

Interim Project Report

Detection of Depression from Textual Data in Social Media Using Machine Learning

Done by

Malavika P (MAC23MCA-2037)

Under the guidance of Prof. Biju Skaria

Introduction

Depression is one of the leading causes of suicide worldwide. However, a significant percentage of depression cases go undiagnosed and untreated. According to the World Health Organization (WHO), depression is the most prevalent mental disorder, affecting over 300 million people globally. It is also responsible for more than two-thirds of suicides annually.

Recent studies have demonstrated that messages posted by individuals with major depressive disorder on social media platforms can be analyzed to predict their likelihood of suffering from depression. Social media platforms, where people freely share their thoughts and feelings, offer a valuable source for monitoring health issues and trends. For instance, posts on platforms like Twitter and Facebook enable researchers to investigate multiple aspects of mental health. Specifically, studies have shown that tweets from individuals with major depressive disorder can be used to predict future episodes of depression.

A recent survey indicated that an increasing number of people, particularly teenagers and young adults, are turning to social media to express their feelings of depression. However, related work in this domain often relies on specific keywords like 'depression' and 'diagnose' when utilizing the data. In reality, social media users suffering from depression are unlikely to use such words directly due to the social stigma surrounding the condition. As a result, many affected individuals turn to less formal resources, such as social media, to seek support.

This project aims to develop a machine learning model for detecting depression from textual data on social media. By analyzing posts without relying on explicit keywords, the project seeks to identify patterns and signals indicative of depression, providing a tool for early detection and intervention.

Literature Review

Paper 1: Tadesse MM, Lin H, Xu B, Yang L. “Detection of depression-related posts in reddit social media forum.” Ieee Access. 2019 Apr 4

The paper titled "Detection of Depression-Related Posts in Reddit Social Media Forum" by Michael M. Tadesse et al. presents a study on identifying depression-related content on Reddit using natural language processing (NLP) and machine learning techniques. The authors address the growing concern of depression as a major contributor to global disability and a significant cause of suicide. They aim to examine Reddit users' posts to detect factors that may reveal depressive attitudes.

The researchers use natural language processing (NLP) and machine learning techniques to train and test their method. They identify words commonly used by people with depression and analyse their connection to mental health. Methods like N-grams (word combinations), LIWC (emotional and psychological content analysis), and LDA (topic identification) help them understand how language on social media can indicate depression.

The study compares single and combined feature sets using various text classification methods like including Logistic Regression, Support Vector Machine, Random Forest, Adaptive Boosting, and Multilayer Perceptron. Results show that proper feature selection and combination lead to higher performance. The best depression detection performance (91% accuracy, 0.93 F1 score) is achieved by combining LIWC, LDA, and bigram features using a Multilayer Perceptron neural network.

This paper helps improve detecting depression on social media, like Reddit, by combining language analysis and machine learning, which could lead to earlier support for those at risk.

Title	Detection of Depression-Related Posts in Reddit Social Media Forum
Area of Work	Mental health detection in social media using NLP and machine learning.
Dataset	The dataset was built by Inna Pirina et al. and consists of a list of depressed and non-depressed users.
Methodology/Strategy	Data pre-processing, Feature extraction, Text classification using various machine learning algorithms.
Algorithm	LR, SVM, RF, AdaBoost, MLP
Result/Precision	91% accuracy and 0.93 F1 score using LIWC+LDA+bigram features with MLP.
Advantages	Combines multiple NLP techniques for improved performance, Examines both single and combined feature sets
Future Proposal	Examine the link between users' personality and depression-related behaviour on social media, Apply the method to other mental health disorders, Test the approach on non-English language data.

Paper 2: Chiong R, Budhi GS, Dhakal S, Chiong F. "A textual-based featuring approach for depression detection using machine learning classifiers and social media texts." Computers in Biology and Medicine. 2021 Aug 1

The paper "A textual-based featuring approach for depression detection using machine learning classifiers and social media texts" by Raymond Chiong et al. introduces a new way to detect depression by analyzing social media posts with machine learning. The paper focus on the problem of undiagnosed depression and its serious consequences. They propose a method that uses text-based features and different machine learning classifiers to find signs of depression in social media posts, even when specific words like "depression" or "diagnosis" are not used.

The study uses various text preprocessing and feature extraction methods, such as bag-of-words and n-grams. The authors test their approach on multiple datasets from different social media platforms and compare the performance of individual and combined machine learning classifiers. They also tackle the issue of imbalanced datasets with dynamic sampling techniques. The results show that their method effectively detects depression across different social media sources, even when trained on one platform and tested on others. This paper helps advance machine learning, natural language processing, and mental health research by providing a method to detect early signs of depression using social media data.

Title	A textual-based featuring approach for depression detection using machine learning classifiers and social media texts
Area of Work	Depression detection using social media data and machine learning
Dataset	<p>The document references several Kaggle datasets related to depression and suicide analysis:</p> <p>Shen et al.(2017): Depression dataset.</p> <p>Eye BB (2020): Focuses on depression analysis.</p> <p>Tanwar R (2020): Based on the diary of a 17-year-old girl, Victoria, who committed suicide due to depression.</p> <p>Komati N (2020): Includes posts from Reddit communities r/SuicideWatch and r/depression.</p> <p>Virahonda S (2020): Consists of comments related to depression and anxiety.</p>
Methodology/Strategy	Text preprocessing, Feature extraction using bag-of-words and n-grams, Training and testing ML models on Twitter datasets, Testing trained models on non-Twitter depression datasets, Experimenting with sampling techniques for imbalanced data.
Algorithm	<p>Single classifiers: LR, SVM, MLP, DT</p> <p>Ensemble classifiers: RF, AdaBoost, Bagging, Gradient Boosting</p>
Result/Precision	<p>Best single classifier: LR (92.61% accuracy on Eye's dataset)</p> <p>Best ensemble: RF (balanced accuracy for depression/non-depression classes)</p> <p>Models performed well even without depression-specific keywords in training data</p> <p>Generalized well to non-Twitter depression datasets</p>
Advantages	<p>Effective depression detection without relying on specific keywords,</p> <p>Generalizable approach that works across different social media platforms</p>
Future Proposal	Incorporate unsupervised learning techniques, Explore multimodal approaches (text + images/video), Investigate more sophisticated NLP techniques, Develop real-time depression monitoring systems, Conduct longitudinal studies to track depression progression.

Paper 3: Islam MR, Kabir MA, Ahmed A, Kamal AR, Wang H, Ulhaq A. “Depression detection from social network data using machine learning techniques.” Health information science and systems. 2018 Dec

The paper titled "Depression detection from social network data using machine learning techniques" by Islam et al. presents an innovative approach to detect depression through analysis of social media data, specifically Facebook comments. The authors explore how social networks can be used to monitor and improve mental health and uses machine learning to study the language and emotions in Facebook comments to find signs of depression.

