
Fitting 10 folds for each of 27 candidates, totalling 270 fits
Best parameters for XGBClassifier: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Best cross-validation score for XGBClassifier: 0.9787500000000001

Running GridSearch for KNeighborsClassifier...
Fitting 10 folds for each of 12 candidates, totalling 120 fits
Best parameters for KNeighborsClassifier: {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}
Best cross-validation score for KNeighborsClassifier: 0.96875

Running GridSearch for SGDClassifier...
Fitting 10 folds for each of 54 candidates, totalling 540 fits
Best parameters for SGDClassifier: {'alpha': 0.0001, 'loss': 'log_loss', 'max_iter': 1000, 'penalty': 'elasticnet'}
Best cross-validation score for SGDClassifier: 0.9850000000000001
```

Figure 14

- In figure 15, we encode labels, that is convert text labels to numbers because many machine learning models cannot work directly with text labels (like "anxiety disorder", "schizophrenia", etc.). The LabelEncoder() does this conversion automatically.
- The printed output shows the mapping of category names to numbers.

```
from sklearn.preprocessing import LabelEncoder

# Encode the target labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Check the mapping for reference
print("Label Mapping:")
for i, label in enumerate(label_encoder.classes_):
    print(f"{i}: {label}")
```

```
Label Mapping:
0: anxiety disorder
1: bipolar disorder
2: clinical depression
3: post-traumatic stress disorder
4: schizophrenia
```

Figure 15

4.2 Application of BERT Algorithm

Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art natural language processing (NLP) model developed by Google, designed to better understand the context of words in a sentence. Unlike traditional unidirectional models that process text from left to right or right to left, BERT uses a bidirectional approach to learn context from both directions, improving its understanding of language.

We have explored test classification using BERT, the goal was to assess the performance of BERT and compare it with the other models.

- The dataset was split into training and testing sets, with an 80-20% split using 'train_test_split'.
- The BertTokenizer from the Hugging Face transformers library was used to tokenize the text. Text was truncated or padded to a maximum length of 200 tokens. Refer figure 16 below.

Figure 16

- The labels (mental health conditions) were encoded using LabelEncoder to transform categorical labels into numerical values that could be processed by the model (refer figure 17).

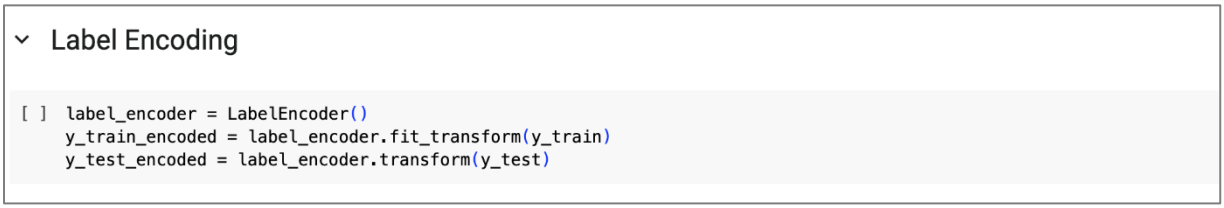


Figure 17

- In figure 18, we define a **custom dataset class** called MentalHealthDataset, which is used to handle mental health data in **PyTorch**. The purpose of this class is to structure the data so that it can be easily fed into a machine learning model for training and testing.

Prepare a Dataset for PyTorch

+ Code + Text

```
[ ] class MentalHealthDataset(torch.utils.data.Dataset):
        def __init__(self, encodings, labels):
            self.encodings = encodings
            self.labels = torch.tensor(labels, dtype=torch.long)

        def __len__(self):
            return len(self.labels)

        def __getitem__(self, idx):
            item = {key: val[idx] for key, val in self.encodings.items()}
            item['labels'] = torch.tensor(self.labels[idx])
            return item

    # Create datasets
    train_dataset = MentalHealthDataset(train_encodings, y_train_encoded)
    test_dataset = MentalHealthDataset(test_encodings, y_test_encoded)
```

Figure 18

- In figure 19, the BERT model used was 'bert-base-uncased', a pre-trained version of BERT that works with English text. It was fine-tuned for the sequence classification task.
- In figure 19, 'Set up Training with PyTorch' of the code is setting up data loading, optimization, and GPU usage for training a machine learning model in PyTorch.
    - DataLoader: Data was prepared using PyTorch's DataLoader to manage batching, shuffling, and loading during training.
    - Optimizer is how the model learns: The `AdamW` optimizer was chosen for fine-tuning the BERT model, with a learning rate of `2e-5` and weight decay `0.01`.
    - The model was moved to GPU (if available) for faster training (see figure 19 and 20).
    - Scheduler: A learning rate scheduler was set up to decay the learning rate linearly throughout the training process. This improves the stability of training and fine-tuning.
    - Mixed Precision: The training loop utilizes mixed precision with `autocast` and `GradScaler` to speed up training while maintaining model accuracy. This is particularly helpful when training large models like BERT on GPU.

Load a pretrained BERT model

```
[ ] model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=len(label_encoder.classes_))
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Set Up Training with PyTorch

```
[ ] # Define data loaders
    train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=32)

    # Define optimizer
    optimizer = AdamW(model.parameters(), lr=2e-5, weight_decay=0.01)

    # Move model to GPU if available
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    model.to(device)
```

Figure 19

Figure 20

- In figure 21, the training loop iterates for 5 epochs, performing the following steps in each epoch:
    - Forward Pass: The model processes a batch of tokenized text and calculates the loss.
    - Backward Pass: The loss is backpropagated to adjust model weights.
    - Gradient Scaling: The GradScaler is used to scale the gradients for stability during mixed precision training.
    - Optimizer Step: The optimizer updates the model's parameters based on the scaled gradients.
    - Learning Rate Scheduler Step: The learning rate is adjusted based on the scheduler.

Figure 21

In the next section, we evaluate the performance of these models. **The best performing models are identified after the performance evaluation.**

4.3 Performance Evaluation

1) Performance Evaluation using the 100 words per partition dataset:

We evaluate the performance of the models using several techniques:

- **Accuracy score:** Measures how many predictions were correct as a percentage
- **Macro F1 score:** Arithmetic mean (unweighted mean) of all the per-class F1 scores
- **Classification report:** A summary table that displays the following for each class:
    - Precision – Of all the positive predictions made, how many were true positives
    - Recall – Of all the actual positives, how many were predicted to be positive
    - F1-score – A balance between precision and recall (harmonic mean of precision and recall)
- **Confusion matrix:** Shows how well a classification model is performing by comparing its predictions to the actual values. It helps identify where the model is making errors.

We use these libraries to help measure and visualize the performance of our machine learning models used in section 4.1:

- classification_report - Summarizes model performance.
- confusion_matrix - Shows a table comparing actual vs. predicted labels.
- accuracy_score - Calculates how often the model is correct.
- seaborn and matplotlib.pyplot - Create a heatmap to visualize results.

The code in the following figure 23 goes through the best performing models:

- Then it uses the trained model to predict the categories for the test data for each model.
- Thereafter, it converts the numeric labels back to the text labels or categories we defined.
- The performance metrics are then printed one after another.

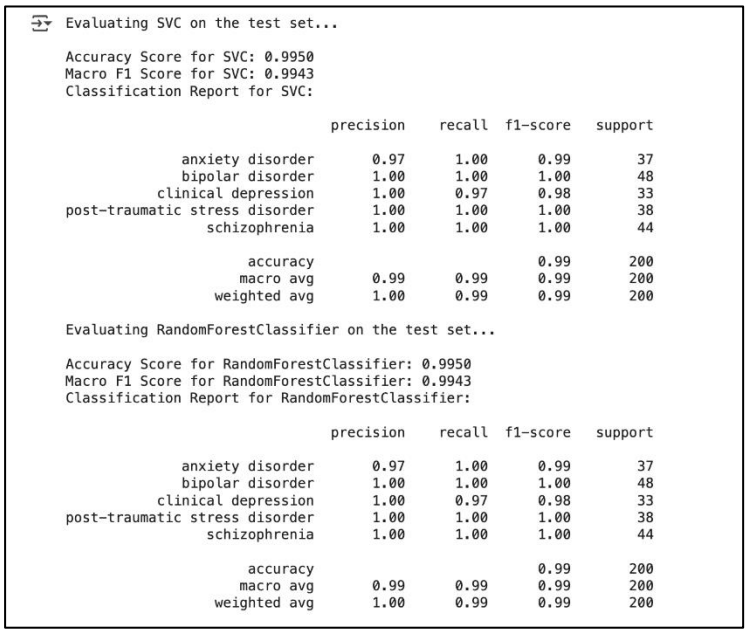
Evaluate Models and Interpret Results

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,f1_score
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[ ] performance_data = []

    for model_name, best_model in grid_search_best:
        print(f"Evaluating {model_name} on the test set...\n")

        y_pred_encoded = best_model.predict(X_test)
        y_pred_decoded = label_encoder.inverse_transform(y_pred_encoded)
        y_test_decoded = label_encoder.inverse_transform(y_test_encoded)

        # Accuracy
        acc = accuracy_score(y_test_decoded, y_pred_decoded)

        # Macro F1-score
        macro_f1 = f1_score(y_test_decoded, y_pred_decoded, average='macro')

        print(f"Accuracy Score for {model_name}: {acc:.4f}")
        print(f"Macro F1 Score for {model_name}: {macro_f1:.4f}")

        performance_data.append({'Model': model_name, 'Accuracy': acc, 'Macro F1-Score': macro_f1})

        # Classification Report
        print(f"Classification Report for {model_name}:\n")
        print(classification_report(y_test_decoded, y_pred_decoded))

    # Convert performance data into a DataFrame
    performance_df = pd.DataFrame(performance_data)
    print("\nPerformance Summary:\n", performance_df)
```

Figure 23

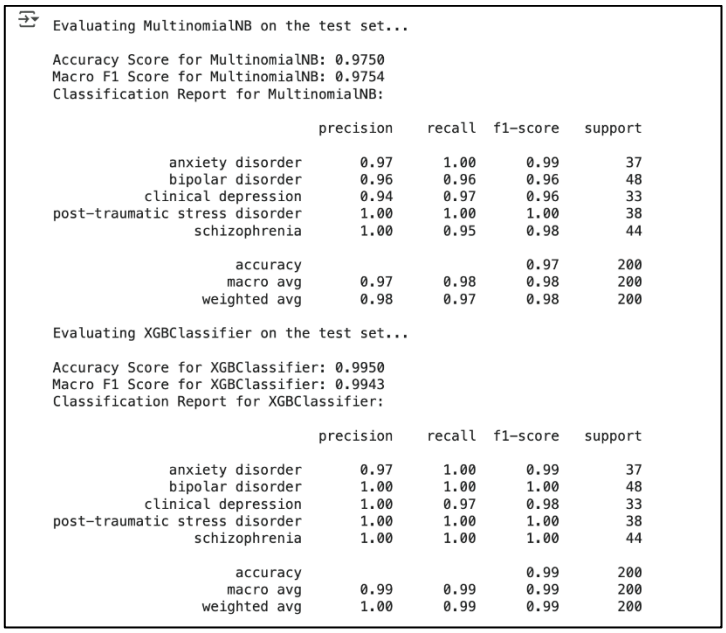
The following figures present the individual performance of each of the models.



