Emotion Analysis and Detection in Mental Health Using Natural Language Processing Techniques

- Ganga Vamsik S

Introduction

- The intersection of Natural Language Processing (NLP) and mental health is a rapidly growing field, with significant potential to improve emotional understanding through technology.
- This project uses NLP techniques to analyze and classify emotions within mental health-related conversations.
- It focuses on NLP-based framework that processes and interprets user-generated text data, specifically targeting emotional expressions associated with mental health issues.
- This initiative aims to provide insights that can enhance mental health support and interventions using different NLP models.

Dataset Overview

- NLP Mental Health Conversations (Emotion Detection NLP Mental Health): This dataset comprises real-world conversations about mental health on social media , allowing us to understand emotional expressions within various contexts.
- Dataset Overview

First 5 Rows of the Dataset:

[2]:

	tweet_id	Emotion	Text
0	1956967341	empty	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	wants to hang out with friends SOON!
4	1956968416	neutral	@dannycastillo We want to trade with someone w...

Text Preprocessing

i. Text Cleaning: Convert text to lowercase, remove URLs, mentions, hashtags, punctuation, and numbers, then tokenize.

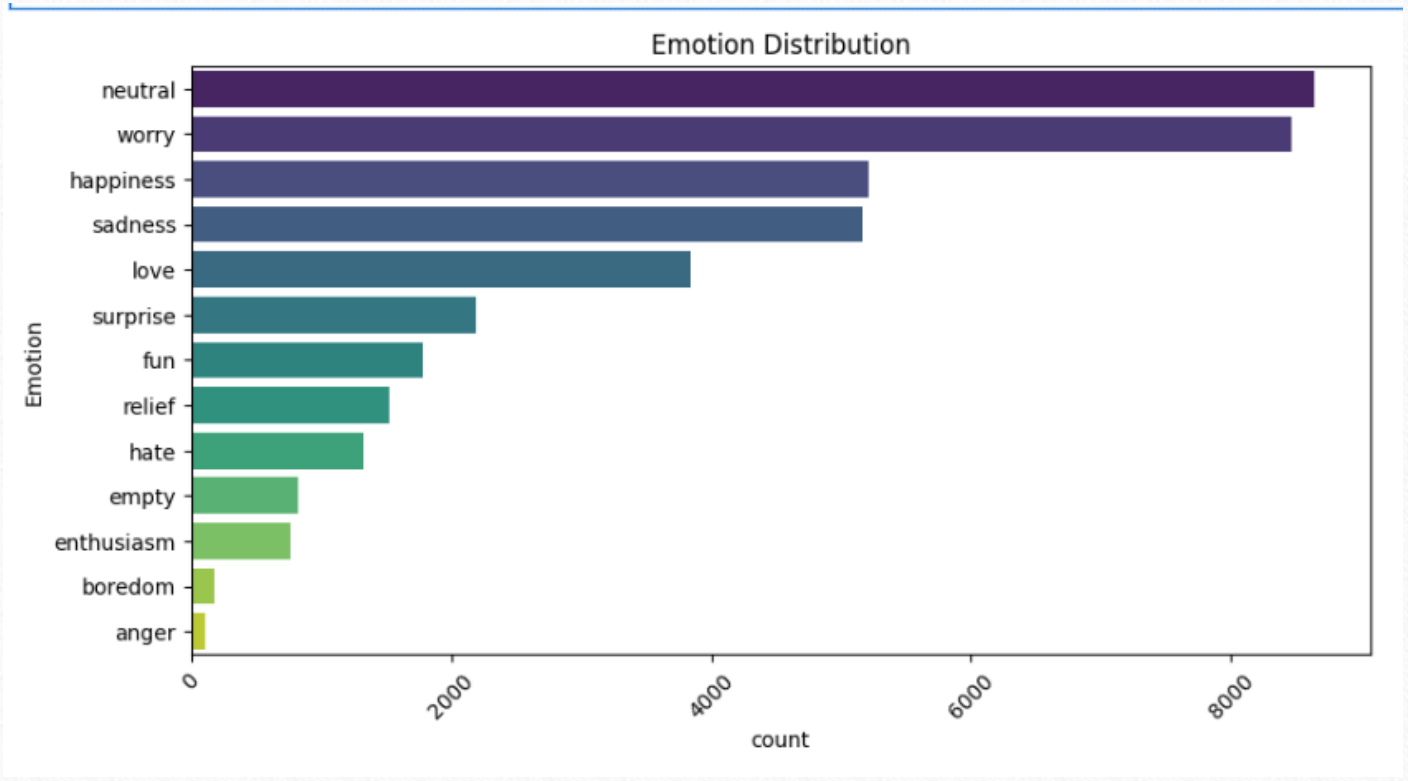
ii. Stopword Removal: Exclude common stop words and lemmatize words using WordNetLemmatizer to reduce inflection.

iii. Dataset Processing: Apply preprocessing on dataset text to create a cleaned column for further analysis