The study utilizes a dataset of 7,145 Facebook comments, categorized into depression-indicative and non-depression-indicative groups. The authors extract psycholinguistic features using the Linguistic Inquiry and Word Count (LIWC) software, focusing on emotional processes, temporal processes, and linguistic style. They then apply and compare four machine learning classifiers: Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Trees (DT), and Ensemble methods.

The researchers conduct experiments to evaluate the performance of these classifiers across different feature sets. Their results indicate that the Decision Tree classifier generally outperforms other methods, achieving the highest accuracy in detecting depression-indicative comments. The study also includes a time series analysis, revealing that depression-indicative comments are more prevalent during AM hours.

Overall, this paper contributes to the field of mental health informatics by demonstrating the potential of machine learning techniques in detecting depression through social media data. The findings suggest that automated analysis of social network content could be a valuable tool for early detection and intervention of depression, potentially improving mental health outcomes.

Title	Depression detection from social network data using machine learning techniques
Area of Work	Mental Health, Social Network Analysis
Dataset	Publicly available Facebook data (from bipolar, depression, and anxiety Facebook pages)
Methodology/Strategy	Collected Facebook comments data, Used LIWC software to extract linguistic features, Applied machine learning classifiers
Algorithm	Decision Tree, K-NN, Support Vector Machine, Ensemble Methods
Result/Precision	Decision Tree performed best overall Accuracy between 60-80% for different features
Advantages	Uses real social media data, Examines linguistic, emotional, and temporal features
Future Proposal	Use more types of emotional features, verify techniques on larger datasets, apply more LIWC attributes (used 21 out of 50+)

Summary Table

Paper 1	The paper "Detection of Depression-Related Posts in Reddit Social Media Forum" explores detecting depression on Reddit using natural language processing (NLP) and machine learning. It focuses on analyzing language patterns of depressed users, examining features such as n-grams, LIWC categories, and LDA topic modeling. Various classifiers, including LR, SVM, RF, AdaBoost, and MLP, were tested. The SVM classifier with bigrams achieved 80% accuracy, while combining LIWC, LDA, and bigram features with a multilayer perceptron reached 91% accuracy and a 0.93 F1 score. The study highlights that effective feature selection and integration can enhance depression detection in social media text.
Paper 2	The paper "A Textual-Based Featuring Approach for Depression Detection Using Machine Learning Classifiers and Social Media Texts" by Raymond Chiong et al. presents a method for detecting depression through social media posts without relying on specific keywords. The approach uses textual features extracted via bag-of-words, and n-grams, and evaluates various classifiers, including LR, SVM, MLP, and ensembles. Experiments on Twitter and non-Twitter datasets demonstrate that the method effectively identifies depression with over 90% accuracy. Notably, LR and RF performed best, and dynamic under sampling improved results on imbalanced datasets. The study shows that generalized machine learning techniques can detect depression in social media texts, regardless of keyword usage.
Paper 3	The paper explores detecting depression from Facebook comments using machine learning techniques. It synthesizes emotion detection literature and identifies four feature types: emotional process, temporal process, linguistic style, and a combined set. Experiments were conducted using LIWC software to extract psycholinguistic features, and various classifiers, including Decision Trees, SVM, KNN, and Ensemble methods, were tested. Decision Trees performed best, with the highest recall and F-measure, achieving 60-80% accuracy in detecting depression-indicative comments. The study also includes time series analysis to track depression patterns over different times and months.

Project Proposal

Based on the analysis of the above 3 papers, it is evident that depression is a widespread issue in contemporary society, with social media emerging as an informal platform for individuals to express their feelings. Many individuals suffering from depression tend to express their emotions through social media posts and comments.

In response to this, my proposed system aims to detect depression from textual data posted on social media using machine learning classifiers and feature extraction methods. I have chosen to employ N-gram, and TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique methods to extract features for detecting depression in text, regardless of specific keywords such as 'depressed' or 'not depressed'. The system synthesizes emotion detection literature and identifies 4 types of features: emotional process, linguistic style, temporal process and a combined set.

The project intends to utilize Multi-Layer Perceptron (MLP) and Random Forest (RF) for ensemble methods, as well as Decision Trees (DT) to achieve the most accurate results. Additionally, the system will include a feature that suggests remedies for users identified as having depression.

The expected output of the system is to analyse text or any social media comment or post. If the post indicates any tone of sadness or contains depressive content, the model will determine that the user is experiencing depression, and vice versa. Furthermore, the system will provide suggested remedies for users identified as depressed.

To facilitate user interaction, I plan to implement a web interface using Flask. This interface will allow users to input text to be evaluated for depression. The output will indicate whether the text suggests the user is experiencing depression, and if so, will provide suggested remedies in an additional text field.

By utilizing advanced machine learning techniques and comprehensive feature extraction methods, the system aims to provide an accurate and practical tool for detecting depression in social media text and offering supportive suggestions.

System Analysis

Analysis of the Dataset

About the Dataset

The source of the Dataset is from Kaggle.com

LINK: <https://www.kaggle.com/datasets/bababullseye/depression-analysis>

Kaggle is a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish datasets, explore and build models in a web-based data-science environment.

Depression analysis dataset by BB eye is a valuable resource for depression analysis in social media textual data. It consists of 10314 records and is structured into three columns: the text of the post or comment, a unique identifier (id), and a class label. The class label indicates whether a piece of text is depressive or non-depressive, with "0" representing non-depressive content and "1" representing depressive content.

This dataset is particularly useful for training supervised learning models aimed at detecting depression-related content from social media posts. The absence of explicit keywords like "depressed" or "not depressed" presents an interesting challenge for the model, emphasizing the need for more nuanced text classification techniques.

Unnamed: 0		message	label
0	106	just had a real good moment. i missssssssss hi...	0
1	217	is reading manga http://plurk.com/p/mzp1e	0
2	220	@comeagainjen http://twitpic.com/2y2lx - http://...	0
3	288	@lapcat Need to send 'em to my accountant tomo...	0
4	540	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0
5	624	so sleepy. good times tonight though	0
6	701	@SilkCharm re: #nbn as someone already said, d...	0

Exploratory Analysis

This code will first read a csv file and display the dataset using `df.head()` function, where dataset contains `df` as dataframe. This can be useful for getting a quick sense of the structure and contents of the data.

```
df = pd.read_csv("/content/drive/MyDrive/depression/send/sentiment_tweets31.csv")
df.head()
```

	Unnamed: 0	message	label
0	106.0	just had a real good moment. i missssssssss hi...	0
1	217.0	is reading manga http://plurk.com/p/mzp1e	0
2	220.0	@comeagainjen http://twitpic.com/2y2lx - http://...	0
3	288.0	@lapcat Need to send 'em to my accountant tomo...	0
4	540.0	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

Data Preprocessing

Data Cleaning

Data cleaning is an important step in the machine learning process, as it involves preparing the data for modelling by identifying and addressing inconsistencies, errors, and missing values in the dataset. The quality of the data can have a significant impact on the accuracy and effectiveness of the machine learning model. The following are some common techniques used in data cleaning in machine learning:

1. Removing duplicates: This involves identifying and removing duplicate records from the dataset.
2. Handling missing values: Missing values can be replaced with appropriate values, such as the mean, median, or mode of the data. Alternatively, the missing values can be removed altogether, but this approach should be used with caution as it can result in a loss of valuable data.
3. Removing irrelevant features: Irrelevant features are variables that do not add value to the model and can reduce accuracy. Feature selection techniques, such as correlation analysis or model-based methods, can identify and remove these unnecessary features.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10314 entries, 0 to 10313
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Unnamed: 0   10314 non-null  int64
1   message      10314 non-null  object
2   label        10314 non-null  int64
dtypes: int64(2), object(1)
memory usage: 241.9+ KB
```

The `info()` method provides a concise summary of a `DataFrame`, including the number of non-null values, the data type of each column, and the memory usage of the `DataFrame`. It also lists the total number of columns, and the number of entries in each column. This shows that the `DataFrame` has 10314 entries and 3 columns. The first column is an unnamed index and the other two columns are labelled "message" and "label". The first column is float data type, last column is integer datatype and second column is of type object (which usually contain strings). The output also shows that there are no null values in any of the columns. Finally, the memory usage of the `DataFrame` is shown to be 241.9+ KB

```
df.duplicated()
```

```
0
0    False
1    False
2    False
3    False
4    False
...
10309  False
10310  False
10311  False
10312  False
10313  False
10314 rows x 1 columns
```

There are no duplicate values in the dataset. So, it returns the dataset with all field values as False. If any field is True, then it returns True for that field.

```

▶ missing=df.isnull().sum()
print(missing)

⇄ message    0
   label    0
   dtype: int64

```

There are no missing values in the dataset.

```

▶ df = df.drop(columns=['Unnamed: 0'])

print(df.head())

⇄

```

	message	label
0	just had a real good moment. i misssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat Need to send 'em to my accountant tomo...	0
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

Dropped Irrelevant feature column Unnamed: 0.

From these our dataset doesn't contain any missing values, duplicates, and irrelevant features.

▼ Preprocessing

```
[ ] tweet=df.message
```

```
[ ] stopwords = nltk.corpus.stopwords.words("english")
```

```

#extending the stopwords to include other words used in twitter such as retweet(rt) etc.
other_exclusions = ["#ff", "ff", "rt"]
stopwords.extend(other_exclusions)
stopwords.remove('not')
stemmer = PorterStemmer()

```

```
def preprocess(tweet):
```

```

    # removal of extra spaces
    regex_pat = re.compile(r'\s+')
    tweet_space = tweet.str.replace(regex_pat, ' ',regex=True)

```

```

    # removal of @name[mention]
    regex_pat = re.compile(r'@[w\~]+')
    tweet_name = tweet_space.str.replace(regex_pat, '',regex=True)

```

```

# removal of links[https://abc.com]
giant_url_regex = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&]|' + '!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
tweets = tweet_name.str.replace(giant_url_regex, '', regex=True)

# removal of punctuations and numbers
punc_remove = tweets.str.replace("[^a-zA-Z]", " ", regex=True)
# remove whitespace with a single space
newtweet=punc_remove.str.replace(r'\s+', ' ')
# remove leading and trailing whitespace
newtweet=newtweet.str.replace(r'^\s+|\s+$','')
# replace normal numbers with numbr
newtweet=newtweet.str.replace(r'\d+(\.\d+)?','numbr')
# removal of capitalization
tweet_lower = newtweet.str.lower()

# tokenizing
tokenized_tweet = tweet_lower.apply(lambda x: x.split())

# stemming of the tweets
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x])

```

```

# removal of stopwords
tokenized_tweet= tokenized_tweet.apply(lambda x: [item for item in x if item not in stopwords])

for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])
    tweets_p= tokenized_tweet

return tweets_p

processed_tweets = preprocess(tweet)

df['processed_messages'] = processed_tweets
print(df[["message","processed_messages"]].head(10))

```

The preprocessing tasks aim to clean and standardize the Dataset, making it more suitable for subsequent analysis or modelling. The provided code successfully performs these tasks, and the processed messages can be used for various natural language processing tasks, sentiment analysis, or machine learning applications.

NLTK is a powerful library for natural language processing in Python. It provides various tools and resources for tasks such as tokenization, stemming, and stopwords removal. In this project, NLTK is employed to handle English stopwords and perform stemming. The re library is used for pattern matching and manipulation of strings. It plays a crucial role in the removal of mentions, hyperlinks, and other specific patterns from Twitter data.

- **Removal of Extra Spaces, Mentions, and Hyperlinks:** Regular expressions are used to remove extra spaces, mentions (starting with '@'), and hyperlinks from the messages.
- **Removal of Punctuation, Numbers, and Capitalization:** Punctuation and numbers are removed, and the messages are converted to lowercase. This step ensures uniformity and reduces noise in the text.
- **Tokenization:** The lowercase, punctuation-stripped tweets are tokenized into lists of words using the `split()` function.
- **Stemming:** The Porter stemming algorithm is applied to each tokenized message, reducing words to their stems.
- **Removal of Stopwords:** Stopwords, including custom Twitter-specific terms, are removed from the tokenized and stemmed tweets.
- **Dataset Update:** The processed messages are added as a new column ('processed_messages') to the original dataset.

```

                                message \
0  just had a real good moment. i misssssssss hi...
1      is reading manga http://plurk.com/p/mzp1e
2  @comeagainjen http://twitpic.com/2y2lx - http:...
3  @lapcat Need to send 'em to my accountant tomo...
4      ADD ME ON MYSPACE!!! myspace.com/LookThunder
5      so sleepy. good times tonight though
6  @SilkCharm re: #nbn as someone already said, d...
7      23 or 24i;C possible today. Nice
8      nite twitterville workout in the am -ciao
9  @daNanner Night, darlin'! Sweet dreams to you

                                processed_messages
0      real good moment misssssssss much
1      read manga
2
3  need send em account tomorrow oddli even refer...
4      add myspac myspac com lookthund
5      sleepi good time tonight though
6  nbn someon already said doe fiber home mean le...
7      c possibl today nice
8      nite twittervil workout ciao
9      night darlin sweet dream

```

Analysis of Feature Variables

Feature variables are input variables used to predict the class. In this dataset there is only one and the most important feature variable.

- message: This contains the text of the tweet. The text can be transformed into features using techniques like text vectorization (n-gram,TF-IDF)

Analysis of Class Variables

Class variables are target variables that is needed to be predicted.

- label: This appears to be a binary label indicating sentiment, with values such as 0 and 1.

This Dataset has 2 Classes:

1. Non-Depressive: It is denoted as 0.
2. Depressive: It is denoted as 1.