```
Evaluating SVC on the test set...

Accuracy Score for SVC: 0.9950
Macro F1 Score for SVC: 0.9943
Classification Report for SVC:

                                precision    recall  f1-score   support

             anxiety disorder       0.97      1.00      0.99        37
             bipolar disorder       1.00      1.00      1.00        48
          clinical depression       1.00      0.97      0.98        33
post-traumatic stress disorder       1.00      1.00      1.00        38
                  schizophrenia       1.00      1.00      1.00        44

                       accuracy                           0.99       200
                      macro avg       0.99      0.99      0.99       200
                   weighted avg       1.00      0.99      0.99       200

Evaluating RandomForestClassifier on the test set...

Accuracy Score for RandomForestClassifier: 0.9950
Macro F1 Score for RandomForestClassifier: 0.9943
Classification Report for RandomForestClassifier:

                                precision    recall  f1-score   support

             anxiety disorder       0.97      1.00      0.99        37
             bipolar disorder       1.00      1.00      1.00        48
          clinical depression       1.00      0.97      0.98        33
post-traumatic stress disorder       1.00      1.00      1.00        38
                  schizophrenia       1.00      1.00      1.00        44

                       accuracy                           0.99       200
                      macro avg       0.99      0.99      0.99       200
                   weighted avg       1.00      0.99      0.99       200
```

Figure 24

- In figure 24, SVC showed a higher precision (i.e, 100%) across all categories except for anxiety disorder which was 97%, a lower precision. However, the recall was lower for clinical depression category at 97% with all other categories having 100% recall. As a balanced measure, the F1 score was 100% for 3 of the categories except for anxiety (99%) and clinical depression (98%).
- As for Random Forest, the results are the same.

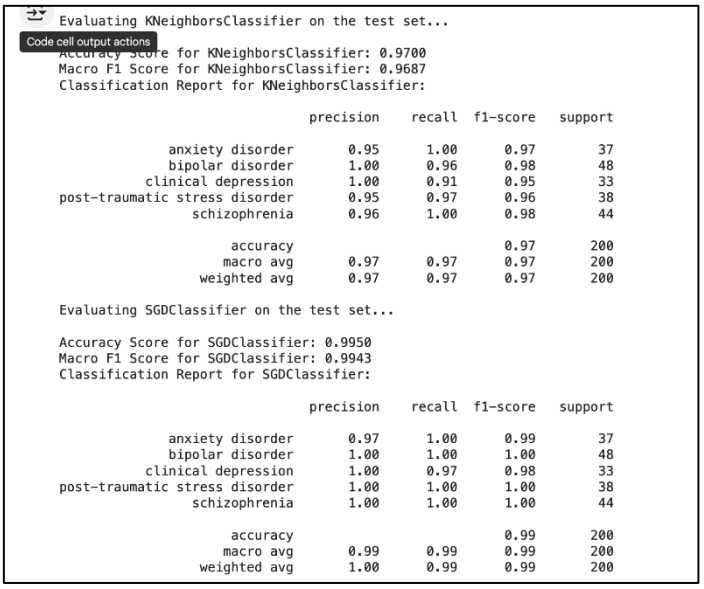


```
Evaluating MultinomialNB on the test set...

Accuracy Score for MultinomialNB: 0.9750
Macro F1 Score for MultinomialNB: 0.9754
Classification Report for MultinomialNB:

                                precision    recall  f1-score   support

             anxiety disorder       0.97      1.00      0.99        37
             bipolar disorder       0.96      0.96      0.96        48
          clinical depression       0.94      0.97      0.96        33
post-traumatic stress disorder       1.00      1.00      1.00        38
                  schizophrenia       1.00      0.95      0.98        44

                       accuracy                           0.97       200
                      macro avg       0.97      0.98      0.98       200
                   weighted avg       0.98      0.97      0.98       200

Evaluating XGBClassifier on the test set...

Accuracy Score for XGBClassifier: 0.9950
Macro F1 Score for XGBClassifier: 0.9943
Classification Report for XGBClassifier:

                                precision    recall  f1-score   support

             anxiety disorder       0.97      1.00      0.99        37
             bipolar disorder       1.00      1.00      1.00        48
          clinical depression       1.00      0.97      0.98        33
post-traumatic stress disorder       1.00      1.00      1.00        38
                  schizophrenia       1.00      1.00      1.00        44

                       accuracy                           0.99       200
                      macro avg       0.99      0.99      0.99       200
                   weighted avg       1.00      0.99      0.99       200
```

Figure 25

- In figure 25, MultinomialNB showed a higher precision (i.e, 100%) across PTSD and schizophrenia, however, the precision was lower for anxiety (97%), bipolar (96%) and clinical depression (94%). This means there were many false positives in their outcomes.
- The recall was higher (100%) for anxiety and PTSD, but lower for bipolar (96%), clinical depression (97%) and schizophrenia (95%), which means it had more false negatives.

- As a balanced measure, only PTSD had an F1 score of 100%, whereas others were relatively lower but good scores above 95%.
- As for XGBoost in figure 25, the results were comparatively higher and improved. It showed a higher precision (i.e, 100%) across all categories except for anxiety disorder which was 97%, a lower precision. However, the recall was lower for clinical depression category at 97% with all other categories having 100% recall. As a balanced measure, the F1 score was 100% for 3 of the categories except for anxiety (99%) and clinical depression (98%).



```
Evaluating KNeighborsClassifier on the test set...
Code cell output actions
Accuracy Score for KNeighborsClassifier: 0.9700
Macro F1 Score for KNeighborsClassifier: 0.9687
Classification Report for KNeighborsClassifier:

                               precision    recall  f1-score   support

              anxiety disorder       0.95      1.00      0.97        37
              bipolar disorder       1.00      0.96      0.98        48
           clinical depression       1.00      0.91      0.95        33
post-traumatic stress disorder       0.95      0.97      0.96        38
                  schizophrenia       0.96      1.00      0.98        44

                       accuracy                           0.97       200
                      macro avg       0.97      0.97      0.97       200
                   weighted avg       0.97      0.97      0.97       200

Evaluating SGDClassifier on the test set...

Accuracy Score for SGDClassifier: 0.9950
Macro F1 Score for SGDClassifier: 0.9943
Classification Report for SGDClassifier:

                               precision    recall  f1-score   support

              anxiety disorder       0.97      1.00      0.99        37
              bipolar disorder       1.00      1.00      1.00        48
           clinical depression       1.00      0.97      0.98        33
post-traumatic stress disorder       1.00      1.00      1.00        38
                  schizophrenia       1.00      1.00      1.00        44

                       accuracy                           0.99       200
                      macro avg       0.99      0.99      0.99       200
                   weighted avg       1.00      0.99      0.99       200
```

Figure 26a

- In figure 26a, k-NN results fluctuated again. The precision was lower for anxiety (95%), PTSD (95%) and schizophrenia (96%). For recall, the numbers dropped again for bipolar (96%), clinical depression (91%) and PTSD (97%). As a result, F1 scores of all categories ranged from 95% to 98%, which is relatively a good number but lower than other models.

- As for SGD in figure 26a, the results were comparatively higher. It showed a higher precision (i.e, 100%) across all categories except for anxiety disorder which was 97%, a lower precision. However, the recall was lower for clinical depression category at 97% with all other categories having 100% recall. As a balanced measure, the F1 score was 100% for 3 of the categories except for anxiety (99%) and clinical depression (98%).

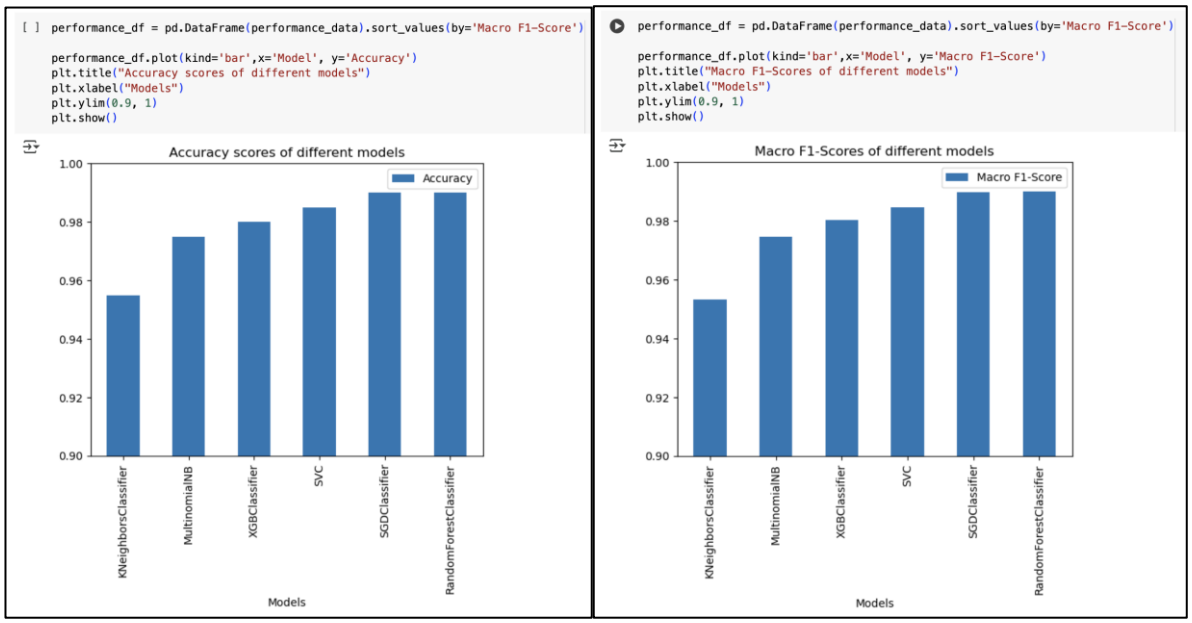
2) Performance Evaluation using the 75 words per partition dataset:

We performed the same evaluation using the 75 words per partition dataset for comparison. Overall, the accuracy was lower than those results that were generated for the 100 words per partition dataset.

- As per figure 26b below, Random Forest Classifier reported the highest and best performance in accuracy (99%) and Macro F1-Score (99%).
- The SGD classifier came close in accuracy, but it was lower in Macro F1-Score at 98%.
- The lowest performing model in both attributes was k-NN Classifier.

```
Performance Summary:
                      Model  Accuracy  Macro F1-Score
0                       SVC     0.985        0.984745
1  RandomForestClassifier     0.990        0.990149
2           MultinomialNB     0.975        0.974750
3           XGBClassifier     0.980        0.980378
4    KNeighborsClassifier     0.955        0.953336
5           SGDClassifier     0.990        0.989681
```

Figure 26b



The following tables report the performance in terms of precision, recall and f1-score of each model.

