Mental Health Text Classification Report

Objective:

The primary goal of this project is to build a machine learning system capable of classifying mental health statuses (such as *Anxiety, Depression, Suicidal thoughts*, etc.) based on user-provided textual statements. The system uses **Natural Language Processing (NLP)** and **machine learning classifiers** to achieve this task with a high degree of accuracy.

Dataset Overview:

- **Source:**Sentiment Analysis for Mental Health
- **Type:** Combined Data.csv
- **Total Entries after cleaning:** 52,681
- **Target Variable:** status (mental health status)
- **Classes:**
    - Normal
    - Depression
    - Suicidal
    - Anxiety
    - Bipolar
    - Stress
    - Personality disorder

Initial Statistics:

Feature	Mean	Std Dev	Min	25%	50%	75%	Max
num_of_characters	578.71	846.27	2	80	317	752	32759
num_of_sentences	6.28	10.69	1	1	3	8	1260

Data Cleaning & Preprocessing:

1. **Missing Data Handling:**

- All null values were dropped to ensure clean data.

2. **Text Normalization & Cleaning:**

- Lowercased all statements.
- Removed:
    - URLs

    - Markdown links
    - User mentions (@)
    - Special characters and punctuation

3. **Tokenization & Stemming:**

- Tokenized using NLTK’s word_tokenize()
- Applied stemming using PorterStemmer to create a normalized token column tokens_stemmed.

4. **Feature Engineering:**

- num_of_characters – Length of the statement
- num_of_sentences – Number of sentences using sent_tokenize()

Data Visualization:

Word Clouds:

- Generated word clouds for each mental health class using their tokenized statements.
- Colors randomly assigned from a fixed palette to distinguish topics per class.

Text Vectorization:

- **Method Used:** TF-IDF Vectorizer
- **Parameters:**
    - ngram_range=(1,2)
    - max_features=50000
- Combined TF-IDF features with numerical features (num_of_characters, num_of_sentences) using hstack.

Handling Class Imbalance:

- Applied **RandomOverSampler** from imblearn to balance class distribution in the training set.

Model Building:

Classifiers Used:

1. **Decision Tree**
    - max_depth=9, min_samples_split=5
    - Accuracy: **61.85%**
2. **Logistic Regression**
    - penalty='l1', C=10
    - Accuracy: **76.32%**
3. **XGBoost Classifier**
    - learning_rate=0.2, max_depth=7, n_estimators=500

    - Accuracy: **80.80%** (Best)

Evaluation Metrics:

- Used:
    - Accuracy Score
    - Classification Report (Precision, Recall, F1-score)
    - Confusion Matrix with heatmaps for visual clarity.

XGBoost Summary:

Class	Precision	Recall	F1-score
Anxiety	0.83	0.86	0.85
Bipolar	0.88	0.82	0.85
Depression	0.78	0.73	0.75
Normal	0.92	0.93	0.93
Personality disorder	0.84	0.65	0.74
Stress	0.67	0.76	0.72
Suicidal	0.69	0.73	0.71

Model Comparison Visualization:

A bar plot was generated to compare accuracy scores across classifiers:

Classifier	*Accuracy*
XGBoost	80.80%
Logistic Regression	76.32%
Decision Tree	61.85%

Model Saving:

To enable deployment or future use:

- **XGBoost Model:** Saved in JSON format (xgb_model.json)
- **TF-IDF Vectorizer:** Saved as pickle (tfidf_vectorizer.pkl)
- Label Encoder can also be saved similarly if needed.

Strengths of the Project:

- Comprehensive NLP preprocessing pipeline
- Class imbalance handled effectively
- Ensemble-based model (XGBoost) showed strong performance
- Visual insights through word clouds and confusion matrices
- Includes both text-based and numerical features for better modeling

Deployment with Flask API:

To serve the trained model, a Flask API is developed allowing real-time predictions via POST requests.

Key Features:

- Handles preprocessing (cleaning, stemming).
- Loads saved model (xgb_model.json), vectorizer (tfidf_vectorizer.pkl), and label encoder (label_encoder.pkl).
- Calculates both textual and numeric features.
- Supports CORS for cross-origin requests (ideal for web frontends).
- Provides a /predict endpoint that takes a JSON payload with a "text" field and returns the predicted mental health label.

Code snap:

```python
from flask import Flask, request, jsonify
import joblib
import numpy as np
from xgboost import XGBClassifier
from scipy.sparse import hstack
import re
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer
from flask_cors import CORS
import logging

# Initialize Flask app
app = Flask(__name__)
CORS(app)  # Enable CORS for cross-origin requests

# Setup logging
logging.basicConfig(level=logging.INFO)

# Download required NLTK data
nltk.download('punkt')

# Load models and vectorizers at startup
try:
    xgb_model = XGBClassifier()
    xgb_model.load_model("xgb_model.json")
    vectorizer = joblib.load("tfidf_vectorizer.pkl")
    lbl_enc = joblib.load("label_encoder.pkl")
    logging.info("Model and vectorizers loaded successfully.")
except Exception as e:
    logging.error(f"Failed to load model/vectorizer: {e}")
    raise

# Preprocessing function
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'http[s]?://\S+', '', text)  # Remove URLs
    text = re.sub(r'\[.*?\]\(.*?\)', '', text)  # Remove markdown links
    text = re.sub(r'@\w+', '', text)            # Remove mentions
    text = re.sub(r'[^\w\s]', '', text)         # Remove punctuation
    tokens = word_tokenize(text)
    stemmer = PorterStemmer()
    stemmed_tokens = ' '.join(stemmer.stem(token) for token in tokens)
    return stemmed_tokens

# Prediction function
def predict_mental_health_status(text):
    preprocessed_text = preprocess_text(text)
    tfidf_features = vectorizer.transform([preprocessed_text])
    num_characters = len(text)
    num_sentences = len(sent_tokenize(text))
    additional_features = np.array([[num_characters, num_sentences]])
    combined_features = hstack([tfidf_features, additional_features])
    prediction = xgb_model.predict(combined_features)[0]
    predicted_label = lbl_enc.inverse_transform([prediction])[0]
    return predicted_label

# API Endpoint
@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    if not data or 'text' not in data:
        return jsonify({'error': 'Missing "text" in request'}), 400

    user_text = data['text']
    try:
        result = predict_mental_health_status(user_text)
        return jsonify({'user_text': user_text, 'prediction': result})
    except Exception as e:
        logging.error(f"Prediction error: {e}")
        return jsonify({'error': str(e)}), 500

# Run server
if __name__ == '__main__':
    app.run(debug=True)
```

Conclusion:

The project effectively combines NLP, classical ML, and deployment:

- Achieves about **81% accuracy** with XGBoost.
- Handles real-time classification via a robust **Flask API**.
- Useful for mental health monitoring tools, clinical screening, or digital therapeutics.