```
[ ] dt_transformed=df[['label','processed_messages']]
    y=dt_transformed.iloc[:, :-1].values
```

```
[ ] print(dt_transformed)
```

	label	processed_messages
0	0	real good moment misssssssss much
1	0	read manga
2	0	need send em account tomorrow oddli even refer...
3	0	add myspac myspac com lookthund
4	0	sleepi good time tonight though
...
10264	1	mani suffer depress sad feel noth persist nag ...
10265	1	depress g herbo mood done stress peopl deserv
10266	1	depress succumb brain make feel like never enough
10267	1	ketamin nasal spray show promis depress suicid...
10268	1	dont mistak bad day depress everyon ha em

```
[ ] print(y)

[[0]
 [0]
 [0]
 ...
 [1]
 [1]
 [1]]
```

In this code, the variable `dt_trasformed` is created as a new `DataFrame` by selecting only the 'label' and 'messages' columns from the original 'dataset'. This means that `dt_trasformed` now contains only the 'label' and 'messages' columns from the original dataset. The next line of code `y = dt_trasformed.iloc[:, :-1].values` selects all rows of the 'label' column of `dt_trasformed` using the `.iloc()` method and stores them as a NumPy array in the variable `y`. The `[:, :-1]` syntax selects all rows and all columns except for the last column, which in this case is the 'message' column. This means that `y` will contain the values of the 'label' column as a NumPy array.

Feature Extraction

TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique is employed to convert a collection of text documents into a numerical representation that is suitable for machine learning tasks. This code utilizes the `TfidfVectorizer` class from `scikit-learn` to perform TF-IDF feature extraction on a dataset of processed messages. Below is an explanation for the TF-IDF feature extraction using the provided code.

1. The `TfidfVectorizer` is configured with the following parameters:
`ngram_range = (1, 2)` : Considers both unigrams and bigrams in the text, capturing single words and two-word combinations.

`max_df = 0.75`: Ignores terms that appear in more than 75% of the documents, helping filter out common and less informative words.

`min_df = 5`: Ignores terms that appear in fewer than 5 documents, eliminating rare words that may not contribute significantly.

`max_features = 10000`: Limits the vocabulary size to the top 10,000 features, controlling the dimensionality of the resulting TF-IDF matrix.

2. TF-IDF Feature Matrix:

The `fit_transform` method is applied to the 'processed_messages' column of the dataset, transforming the text data into a TF-IDF feature matrix. The resulting `tfidf` variable represents a sparse matrix where each row corresponds to a message, and each column corresponds to a unique word or word combination in the vocabulary.

3. Conversion to Pandas DataFrame:


The TF-IDF sparse matrix is converted to a Pandas DataFrame (`p`) using `DataFrame(tfidf)`. This DataFrame provides a more structured and user-friendly representation of the TFIDF features, allowing for easier inspection and analysis.

4. Preview of the TF-IDF DataFrame:

The `head()` method is used to display the first few rows of the TF-IDF DataFrame (`p`), providing an insight into the numerical representation of the processed messages.

```
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_df=0.75, min_df=5, max_features=10000)

# TF-IDF feature matrix
tfidf = tfidf_vectorizer.fit_transform(df['processed_messages'])
tfidf
```

 <13724x3995 sparse matrix of type '<class 'numpy.float64'>' with 104478 stored elements in Compressed Sparse Row format>

```
[ ] from pandas.core.frame import DataFrame
    p=DataFrame(tfidf)
```

```
[ ] p.head()
```



0

0 (0, 1410)\t0.6263910020151383\n (0, 2244)\t...

1 (0, 2774)\t1.0

2 (0, 3486)\t0.2576860173264403\n (0, 1088)\t...

3 (0, 561)\t0.27699759394237317\n (0, 2263)\t...

4 (0, 1421)\t0.505396925271364\n (0, 3556)\t0...

Data Visualisation

Data visualization serves as a fundamental technique in data analysis, offering a graphical representation of information through charts, graphs, and maps. Its significance lies in transforming complex datasets into visually intuitive formats, allowing for the identification of trends, patterns, and outliers. This visual representation enhances data exploration, enabling easy comparison between data points and facilitating effective communication of insights. The diverse types of visualizations, such as charts for trends, tables for structured data, graphs for relationships, maps for geographic data, and dashboards for comprehensive displays, cater to various analytical needs. Utilizing tools like Tableau, Excel, or programming libraries like Matplotlib, data visualization aids in making data driven decisions by providing accessible insights into massive amounts of information.

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Define the class labels based on your dataset
class_labels = ['0', '1']

# Count the occurrences of each class
class_counts = df['label'].value_counts()

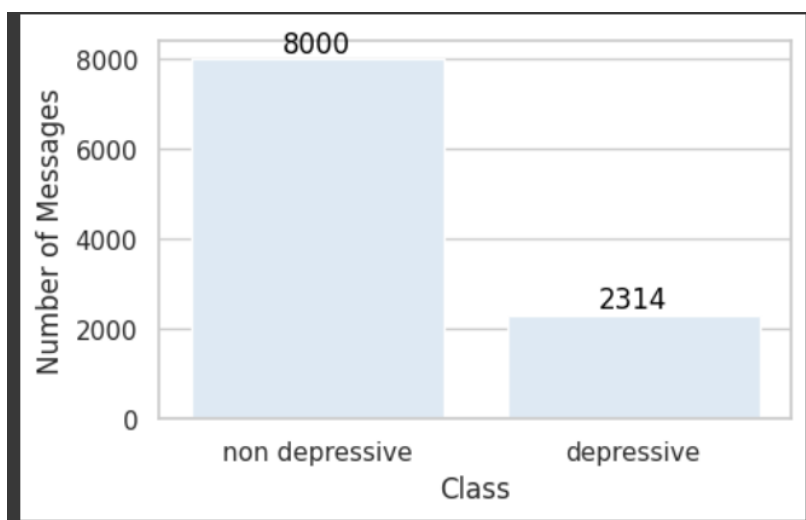
# Set the style and palette for the plot
sns.set(style="whitegrid")
sns.set_palette("Blues")

# Create the plot with a specified size
fig, ax = plt.subplots(figsize=(5, 3))
sns.barplot(x=class_labels, y=class_counts, ax=ax)

# Annotate the bars with counts
for bar, count in zip(ax.patches, class_counts):
    ax.annotate(f'{count}', (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                ha='center', va='bottom', color='black', fontsize=12)

# Set the labels for the axes
ax.set_xlabel('Class', fontsize=12)
ax.set_ylabel('Number of Messages', fontsize=12)

# Display the plot
plt.show()
```



Analysis of Algorithms

Here we use 3 algorithms (Multilayer Perceptron, Decision Tree and Random Forest) for comparison of accuracy and uses an algorithm with high accuracy for classification.

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Working of Algorithm:

Random Forest works in two phases. first is to create the random forest by combining N decision tree, and the second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step 1: Select random K data points from the training set.