```
Evaluating SVC on the test set...

Classification Report for SVC:

                                precision    recall  f1-score   support

              anxiety disorder       0.97      0.97      0.97        37
              bipolar disorder       1.00      1.00      1.00        48
           clinical depression       1.00      1.00      1.00        33
post-traumatic stress disorder       1.00      0.95      0.97        38
                  schizophrenia       0.96      1.00      0.98        44

                       accuracy                           0.98       200
                      macro avg       0.99      0.98      0.98       200
                   weighted avg       0.99      0.98      0.98       200

Evaluating RandomForestClassifier on the test set...

Classification Report for RandomForestClassifier:

                                precision    recall  f1-score   support

              anxiety disorder       1.00      0.97      0.99        37
              bipolar disorder       1.00      1.00      1.00        48
           clinical depression       1.00      1.00      1.00        33
post-traumatic stress disorder       1.00      0.97      0.99        38
                  schizophrenia       0.96      1.00      0.98        44

                       accuracy                           0.99       200
                      macro avg       0.99      0.99      0.99       200
                   weighted avg       0.99      0.99      0.99       200

Evaluating MultinomialNB on the test set...
```

Figure 26c

```
Classification Report for MultinomialNB:

                                precision    recall  f1-score   support

              anxiety disorder       0.95      1.00      0.97        37
              bipolar disorder       1.00      0.94      0.97        48
           clinical depression       0.94      1.00      0.97        33
post-traumatic stress disorder       1.00      0.95      0.97        38
                  schizophrenia       0.98      1.00      0.99        44

                      accuracy                           0.97       200
                     macro avg       0.97      0.98      0.97       200
                  weighted avg       0.98      0.97      0.97       200

Evaluating XGBClassifier on the test set...

Classification Report for XGBClassifier:

                                precision    recall  f1-score   support

              anxiety disorder       1.00      0.95      0.97        37
              bipolar disorder       0.98      1.00      0.99        48
           clinical depression       1.00      1.00      1.00        33
post-traumatic stress disorder       0.97      0.97      0.97        38
                  schizophrenia       0.96      0.98      0.97        44

                      accuracy                           0.98       200
                     macro avg       0.98      0.98      0.98       200
                  weighted avg       0.98      0.98      0.98       200

Evaluating KNeighborsClassifier on the test set...
```

Figure 26d

```
Classification Report for KNeighborsClassifier:

                                precision    recall  f1-score   support

              anxiety disorder       0.88      1.00      0.94        37
              bipolar disorder       0.98      0.98      0.98        48
           clinical depression       0.92      1.00      0.96        33
post-traumatic stress disorder       1.00      0.87      0.93        38
                  schizophrenia       1.00      0.93      0.96        44

                      accuracy                           0.95       200
                     macro avg       0.96      0.96      0.95       200
                  weighted avg       0.96      0.95      0.95       200

Evaluating SGDClassifier on the test set...

Classification Report for SGDClassifier:

                                precision    recall  f1-score   support

              anxiety disorder       0.97      1.00      0.99        37
              bipolar disorder       1.00      1.00      1.00        48
           clinical depression       1.00      1.00      1.00        33
post-traumatic stress disorder       1.00      0.95      0.97        38
                  schizophrenia       0.98      1.00      0.99        44

                      accuracy                           0.99       200
                     macro avg       0.99      0.99      0.99       200
                  weighted avg       0.99      0.99      0.99       200
```

Figure 26e

BERT's Model and Performance Evaluation using the 100 words dataset:

- Once the model finished training, it was evaluated on the test dataset. Predictions were made by processing the test batches through the trained model. The predicted labels were then compared to the true labels using classification report to compute various performance metrics such as:
    - Precision
    - Recall
    - F1-Score
    - Accuracy
- The output of the classification report showed that the model achieved an accuracy of 99%, with high precision and recall across all five mental health conditions (see figure 26f below).

```
model.eval()
predictions = []
true_labels = []

with torch.no_grad():
    for batch in test_loader:
        batch = {key: val.to(device) for key, val in batch.items()}
        outputs = model(**batch)
        logits = outputs.logits
        predictions.extend(torch.argmax(logits, dim=-1).cpu().numpy())
        true_labels.extend(batch['labels'].cpu().numpy())

# Decode labels
predictions = label_encoder.inverse_transform(predictions)
true_labels = label_encoder.inverse_transform(true_labels)

# Classification report
print(classification_report(true_labels, predictions))
```

```
                              precision    recall  f1-score   support

              anxiety disorder       0.98      1.00      0.99        40
              bipolar disorder       0.97      0.97      0.97        40
          clinical depression        1.00      1.00      1.00        40
post-traumatic stress disorder       1.00      1.00      1.00        40
                 schizophrenia       1.00      0.97      0.99        40

                      accuracy                           0.99       200
                     macro avg       0.99      0.99      0.99       200
                  weighted avg       0.99      0.99      0.99       200
```

Figure 26f

In comparison, the BERT model with the 75 words dataset performed the same in its accuracy but relatively lower in its F1-Score (refer figure 26g below).

```
[ ] model.eval()
    predictions = []
    true_labels = []

    with torch.no_grad():
        for batch in test_loader:
            batch = {key: val.to(device) for key, val in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            predictions.extend(torch.argmax(logits, dim=-1).cpu().numpy())
            true_labels.extend(batch['labels'].cpu().numpy())

    # Decode labels
    predictions = label_encoder.inverse_transform(predictions)
    true_labels = label_encoder.inverse_transform(true_labels)

    # Classification report
    print(classification_report(true_labels, predictions))
```

```
                              precision    recall  f1-score   support

              anxiety disorder       1.00      1.00      1.00        40
              bipolar disorder       1.00      0.97      0.99        40
          clinical depression        0.95      1.00      0.98        40
post-traumatic stress disorder       0.98      1.00      0.99        40
                 schizophrenia       1.00      0.95      0.97        40

                      accuracy                           0.98       200
                     macro avg       0.99      0.98      0.98       200
                  weighted avg       0.99      0.98      0.98       200
```

Figure 26g

K-fold Cross-Validation (CV)

K-fold Cross-Validation is a robust technique for assessing machine learning models. We performed 10-fold Cross-Validation for our models.

- 10-fold CV averages performance over 10 different data splits. This reduces the impact of randomness in the training/testing split and gives a more stable estimate of model accuracy. The model accuracy was discussed above.

- Each data point gets to be in both training and test sets (once in test, nine times in train). This is crucial when dealing with small datasets, such as our 1,000 PubMed abstracts, ensuring the model learns from all available data.
- If a model performs well on the training set but poorly on the test folds, it indicates overfitting. We catch such issues by adopting 10-fold CV before deploying the model.
- It gives multiple test accuracy scores, which are averaged to give a final metric (e.g., accuracy, F1-score) as shown above. This is more reliable than a single train-test split, which might have selection bias.
- We used 10-fold cross-validation in combination with GridSearchCV to find the best hyperparameters for our models like SVM, Random Forest, Naïve Bayes, SGD, k-NN, MultinomialNB and XGBoost.

In summary, using 10-fold cross-validation:

- Ensures the model learns from all available abstracts.
- Provides more stable accuracy estimates.
- Helps select the best hyperparameters for models like SVM and Naïve Bayes.

4.4 Comparison and Decision of the Champion Model

Figure 27a and 27b below, is a summary of the performance evaluation in all classical models.

In terms of accuracy, the models SVC, Random Forest, XGBoost and SGD had the highest accuracy among 100-word abstracts, which was 99.5% and relatively high. This is because higher scores closer to 1 are better in performance. In comparison, the MultinomialNB and k-NN models had a lower accuracy which were 97.5% and 97% respectively. These models were accurate, but relatively lower than previously mentioned models. Moreover, the Macro F1-Score depicted a similar pattern in demonstrating how well our machine learning model performs across different categories (mental health disorders) without bias toward any class.

Furthermore, we identified that BERT model's performance was relatively high for both datasets. However, it is well-known that BERT does not perform well for small datasets, hence its application to our dataset with 1000 abstracts was not a fair and appropriate evaluation of the BERT model.

Finding 1: Based on the overall accuracy and macro F1 score (figure 27a), SVC, **Random Forest,** XGBoost and SGD models performed the best for this classification activity of **abstracts with 100 words**. The other two models performed relatively less with higher inaccuracies and F1 scores.

Finding 2: Based on the overall accuracy and macro F1 score (27b below), **Random Forest** model performed the best for this classification activity of **abstracts with 75 words**. The other five models performed relatively less with higher inaccuracies.

Performance Summary:

	Model	Accuracy	Macro F1-Score
0	SVC	0.995	0.994256
1	RandomForestClassifier	0.995	0.994256
2	MultinomialNB	0.975	0.975394
3	XGBClassifier	0.995	0.994256
4	KNeighborsClassifier	0.970	0.968721
5	SGDClassifier	0.995	0.994256

Figure 27a

Performance Summary:

	Model	Accuracy	Macro F1-Score
0	SVC	0.985	0.984745
1	RandomForestClassifier	0.990	0.990149
2	MultinomialNB	0.975	0.974750
3	XGBClassifier	0.980	0.980378
4	KNeighborsClassifier	0.955	0.953336
5	SGDClassifier	0.990	0.989681

Figure 27b

Conclusion of the Champion Model:

In conclusion, the **TF-IDF features in combination with the Random Forest model**, performed the best for this classification activity of research paper abstracts.
Overall, this means that mental health research scholars would receive more accurate and reliable mental health research paper abstract classification results with programs that are tuned to run on the **Random Forest** model, for the two specific datasets used in this assignment.

5.0 Error Analysis

In this section we analyze the confusion matrices and resulting misclassifications and show the properties of the abstract segments that threw the model off.

5.1 Confusion Matrices

For the dataset with 100 words per partition:

To identify and analyze the errors, we plot confusion matrices for all models as shown in figure 28 which is presented as a heatmap. The confusion matrix shows correct and incorrect predictions for each category. The diagonal values are correct predictions and the others are misclassifications.