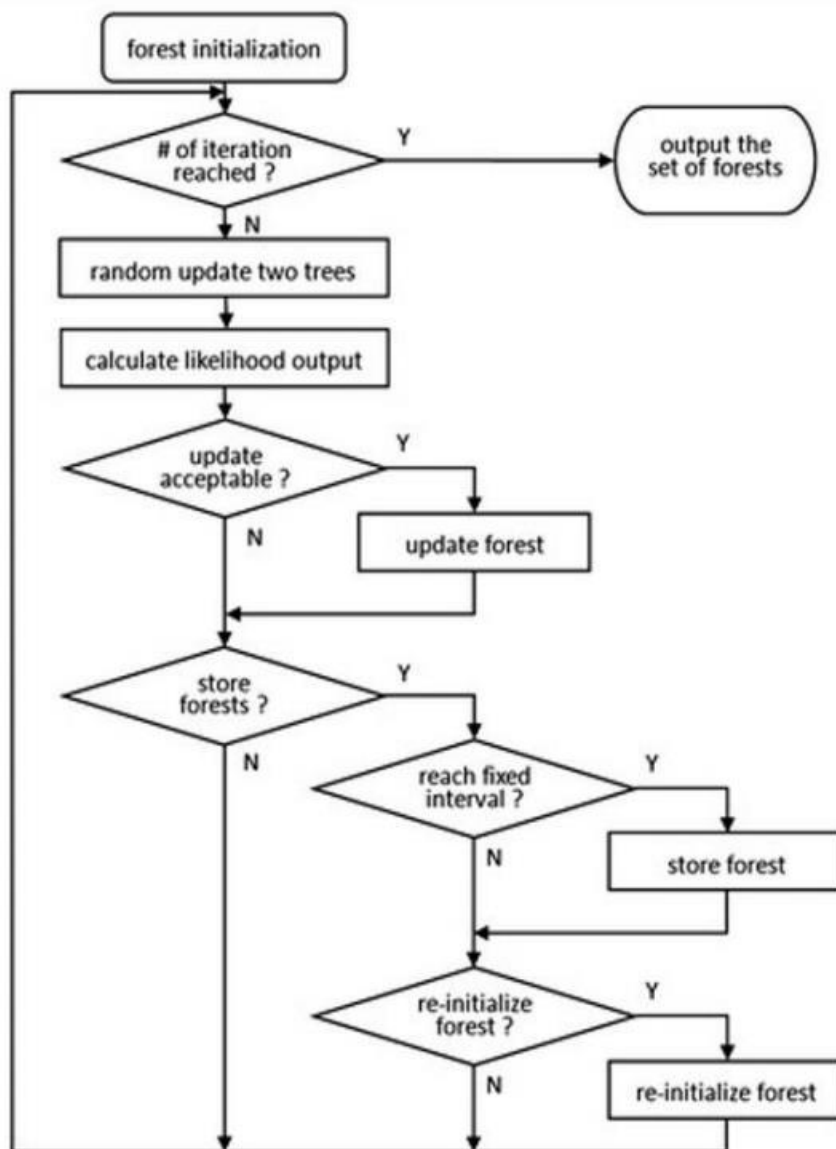
Step 2: Build the decision trees associated with the selected data points (Subsets).

Step 3: Choose the number N for decision trees that you want to build.

Step 4: Repeat Steps 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Activity Diagram of Random Forest:



Decision Tree

A Decision tree is a type of supervised learning algorithm that is commonly used in machine learning to model and predict outcomes based on input data. It is a tree-like structure where each internal node tests on attribute, each branch corresponds to attribute value and each leaf node represents the final decision or prediction. The decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.

Working of the Algorithm:

Step 1: Start with the entire dataset at the root node.

Step 2: Select the best feature to split the data at the root node.

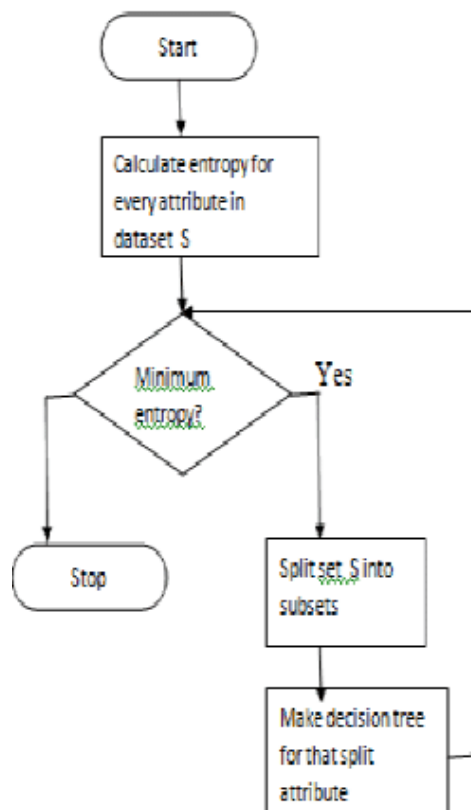
Step 3: Split the dataset into subsets based on the chosen feature and split point.

Step 4: Repeat Steps 2 & 3 for each subset to build the tree.

Step 5: Assign a final prediction to each leaf node.

Step 6: For new data points, pass them down the tree to reach a leaf node and output the prediction.

Activity Diagram of Decision Tree:



Multilayer Perceptron

A Multilayer Perceptron (MLP) is a type of artificial neural network used for supervised learning tasks, such as classification and regression. It consists of an input layer, one or more hidden layers, and an output layer, where each layer is made up of neurons. These neurons are fully connected between layers and use activation functions like ReLU or Sigmoid to learn complex, non-linear relationships in the data. The network is trained using backpropagation, where the weights are adjusted iteratively to minimize the error between predicted and actual outputs. MLPs are versatile and can be applied to a wide range of problems.

Working of the Algorithm:

Step 1: Input Layer

- The input features are fed into the input layer, with each feature represented by a neuron.

Step 2: Forward Propagation

- **Hidden Layers:** Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to the next layer.
- This process repeats through all hidden layers.

Step 3: Output Layer

- The final layer computes the output, applying an activation function if needed (e.g., Softmax for classification).

Step 4: Loss Calculation

- The difference between the predicted and actual output is calculated using a loss function.

Step 5: Backpropagation

- The error is backpropagated to update the weights using gradient descent.

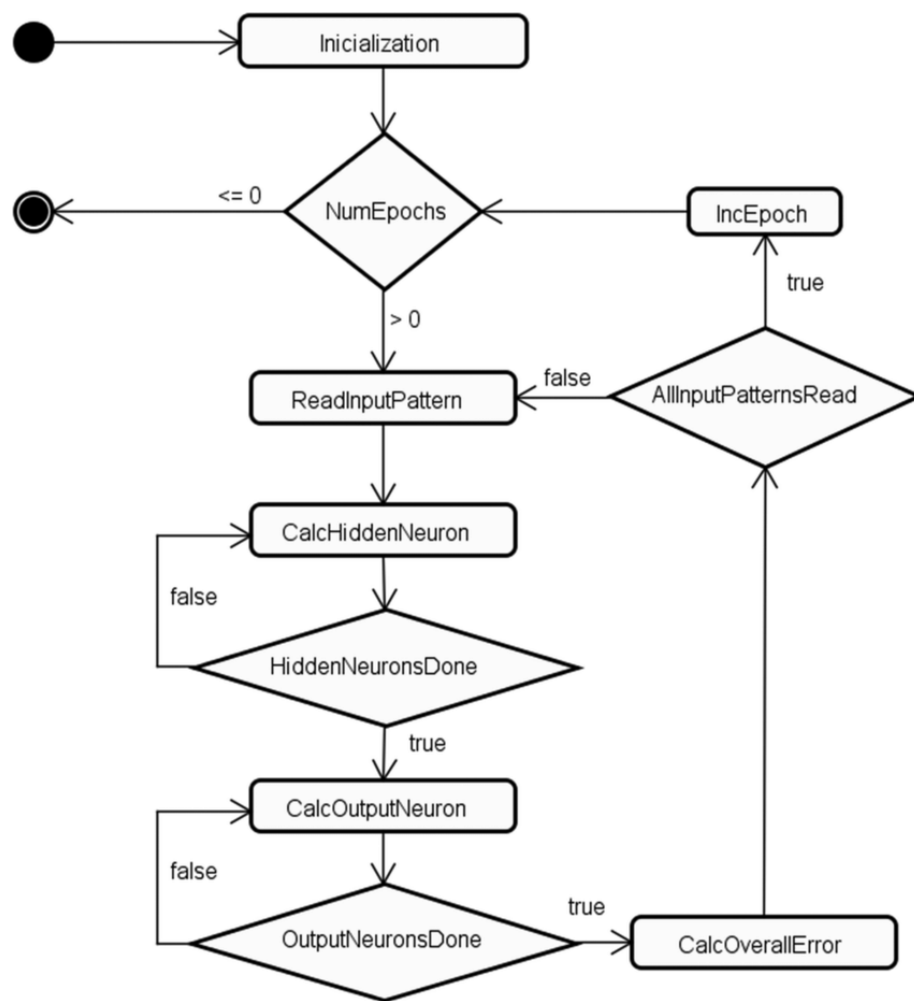
Step 6: Training

- Repeat forward propagation, loss calculation, and backpropagation for several iterations to minimize the loss.

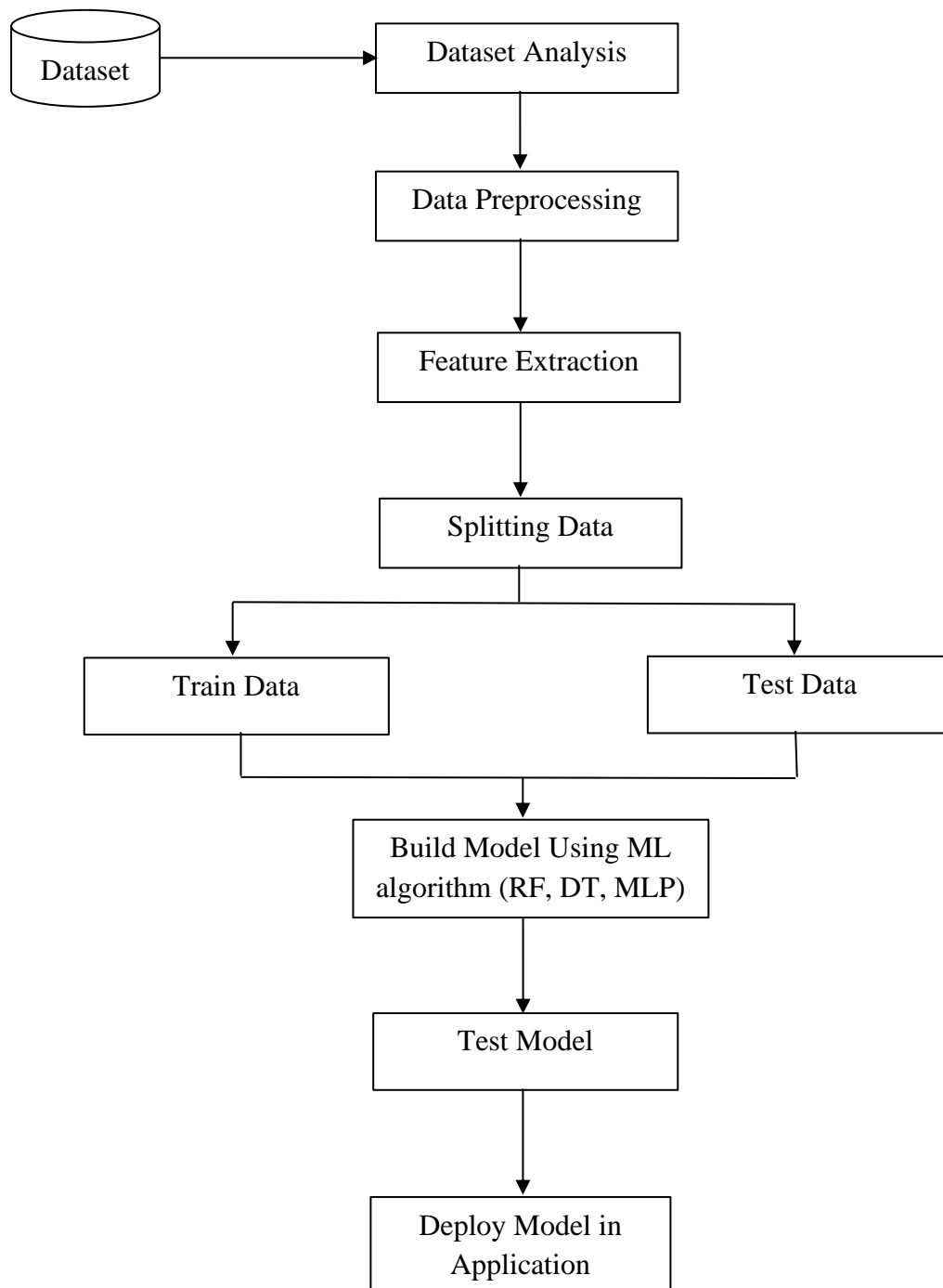
Step 7: Prediction

- After training, the MLP makes predictions on new data using forward propagation.

Activity Diagram of MLP:



Project Pipeline



Project Timeline

- Project proposal approval - 26.07.2024
- Presenting project proposal before the Approval Committee - 29.07.2024 & 30.07.2024
- Initial report submission - 12.08.2024
- Analysis and design report submission - 16.08.2024
- First project presentation - 21.08.2024 & 23.08.2024
- Sprint Release I -Data Preprocessing and Feature Extraction- 30.08.2024
- Sprint Release II -Model Building using MLP, DT and RF- 01.10.2024
- Interim project presentation – 07.10.2024 & 08.10.2024
- Sprint Release III -Interface Design and Deployment- 18.10.2024
- Submission of the project report to the guide - 28.10.2024
- Final project presentation - 28.10.2024 & 29.10.2024
- Submission of project report after corrections - 01.11.2024

System Design

Model Building

Model Planning

1. Objective:

Develop a system to detect depression from textual data on social media using machine learning techniques. The goal is to analyse posts to identify patterns indicative of depression, without relying on explicit keywords such as 'depressed.' This tool aims to assist in early detection of depressive conditions and provide support to those at risk.