```
[ ] # Plot all confusion matrices in a 3-row, 2-column layout
    fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(15, 15))
    axes = axes.flatten()  # Flatten for easy indexing

    for idx, (model_name, best_model) in enumerate(grid_search_best):
        if idx >= 6:  # Limit to 6 models for visualization
            break

        y_pred_encoded = best_model.predict(X_test)
        y_pred_decoded = label_encoder.inverse_transform(y_pred_encoded)

        # Compute confusion matrix
        cm = confusion_matrix(y_test_decoded, y_pred_decoded)

        # Plot confusion matrix
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_, ax=axes[idx])
        axes[idx].set_title(f"Confusion Matrix: {model_name}")
        axes[idx].set_xlabel("Predicted")
        axes[idx].set_ylabel("Actual")

    plt.tight_layout()
    plt.show()
```

Figure 28

As shown in the following figures 29a and 29b, the models SVC, Random Forest, XGBoost, and SGD all have one misclassified abstract. These are our best performing models in the 100 words dataset.

The MultinomialNB and the k-NN models have 5 and 6 misclassified abstracts, respectively. In the error analysis section, we analyze the errors in depth.

Refer to the next section for deeper analysis of the errors.

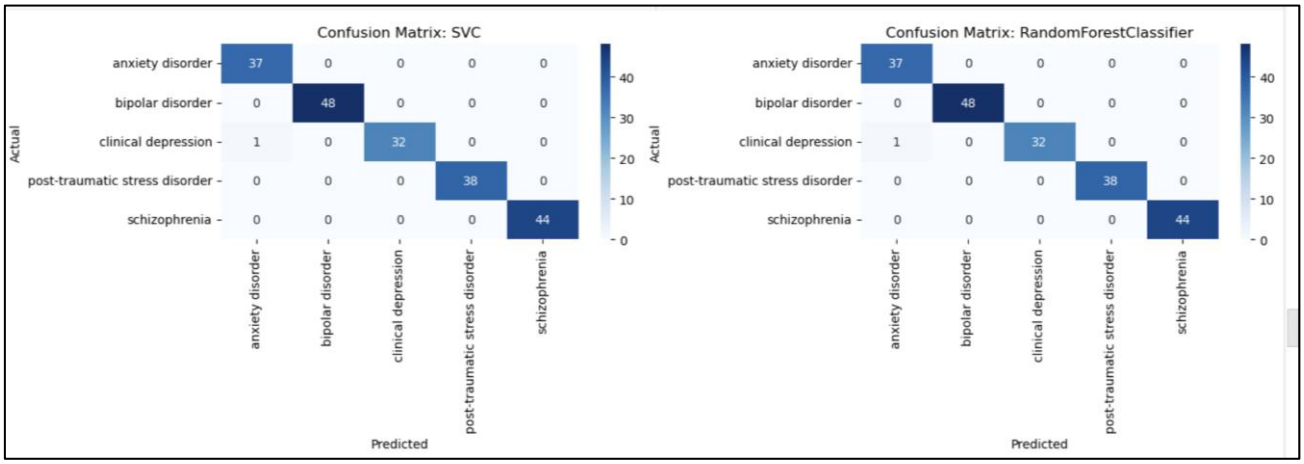


Figure 29a

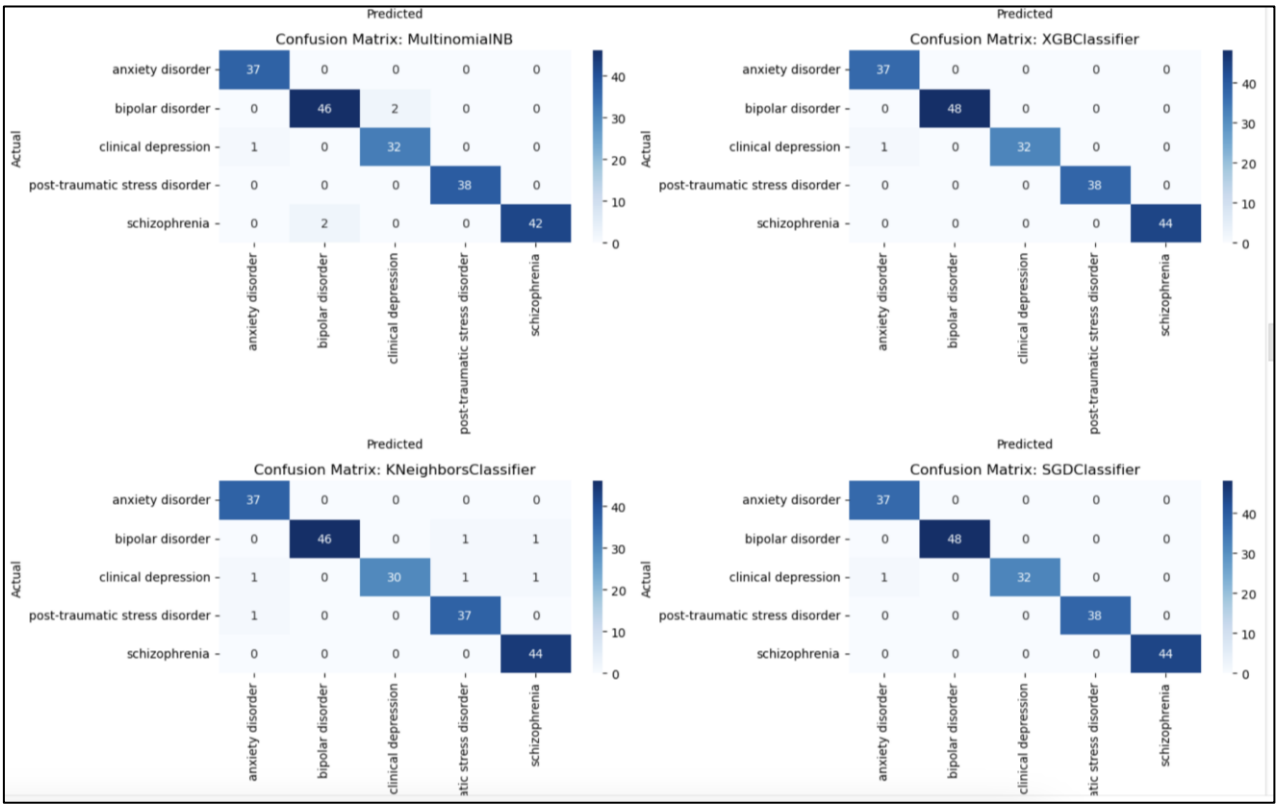


Figure 29b

For the dataset with 75 words per partition:

As shown in the following figures 29c, 29d and 29e, the models have a number of misclassifications, specifically 3 in SVC, 2 in Random Forest, 5 in MultinomiaNB, 4 in 9 in k-NN, 4 in XGBoost, and 2 in SGD. These are our best performing models in the 100 words dataset.

Overall, it is clear that there are more errors in models when evaluated with the 75 words per partition dataset. In the error analysis section, we analyze these errors as trends, in depth.

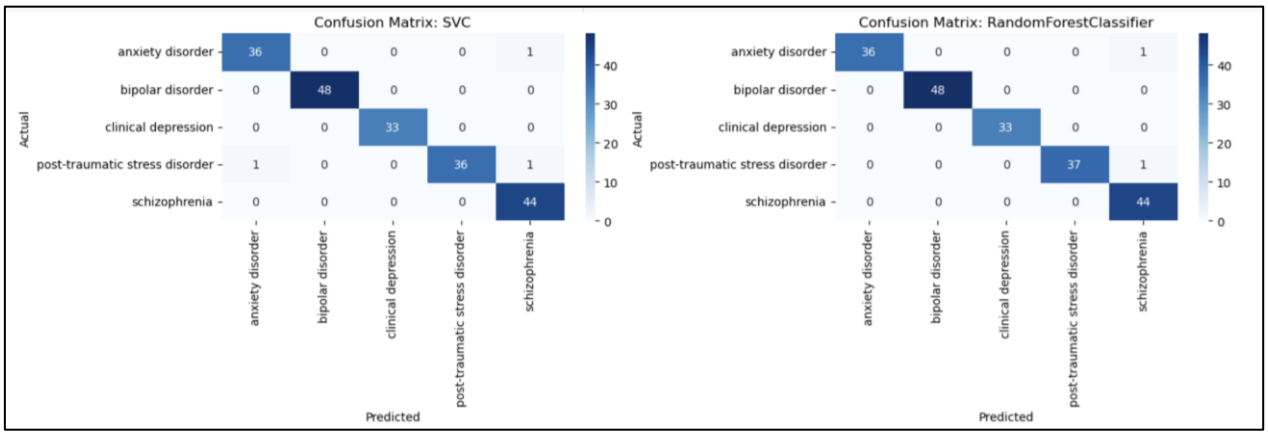


Figure 29c

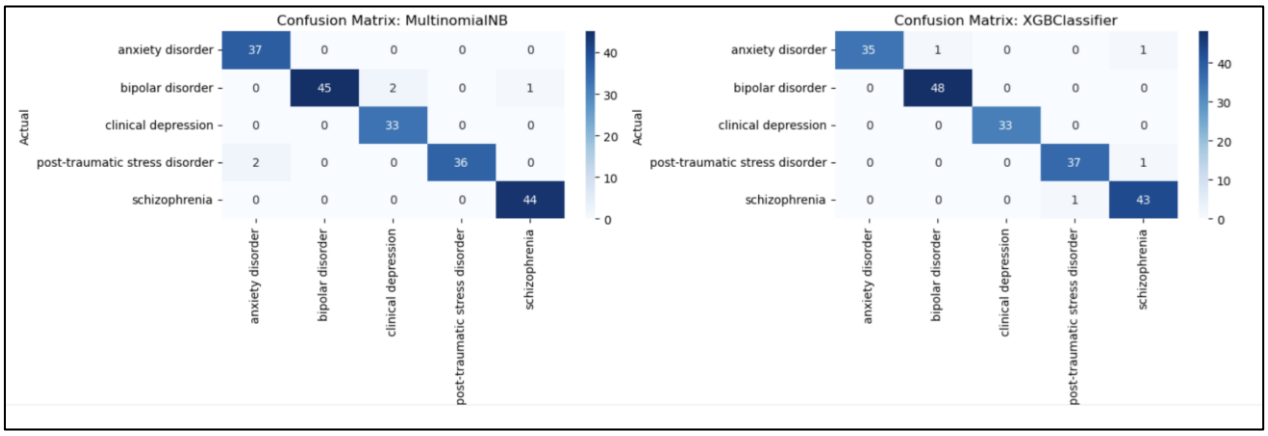


Figure 29d

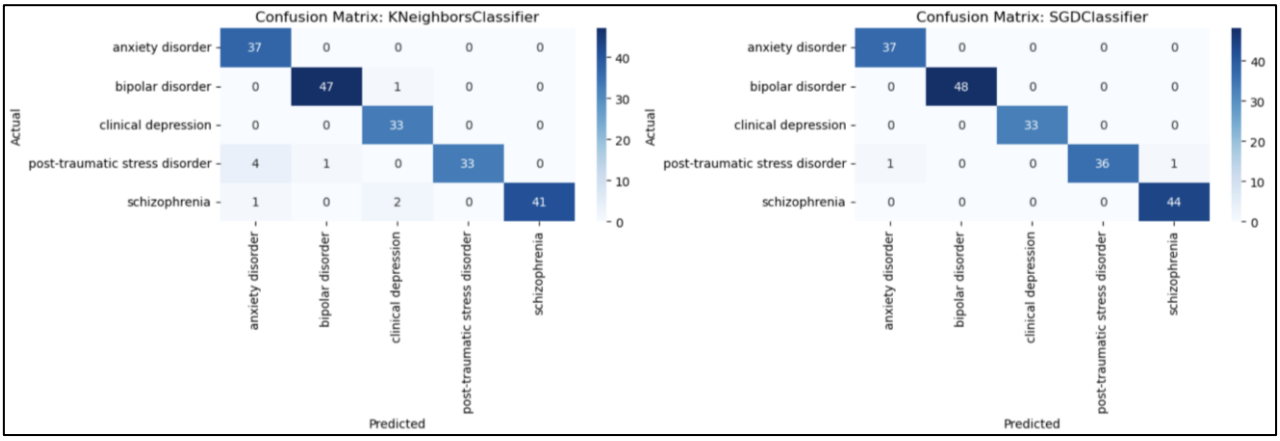


Figure 29e

5.2 Error Analysis

Misclassification Analysis for 100 words per partition dataset

We traced the abstracts of the misclassified rows by each model for the dataset with 100 words per partition. Figure 30 shows the code for extracting the corresponding abstract samples from the misclassified lot. The following section analyzes the abstracts closely and presents the error analysis.