2. Approach:

Conduct a comparative study of three machine learning algorithms: Multilayer Perceptron (MLP), Random Forest (RF), and Decision Tree (DT). Evaluate these models based on their accuracy, precision, and recall in detecting depression from social media text data. Feature extraction methods such as N-grams and TF-IDF will be employed to convert textual data into machine-readable format.

3. Data Preparation:

Utilize a dataset from Kaggle consisting of posts labelled as depressive or non-depressive. Preprocess the dataset by handling missing values, removing duplicates, and performing text cleaning operations such as tokenization, stemming, and stopword removal. Split the dataset into training and testing sets for evaluation.

4. Exploratory Data Analysis (EDA):

Perform EDA to understand the distribution of depressive and non-depressive posts in the dataset. Analyse features such as text length, frequent terms, and emotional tone to identify patterns that could aid in the classification of depressive posts.

5. Model Building:

Implement the machine learning models: Multilayer Perceptron (MLP), Random Forest (RF), and Decision Tree (DT). Use the pre-processed text data with features extracted through TF-IDF and N-grams to train the models. Evaluate their performance on the test data using metrics like accuracy, confusion matrix, and classification reports.

6. Model Comparison:

Compare the performance of the three algorithms based on accuracy, F1 score, and speed. The model that best generalizes to unseen data will be selected for deployment, and suggestions for remedies will be provided for users identified as experiencing depression.

Model Training

The dataset was divided into two parts. X representing the input features, and y representing the target variable. The training set consists of 80% of the data and is used to train the model.

MLP Model:

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize MLP Classifier
clf_mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42)
clf_mlp.fit(X_train, y_train)

# Predict on test data
y_pred_mlp = clf_mlp.predict(X_test)

# Evaluate
print(clf_mlp.score(X_train, y_train))
```

The model consists of 100 neurons in the hidden layer. The maximum number of iterations is set to 300 for training, and the random_state is set to 42 to ensure reproducibility. The classifier is fit to the training data (X_train, y_train), and predictions are made on the test data (X_test).

```
0.9928044448492577
```

The model achieved an accuracy of 99.28% on the training data.

Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

# Initialize Decision Tree Classifier
clf_tree = DecisionTreeClassifier(random_state=42)
clf_tree.fit(X_train, y_train)

# Predict on test data
y_pred_tree = clf_tree.predict(X_test)

# Evaluate
print(clf_tree.score(X_train, y_train))
```

The model is initialized with a set `random_state` of 42, which ensures that the results are reproducible by controlling the randomness in the training process. The classifier is fit to the training data (`X_train`, `y_train`), and predictions are made on the test data (`X_test`).

```
0.9948993533108662
```

The model achieved an accuracy of 99.48% on the training data.

Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Initialize Random Forest Classifier
clf_rf = RandomForestClassifier(max_depth= None, n_estimators= 300, random_state=42)
clf_rf.fit(X_train, y_train)

# Predict on test data
y_pred_rf = clf_rf.predict(X_test)

# Evaluate
print(clf_rf.score(X_train, y_train))
```

A Random Forest Classifier, which is an ensemble learning method based on multiple decision trees is used to train the model. The model consists of 300 decision trees. The maximum depth of each tree is none and `random_state` is set to 42 to ensure reproducibility. The classifier is fit to the training data (`X_train`, `y_train`), and predictions are made on the test data (`X_test`).

```
0.9948993533108662
```

The model achieved an accuracy of 99.48% on the training data.

Model Testing

MLP Model:

```
# Predict on test data
y_pred_mlp = clf_mlp.predict(X_test)

# Evaluate
print(clf_mlp.score(X_train, y_train))
print("MLP Accuracy:", accuracy_score(y_test, y_pred_mlp))
print(classification_report(y_test, y_pred_mlp))
```

MLP Accuracy: 0.8852459016393442					
	precision	recall	f1-score	support	
0	0.91	0.89	0.90	1591	
1	0.85	0.88	0.87	1154	
accuracy			0.89	2745	
macro avg	0.88	0.88	0.88	2745	
weighted avg	0.89	0.89	0.89	2745	

MLP acquired a test accuracy of 88.52%.

Decision Tree:

```
# Predict on test data
y_pred_tree = clf_tree.predict(X_test)

# Evaluate
print(clf_tree.score(X_train, y_train))
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_tree))
print(classification_report(y_test, y_pred_tree))
```

Decision Tree Accuracy: 0.8681238615664845					
	precision	recall	f1-score	support	
0	0.89	0.88	0.89	1591	
1	0.84	0.85	0.84	1154	
accuracy			0.87	2745	
macro avg	0.86	0.87	0.87	2745	
weighted avg	0.87	0.87	0.87	2745	

Decision Tree acquired a test accuracy of 86.81%

Random Forest:

```
# Predict on test data
y_pred_rf = clf_rf.predict(X_test)

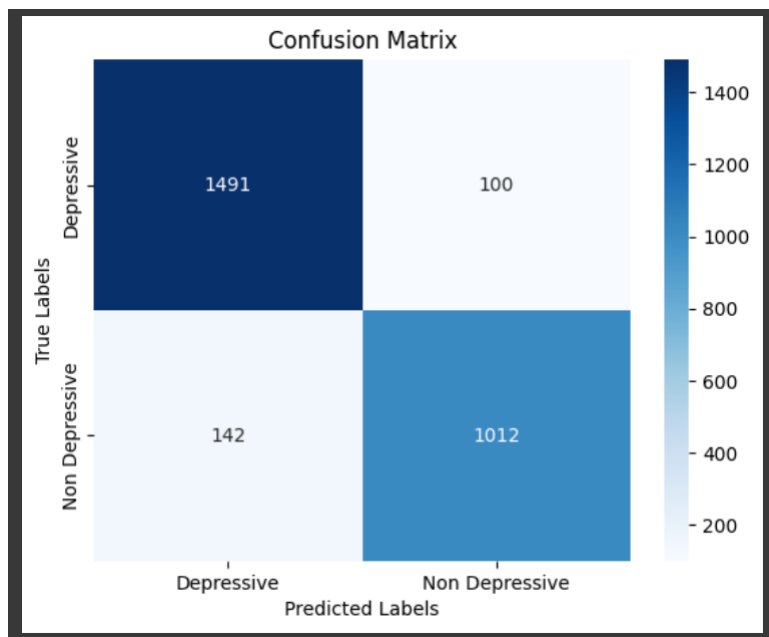
# Evaluate
print(clf_rf.score(X_train, y_train))
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