```
[ ]  # Analyzing misclassified samples separately
     misclassified_samples = {}

     for model_name, best_model in grid_search_best:
         y_pred_encoded = best_model.predict(X_test)
         y_pred_decoded = label_encoder.inverse_transform(y_pred_encoded)

         # Create a DataFrame of misclassified samples
         errors = pd.DataFrame({
             'Text': X_test_text,
             'True Label': y_test_decoded,
             'Predicted Label': y_pred_decoded
         })
         errors = errors[errors['True Label'] != errors['Predicted Label']]

         misclassified_samples[model_name] = errors  # Store for future use
         print(f"Misclassified Samples for {model_name}:\n", errors.head())
```

Figure 30

Figure 31 shows the misclassified abstracts for SVC, **RandomClassifier,** XGBoost and SGD models, which are the best performing models with the 100-word dataset. We iterate that our chosen Champion Model, **RandomClassifier** is also amongst them. There was only one misclassified row in each of them which was the same abstract. Here are some key insights derived from deeper analysis:

- **Insight 1: Some papers focus on two or more mental health disorders in the analyzed categories.** In figure 31, the models SVC, Random Forest, XGBoost and SGD all identified one abstract as misclassified. While closely analyzing the abstract "An informationprocessing paradigm was used to examine", it included the word 'anxiety'. Occasionally, mental health research papers in clinical depression also discuss anxiety as a disorder as they tend to occur together. Hence, mental health research scholars would find a few papers in clinical depression applicable to anxiety studies as well.

```
Misclassified Samples for SVC:
                                                 Text           True Label  \
67  An informationprocessing paradigm was used to ...  clinical depression

     Predicted Label
67  anxiety disorder
Misclassified Samples for RandomForestClassifier:
                                                 Text           True Label  \
67  An informationprocessing paradigm was used to ...  clinical depression

     Predicted Label
67  anxiety disorder
Misclassified Samples for XGBClassifier:
                                                 Text           True Label  \
67  An informationprocessing paradigm was used to ...  clinical depression

     Predicted Label
67  anxiety disorder
Misclassified Samples for SGDClassifier:
                                                 Text           True Label  \
67  An informationprocessing paradigm was used to ...  clinical depression

     Predicted Label
67  anxiety disorder
```

Figure 31

Figure 32 shows the misclassified abstracts for MultinomialNB and k-NN models (non-champion models, see below). There were 5 and 6 abstracts respectively, which were misclassified in each model as per the confusion matrices.

```
Misclassified Samples for MultinomialNB:
                                                  Text           True Label  \
261  For the treatment of acute BPII depression two...     bipolar disorder
866  relationship to other conditions particularly ...        schizophrenia
67   An informationprocessing paradigm was used to ...  clinical depression
822  relationship to other conditions particularly ...        schizophrenia
361  For the treatment of acute BPII depression two...     bipolar disorder

         Predicted Label
261  clinical depression
866     bipolar disorder
67      anxiety disorder
822     bipolar disorder
361  clinical depression
Misclassified Samples for XGBClassifier:
                                                  Text           True Label  \
67  An informationprocessing paradigm was used to ...  clinical depression

     Predicted Label
67  anxiety disorder
Misclassified Samples for KNeighborsClassifier:
                                                  Text  \
319  Machine learning techniques provide new method...
67   An informationprocessing paradigm was used to ...
635  Symptom provocation has proved its worth for u...
55   From them only received psychotropics compared...
72   Clinical depression and other psychological di...

                        True Label                Predicted Label
319               bipolar disorder                   schizophrenia
67              clinical depression                   schizophrenia
635  post-traumatic stress disorder                anxiety disorder
55              clinical depression  post-traumatic stress disorder
72              clinical depression                anxiety disorder
```

Figure 32

- **Insight 2: Some mental health disorders have <u>similar impacts</u>, and often have similar words, such as clinical depression and bipolar disorder.** In figure 32, like Insight 1, the abstract from MultinomialNB "For the treatment of acute BPII depression two..." had a similar issue. Bipolar disorder causes depression; hence the word 'depression' is used often in the abstract. Hence, it is misclassified as clinical depression. Similarly, in k-NN model, the abstract "Symptom provocation has proved its worth for u..." belongs to 'PTSD' but it is considered as an 'anxiety disorder' because it causes stress in both cases which appears in the abstract causing the error. Mental health scholars focusing on one disorder should be aware of the interdependence and impacts of disorders in the real-world and use the classification model with caution.

- **Insight 3: Some papers <u>analyze the relationship</u> of one mental health disorder to another such as schizophrenia to bipolar disorder.** In deeper analysis for the abstracts in the MultinomialNB model, the abstract "relationship to other conditions particularly..." belonged to 'shizophrenia' but it was erroneously classified as 'bipolar disorder' because closer analysis revealed that it contains the word 'bipolar'. Some researchers in mental health choose to analyze the relationship between two or more mental health disorders, so research scholars should be aware of this concern.

- **Insight 4: There are <u>synonyms that relate to other disorders</u> in an abstract.** In the k-NN results, the abstract "From them only received psychotropics compared..." revealed that it

belongs to ‘clinical depression’, but it is predicted as ‘PTSD’. In closer analysis, it shows that it does not include the complete word for PTSD or its acronym, but it includes similar words like ‘stress’ and ‘disorder’ which causes misclassification as these are likely features to identify ‘PTSD’ abstracts.

- **Insight 5: There is specialized jargon in the abstracts, which misclassifies abstracts.** While closely analyzing the above misclassified abstracts, it was clear that noise in the abstracts with technical jargon caused errors that led to misidentified predicted labels for 2-3 abstracts. For example, in this abstract “relationship to other conditions particularly ...” (in MultinomialNB) the true label is ‘schizophrenia’, but it has been predicted as ‘bipolar disorder’. Similarly, the abstract “Machine learning techniques provide new method...” (in k-NN), the true label is ‘bipolar disorder’, but it has been predicted as ‘schizophrenia’.

Word Cloud: Moreover, we created word clouds per category (figure 33 to 38 below) of the most important features that were used in the classification process. We identified a key outcome error in this:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Generating a word cloud for "anxiety disorder" abstracts
anxiety_text = " ".join(df_scraped[df_scraped['Label'] == 'anxiety disorder']['Abstract'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(anxiety_text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Word Cloud for Anxiety Disorder Abstracts")
plt.show()

# Generating a word cloud for "bipolar disorder" abstracts
bipolar_text = " ".join(df_scraped[df_scraped['Label'] == 'bipolar disorder']['Abstract'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(bipolar_text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Word Cloud for Bipolar Disorder Abstracts")
plt.show()

# Generating a word cloud for "clinical depression" abstracts
clinical_depression_text = " ".join(df_scraped[df_scraped['Label'] == 'clinical depression']['Abstract'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(clinical_depression_text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Word Cloud for Clinical Depression Abstracts")
plt.show()
```

Figure 33

- **Insight 6: Generalized words were included in the top features or words generated for prediction in each category.** The word clouds generated for each mental health disorder category (refer figures 34 to 37b) present the most frequent words or phrases found in the abstracts for each category. The size of each word in the cloud corresponds to how often it appears across the abstracts. In this task notably, close analysis reveals that general words like ‘patient’, ‘treatment’, ‘major’, ‘disorder’ and ‘study’ were included in the classification of abstracts (by the program) for all the 5 categories of disorders. This had caused confusion and noise and affected the prediction results and resulted in misclassification.

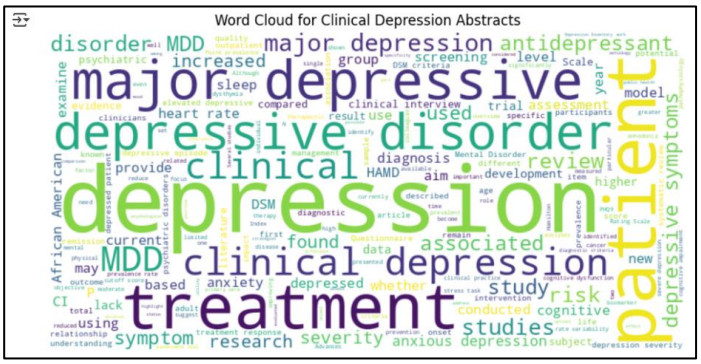


Figure 34

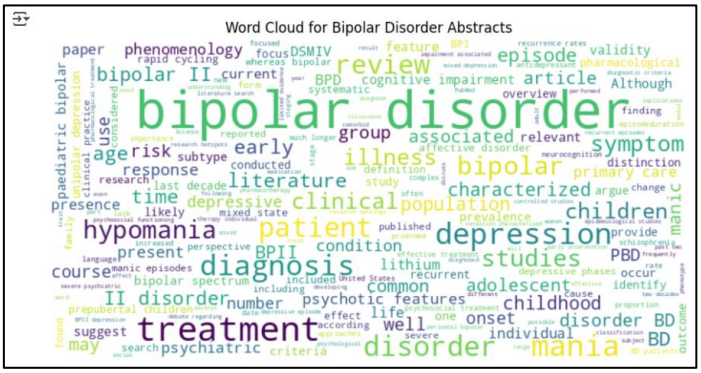


Figure 35

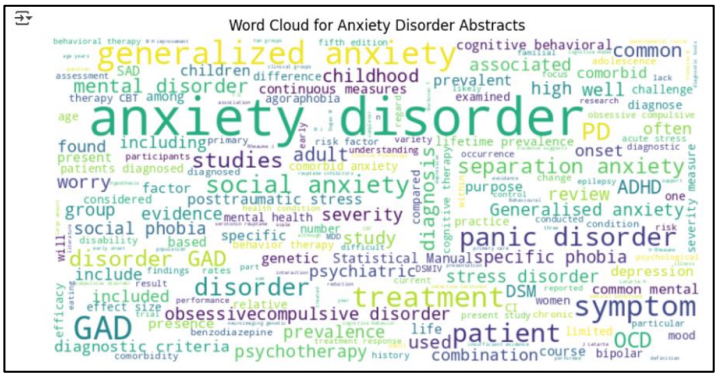


Figure 36

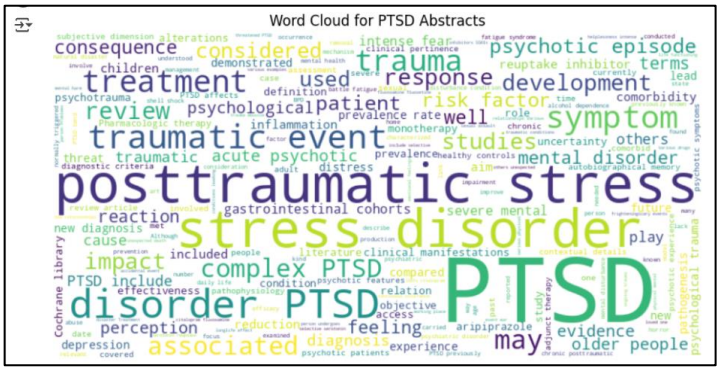


Figure 37a

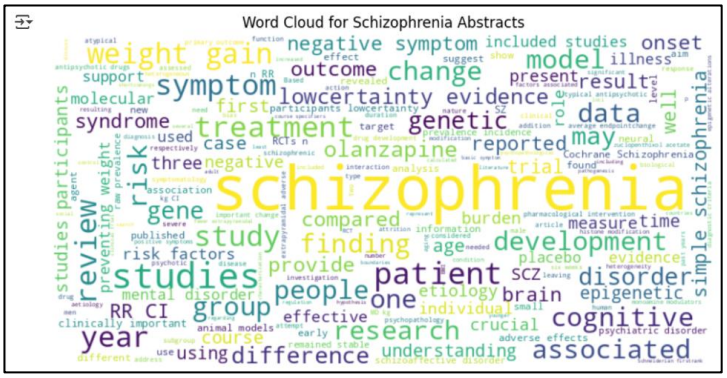


Figure 37b

Misclassification Trends

For the dataset with 75 words per partition, we conducted a misclassification trends analysis because it threw higher percentages of errors which allowed us to analyze the trends. Refer to figure 38 below. It illustrates the true labels against the predicted labels which were misclassified.

When analyzed closely, we identified the following trends:

- PTSD to Anxiety: This was the most frequently occurring misclassification with 8 counts. This misclassification occurs due to the shared symptoms between PTSD and anxiety.
- PTSD was sometimes misclassified as schizophrenia on 4 counts.
- Anxiety disorder was sometimes misclassified as schizophrenia on 3 counts.
- Bipolar disorder was sometimes misclassified as clinical depression on 3 counts.

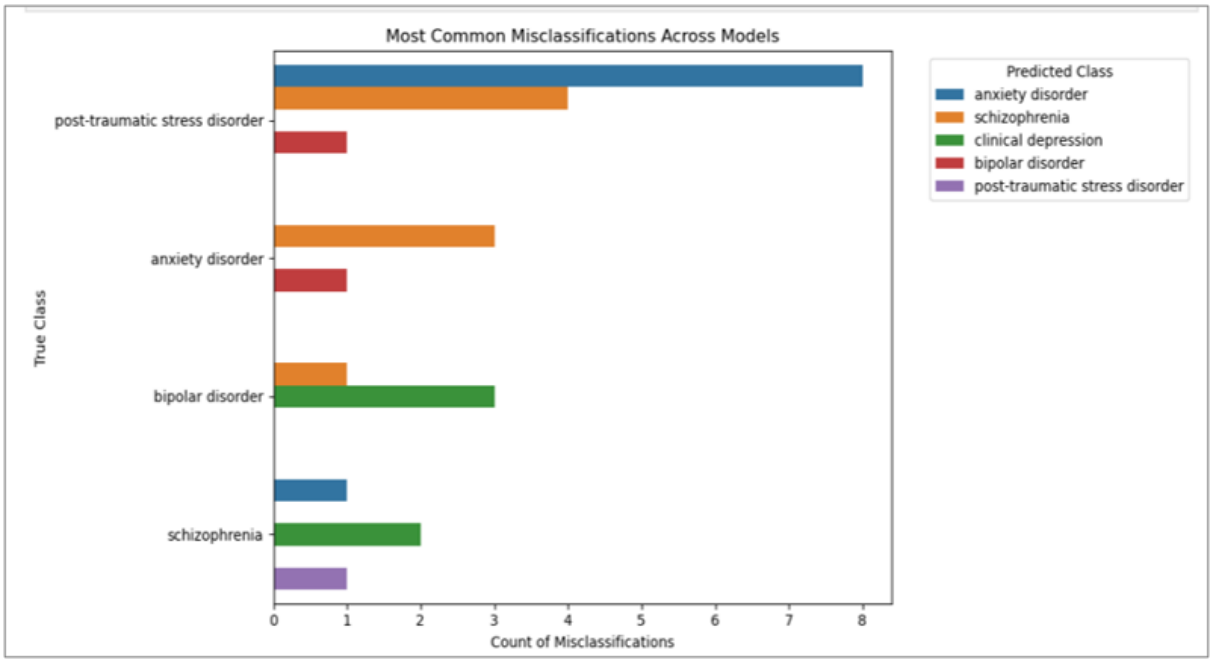


Figure 38

6.0 Graphs and Visualizations

6.1 t-SNE plot for the test dataset

This t-SNE graph provides a general idea of how our abstracts for each category are spread out on the 2D surface.

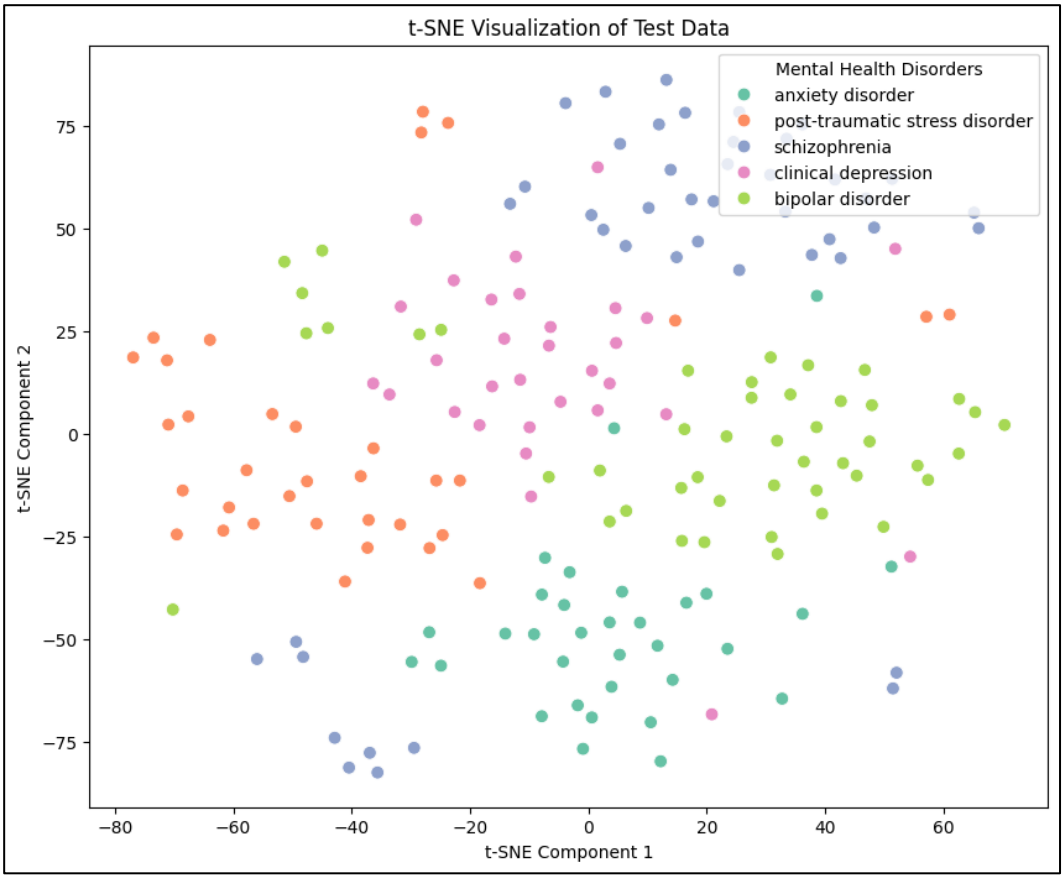


Figure 39

Clusters Indicate Separation: This t-SNE graph shows distinct clusters for different disorders (e.g., separate groups for Anxiety, PTSD, Depression, etc.). It suggests that the model can differentiate between these categories based on their textual features.

Overlap Suggests Confusion: In some areas, different disorders overlap slightly. It means that the abstracts of different disorders share similar language, making it difficult for the classifier to distinguish them, possibly causing errors. A possible cause is:

- The disorders may have semantic similarities (e.g., Anxiety and PTSD might use similar medical terms). This might indicate that our model may have learned meaningful patterns.

6.2 Confusion Matrices for each model

To identify and analyze the errors, we plot confusion matrices for all models as shown in figure 40-42 which is presented as a heatmap. The confusion matrix shows correct and incorrect predictions for each category. The diagonal values are correct predictions and the others are misclassifications.

This is discussed in detail in section 5.1 above. In addition to these visualizations and graphs, we provide visualizations in other sections which are applicable to this point.