Random Forest Accuracy: 0.91183970856102					
	precision	recall	f1-score	support	
0	0.91	0.94	0.92	1591	
1	0.91	0.88	0.89	1154	
accuracy			0.91	2745	
macro avg	0.91	0.91	0.91	2745	
weighted avg	0.91	0.91	0.91	2745	

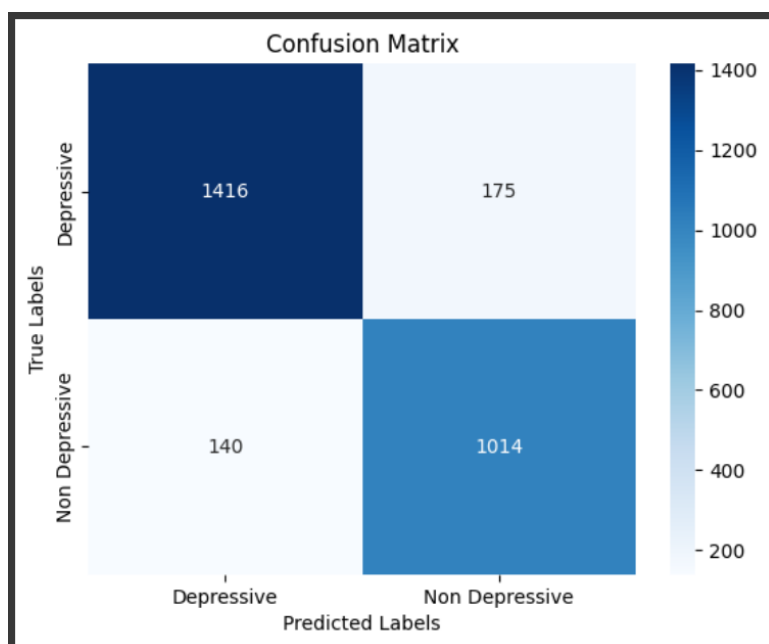
Random Forest acquired a test accuracy of 91.18%.

By Comparison of accuracy, we got to know that Random Forest is the good performing model of highest train and test accuracy.

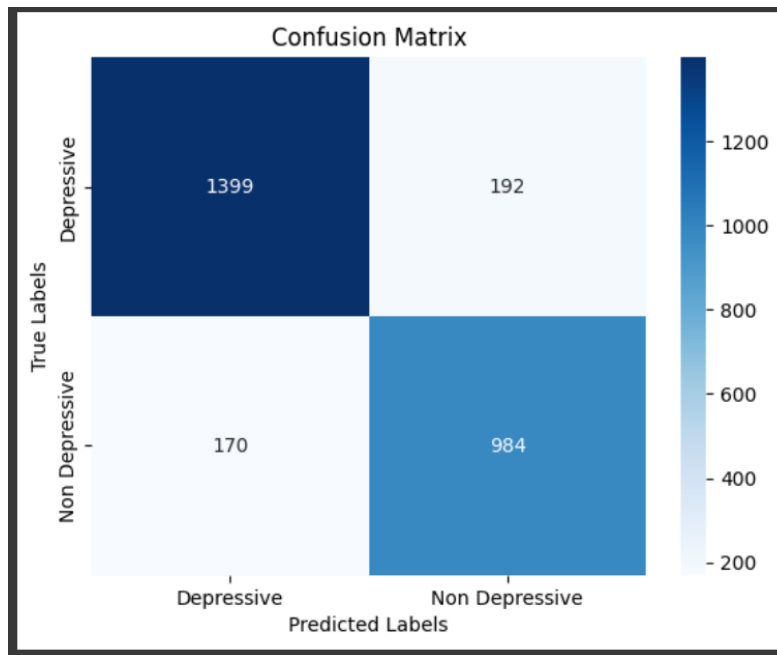
Confusion Matrix of Random Forest:



Confusion Matrix of MLP:



Confusion Matrix of Decision Tree:



Testing with Unseen Input Data:

```
[ ] # Import necessary libraries
    from sklearn.feature_extraction.text import TfidfVectorizer
    from nltk.stem import PorterStemmer
    import re
    import nltk

    df=pd.read_csv("/content/drive/MyDrive/depression/send/preprocessed21.csv")

    processed_messages= df['processed_messages'].astype(str).tolist()

    # Sample input text
    sample ="I've been battling depression for years, and some days it's just overwhelming."

    # Preprocess the input text
    def preprocess_input_text(sample):
        stemmer = PorterStemmer()

        # Define the list of extended stopwords
        extended_stopwords = nltk.corpus.stopwords.words("english")
        other_exclusions = ["#ff", "ff", "rt"]
        extended_stopwords.extend(other_exclusions)

        # Preprocessing steps
        # Step 1: Removal of extra spaces
        tweet_space = re.sub(r'\s+', ' ', sample)
```

```

# Step 2: Removal of @name[mention]
tweet_name = re.sub(r'@\w\-', '', tweet_space)

# Step 3: Removal of links [https://abc.com]
tweet_no_links = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\(\),])?(?:%[0-9a-fA-F][0-9a-fA-F])+', '', tweet_name)

# Step 4: Removal of punctuations and numbers
tweet_no_punctuation = re.sub("[^a-zA-Z]", " ", tweet_no_links)

# Step 5: Remove leading and trailing whitespace
tweet_stripped = tweet_no_punctuation.strip()

# Step 6: Replace normal numbers with 'numbr'
tweet_no_numbers = re.sub(r'\d+(\.\d+)?', 'numbr', tweet_stripped)

# Step 7: Convert to lowercase
tweet_lower = tweet_no_numbers.lower()

# Step 8: Tokenize
tokenized_tweet = tweet_lower.split()

# Step 9: Remove stopwords and stem the tokens
tokenized_tweet = [stemmer.stem(token) for token in tokenized_tweet if token not in extended_stopwords]

```

```

# Step 10: Join the tokens back into a string
processed_input_text = ' '.join(tokenized_tweet)

return processed_input_text

# Preprocess the input text
processed_input_text = preprocess_input_text(sample)

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_df=0.75, min_df=5, max_features=10000)

# Fit the TF-IDF vectorizer on your dataset
tfidf = tfidf_vectorizer.fit_transform(processed_messages) # Assuming processed_messages is your preprocessed dataset

# Transform the processed input text
tfidf_sample = tfidf_vectorizer.transform([processed_input_text])

# Make predictions
predicted_label = clf_rf.predict(tfidf_sample)[0]
# Map class labels to their meanings
class_labels = {
    0: "Non Depressive",
    1: "Depressive"
}

```

```

# Get the label for the predicted class
predicted_label_text = class_labels[predicted_label]

# Print the result
print("Input Text:", sample)
print("Predicted Label:", predicted_label_text)

```

```

Input Text: I'm tired of fighting. It feels like no matter how hard I try, I'm always falling deeper into this pit, and I don't know how much longer I can keep going
Predicted Label: Depressive

```

When an unseen depressive input data is given, it predicts the class label as Depressive.