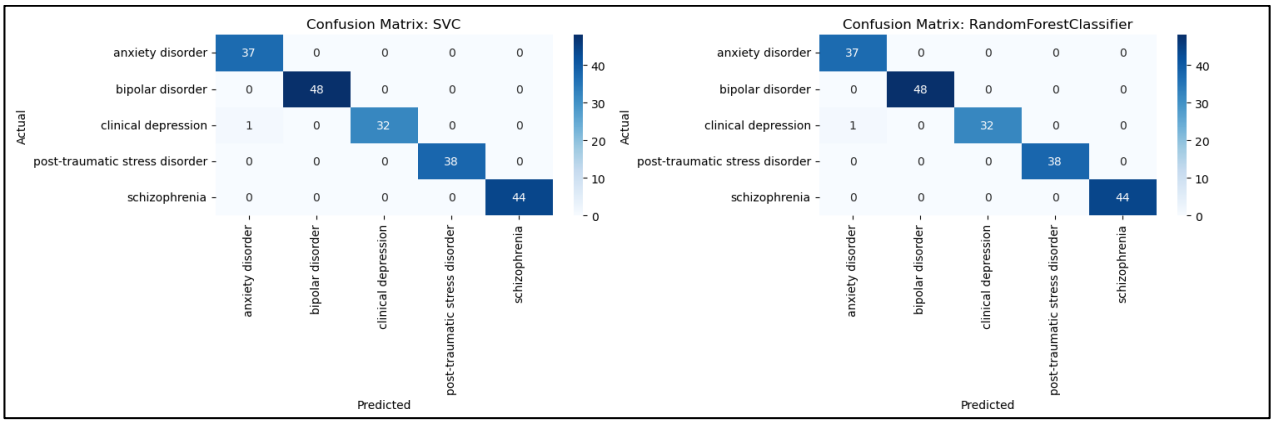


Figure 40

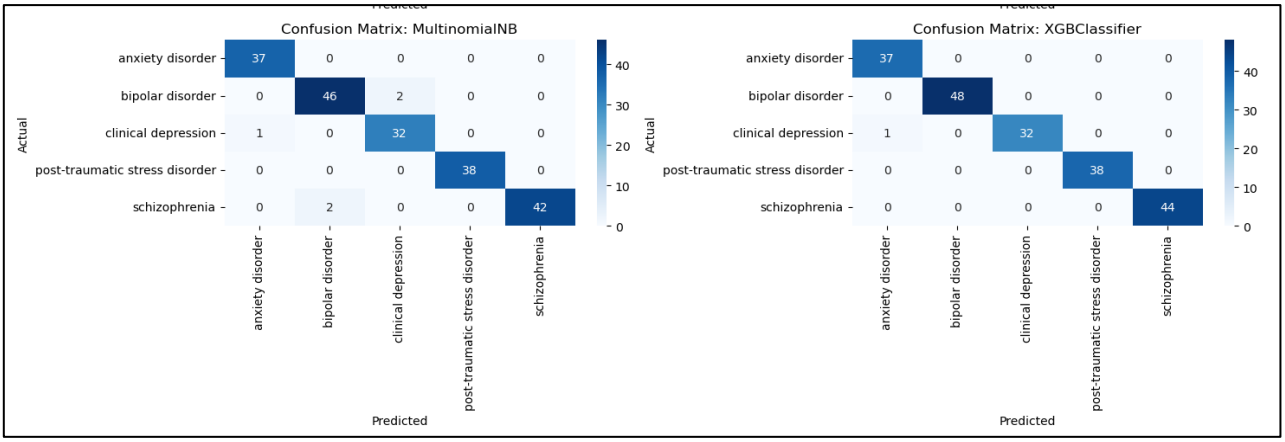


Figure 41

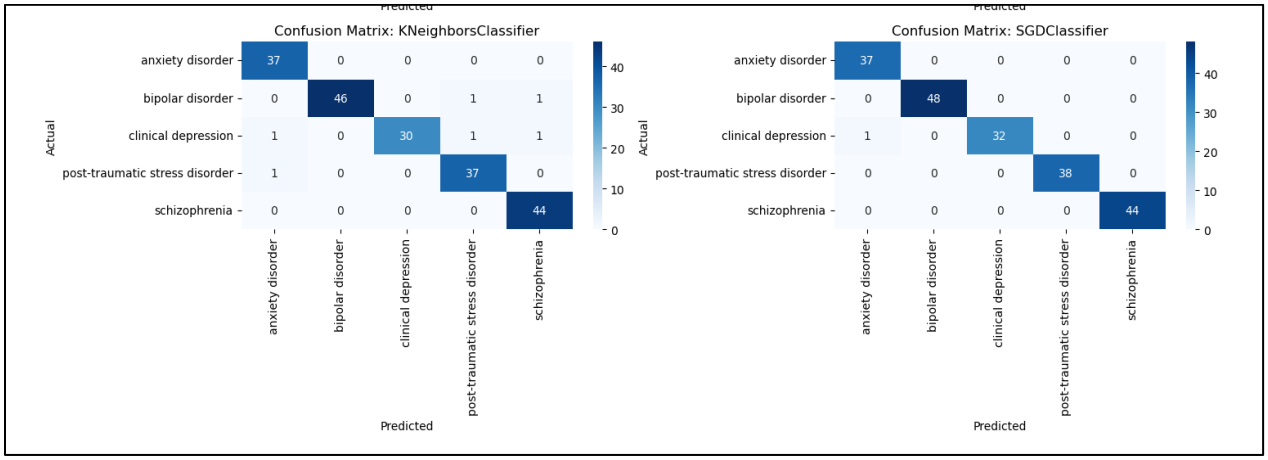


Figure 42

7.0 Analysis of Bias and Variability

7.1 Bias and Variance Graphs

Here, for the bias and variance variability analysis, the error on testing and training dataset has been calculated. For each model, we train it on the first N samples and calculate the error on both the training and test datasets.

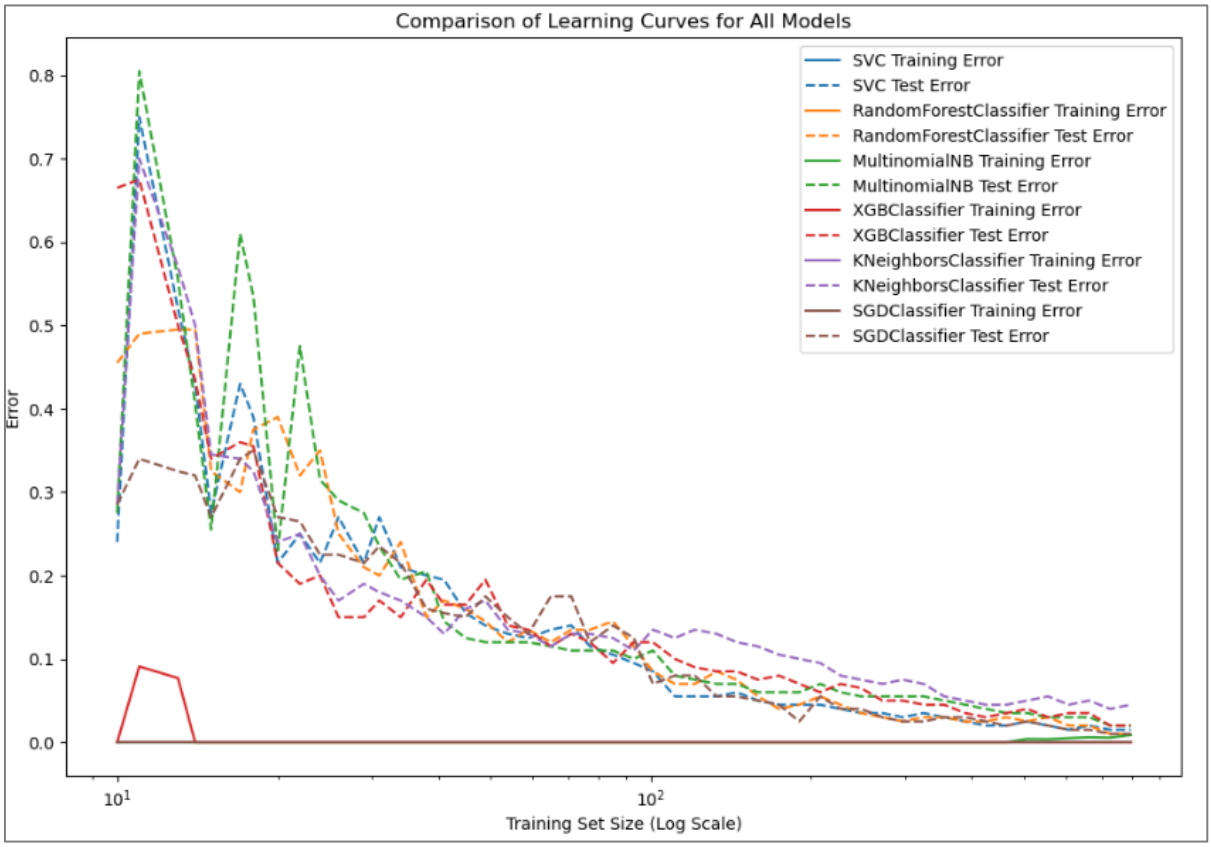


Figure 43

After calculating the errors, we plot both the errors in the proportion of the size of the training dataset size. If the model would have the high bias, both the training and test data set errors will be high and similar or else if the model would have high variance, the training error will be low, but the test error will be high.

We can also conclude if the model is underfitted or overfitted based on the bias and variance distribution. If the learning curve shows that the training and test errors are high and very close to each other. This basically indicated the model is underfitting. Moreover, if the model shows low training error but high-test error, it concludes that the model has high variance that shows that model is overfitted. It means model fits the training data very well, but it is not able to generalize to the new data.

Insights:

As we can see, all our models have Low Bias and Low Variance, since both training and test errors eventually stabilize after certain amount of the training data at low values, this shows that the model is well-calibrated. It generalizes the new data very effectively.

The test errors we have are relatively stable even when we give more training data, this shows that the model is not overfitting and generalizes the data without getting the complex model.

Based on these graphs, we can say that all the models have achieved a good balance between Bias and Variance. Our models seem to have Low Bias and Low Variance on both the datasets (figure 44-46).

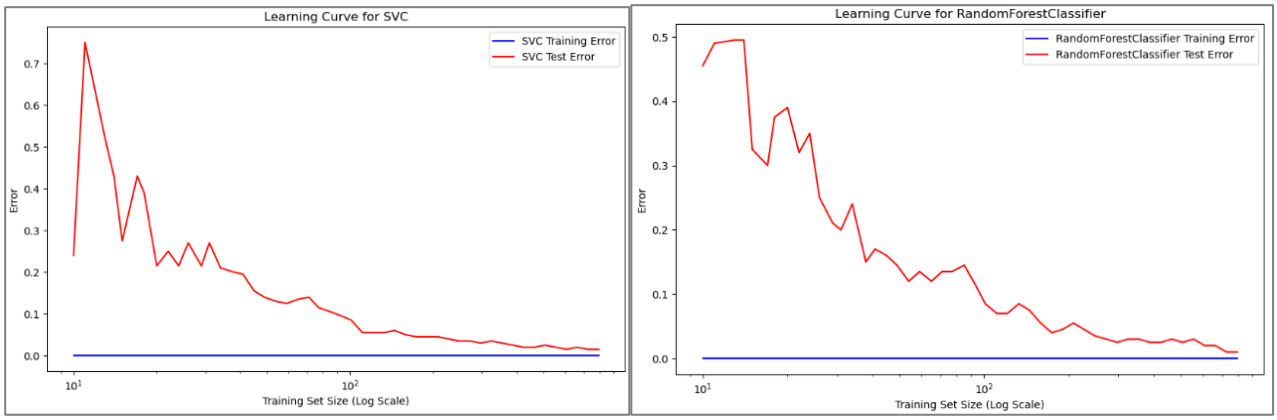


Figure 44

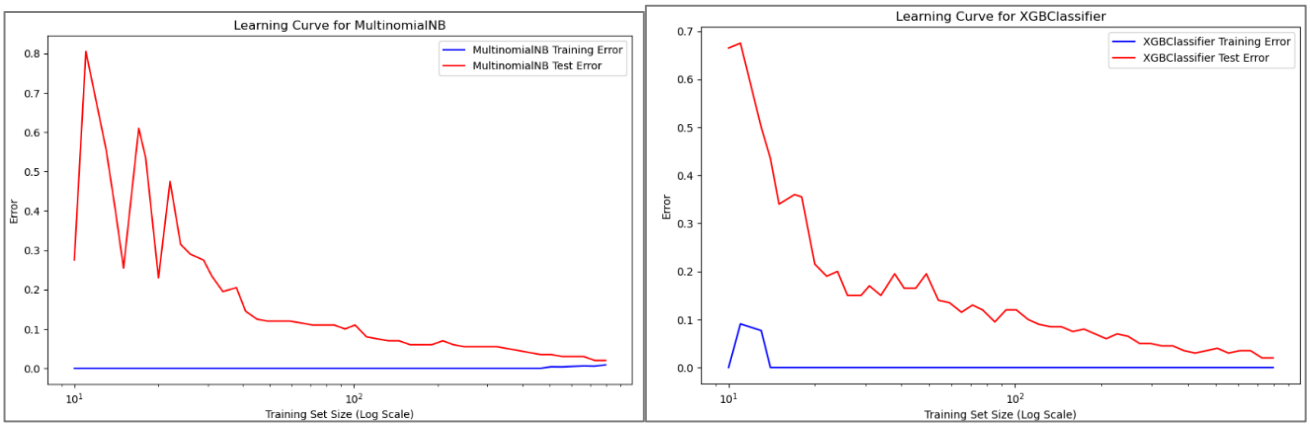


Figure 45

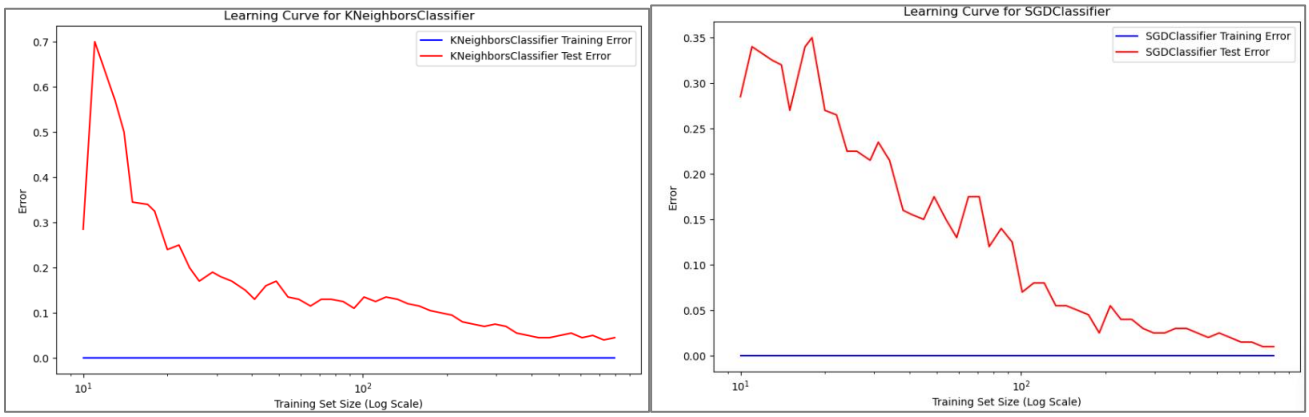


Figure 46

8.0 Identifying, Measuring and Controlling the Machine's thresholds of factors of prediction hardship

8.1 Using two datasets with varying words per partition

We used two datasets to understand which dataset performed well, one with 100 words per partition (default) and the other with 75 words per partition. When we executed the models on both, the dataset with 100 words per partition performed better with higher accuracy and F1-scores. Hence, in all parts of this report we have used the results of the dataset which is 100 words partition. As evidence, following are the two performance evaluation results from the two datasets:

Performance Summary for 75 words per partition:

```
Performance Summary:
                      Model  Accuracy  Macro F1-Score
0                       SVC     0.985        0.984745
1  RandomForestClassifier     0.990        0.990149
2           MultinomialNB     0.975        0.974750
3           XGBClassifier     0.980        0.980378
4    KNeighborsClassifier     0.955        0.953336
5           SGDClassifier     0.990        0.989681
```

Figure 47

Performance Summary for 100 words per partition:

```
Performance Summary:
                      Model  Accuracy  Macro F1-Score
0                       SVC     0.995        0.994256
1  RandomForestClassifier     0.995        0.994256
2           MultinomialNB     0.975        0.975394
3           XGBClassifier     0.995        0.994256
4    KNeighborsClassifier     0.970        0.968721
5           SGDClassifier     0.995        0.994256
```

Figure 48

Insight: As a result, we recommend research scholars to use more words per partition from abstracts to obtain a higher accuracy in their classification output.

8.2 Hyperparameter tuning using GridSearchCV

We used GridSearchCV to test multiple hyperparameter combinations to find which configurations yield poor performance. We used this technique to identify weak spots in model performance, such as underfitting (too simple) or overfitting (too complex). Following is how it applies in the context of this section.

Identifying Prediction Hardship:

- GridSearchCV ensures feature importance and adjusts algorithm sensitivity. For instance, in our mental health dataset, certain terms or features (e.g., medical terminology vs. general language) may be harder for the model to distinguish. Some classifiers (like Naïve Bayes) might struggle with specific text distributions, while others (like SVM) might be more robust.
- We use GridSearchCV to find the best hyperparameters that reduce misclassification of minority classes. It helps detect whether a model is too simple (underfitting) or too complex (overfitting) specially for the Random Forest model.

- It also identifies which algorithms perform better with TF-IDF or n-gram features.

Measuring Prediction Hardship:

- Helps fine-tune decision thresholds for models like SVM (C parameter) or Random Forest (max_depth, n_estimators).
- For SVM, GridSearchCV can find the best C and kernel to minimize prediction errors while maintaining model stability, the same applies to other models' parameters.

Controlling the Machine's Thresholds

- GridSearchCV ensures models are neither too sensitive (high variance) nor too rigid (high bias). It also adjusts learning rate, max_depth, n-gram range, feature selection methods, etc.
- For example, in the application to Random Forest, this technique helps control overfitting and ensures the model generalizes well.

Following is the code for the GridSearchCV and its output:

```
[ ] grid_search_best = []

    # Iterate through models and parameters
    for model, params in zip(models, parameters):
        print(f"Running GridSearch for {model.__class__.__name__}...")

        # Perform grid search
        grid_search = GridSearchCV(
            model,
            param_grid=params,
            cv=10,  # 10-fold cross-validation
            scoring='accuracy',
            verbose=2,
            n_jobs=-1
        )

        # Fit the grid search
        grid_search.fit(X_train, y_train_encoded)

        # Append the best model
        grid_search_best.append((model.__class__.__name__, grid_search.best_estimator_))

        # Output results
        print(f"Best parameters for {model.__class__.__name__}: {grid_search.best_params_}")
        print(f"Best cross-validation score for {model.__class__.__name__}: {grid_search.best_score_}\n")
```

Figure 49

```
Running GridSearch for SVC...
Fitting 10 folds for each of 12 candidates, totalling 120 fits
Best parameters for SVC: {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
Best cross-validation score for SVC: 0.97875

Running GridSearch for RandomForestClassifier...
Fitting 10 folds for each of 27 candidates, totalling 270 fits
Best parameters for RandomForestClassifier: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Best cross-validation score for RandomForestClassifier: 0.9775

Running GridSearch for MultinomialNB...
Fitting 10 folds for each of 3 candidates, totalling 30 fits
Best parameters for MultinomialNB: {'alpha': 1.0}
Best cross-validation score for MultinomialNB: 0.9737500000000001

Running GridSearch for XGBClassifier...
Fitting 10 folds for each of 27 candidates, totalling 270 fits
Best parameters for XGBClassifier: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100}
Best cross-validation score for XGBClassifier: 0.96875

Running GridSearch for KNeighborsClassifier...
Fitting 10 folds for each of 12 candidates, totalling 120 fits
Best parameters for KNeighborsClassifier: {'metric': 'euclidean', 'n_neighbors': 10, 'weights': 'distance'}
Best cross-validation score for KNeighborsClassifier: 0.96125

Running GridSearch for SGDClassifier...
Fitting 10 folds for each of 54 candidates, totalling 540 fits
Best parameters for SGDClassifier: {'alpha': 0.001, 'loss': 'modified_huber', 'max_iter': 1000, 'penalty': 'l2'}
Best cross-validation score for SGDClassifier: 0.985
```

Figure 50

9.0 Conclusion

In conclusion, this assignment provides a **valuable service to classify** mental health research abstracts in the five chosen categories for research scholars. The service provides the ability to scan a public database such as PubMed to identify research papers in five chosen mental health disorders, extract the abstracts and classify them according to the disorder name.

In this process there are four models which are more accurate and perform higher than other models for the 100 words dataset. These are SVC, Random Forest, XGBoost and SGD. **Random Forest Classifier had the highest accuracy and macro f1-score overall when considering both the 100 word and 75 words abstract and emerged as the champion model.**

While the recommended model has high accuracy with a higher number of true positives, tools built with any of these models are still prone to error. Key insights from our error analysis show that:

Insights (from 100 words per partition dataset):

- **Insight 1:** Some papers focus on two or more mental health disorders in the analyzed categories.
- **Insight 2:** Some mental health disorders have similar impacts and often have similar words such as clinical depression and bipolar disorder have the impact 'depression'.
- **Insight 3:** Some papers analyze the relationship of one mental health disorder to another such as schizophrenia to bipolar disorder.
- **Insight 4:** There are individual words that relate to other disorders in an abstract such as, in clinical depression abstracts, words like 'stress' and 'disorder' were used in PTSD.
- **Insight 5:** There is specialized jargon in the abstracts, which confuses models and misclassifies abstracts.
- **Insight 6:** Generalized words were included in the top features or words generated for prediction, in the Word Clouds for each category of mental health disorders.

Trends (from 75 words per partition dataset):

Certain classes are commonly misclassified for other classes as they have overlapping symptoms. Eg: PTSD -> Anxiety

Hence, research scholars should also be aware of the errors that can occur in the classification process of these tools and use the tools wisely.

Some key recommendations for research scholars from this assignment are:

- **Recommendation 1:** The use of longer abstracts with a higher number of words are favorable for improved classifications with higher accuracy.
- **Recommendation 2:** Be aware of the complexities of research papers involving two or more mental health disorder categories considered in this assignment.
- **Recommendation 3:** Based on the results of this assignment, for smaller datasets we recommend research scholars to use the tools developed using classical models like Random Forest Classifiers, for better performance and output.
- **Recommendation 4:** Outside of this assignment, Deep Learning models like BERT would be more accurate with large volumes of domain specific data, possibly with over 10,000 abstracts

compared to other models. In such cases, it is recommended to use tools embedded with that model for higher accuracy.

--

Groupwork: All team members contributed to the assignment and had an equal share of the workload.

10.0 References and Bibliography

Murel, J., & Kavlakoglu, E. (2024). *What is bag of words? | IBM* [Blog]. IBM Think. https://www.ibm.com/think/topics/bag-of-words