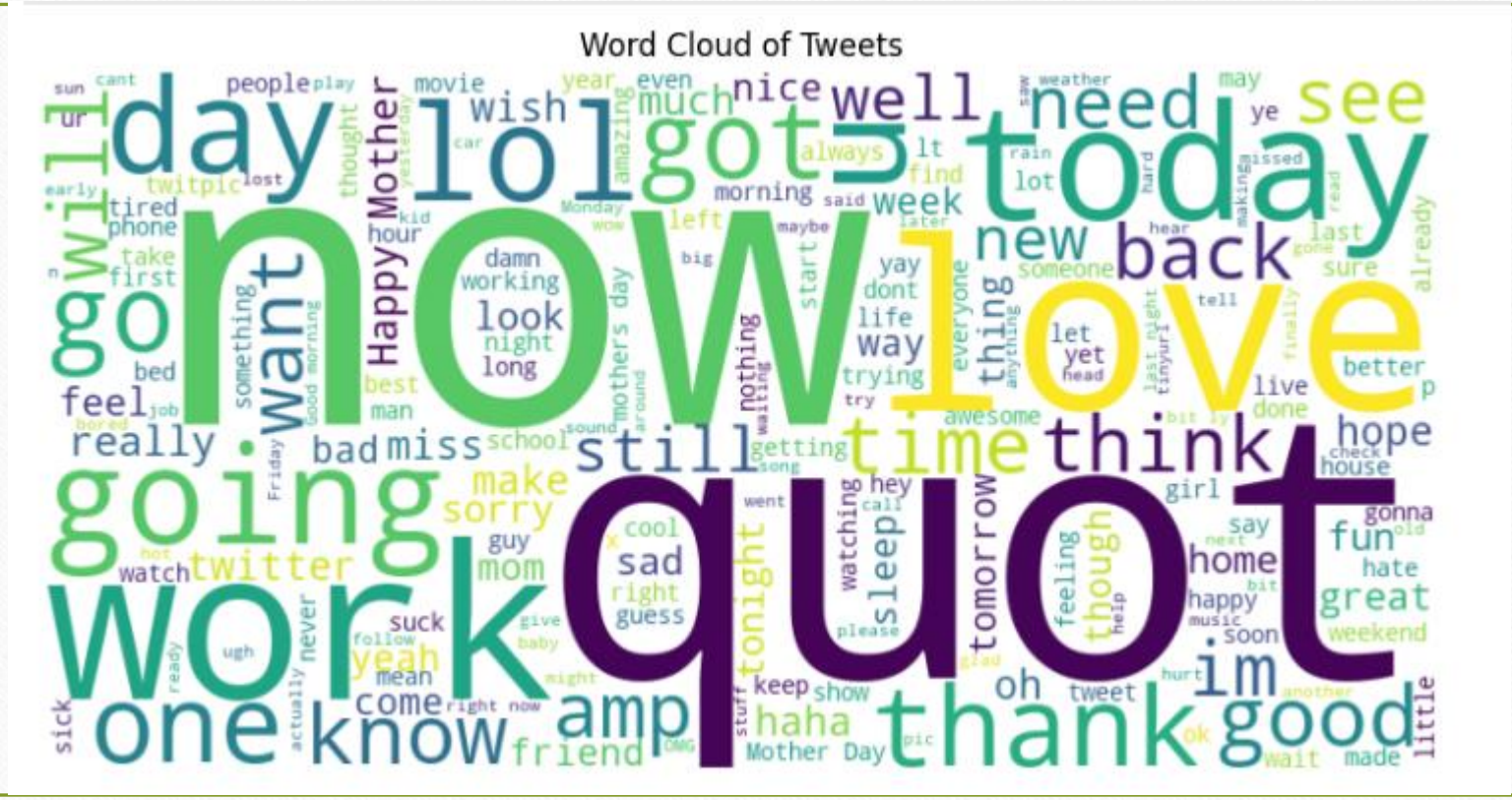
	Text
0	@tiffanylue i know i was listenin to bad habi...
1	Layin n bed with a headache ughhhh...waitin o...
2	Funeral ceremony...gloomy friday...
3	wants to hang out with friends SOON!
4	@dannycastillo We want to trade with someone w...

	Cleaned_Text
0	know listenin bad habit earlier started freaki...
1	layin n bed headache ughhhhwaitin call
2	funeral ceremonygloomy friday
3	want hang friend soon
4	want trade someone houston ticket one

Emotion Distribution



Conversations Overview



Methodology:

- Label Encoding: Encodes categorical emotion labels into numerical values using LabelEncoder.
- Data Splitting: Splits the data into training (80%), validation (10%), and test (10%) sets, ensuring class balance with stratify.
- Count Vectorization: Converts text data into numerical vectors using CountVectorizer, limiting features to 10,000.
- Tensor Conversion: Converts the vectorized text and encoded labels into PyTorch tensors for use in deep learning.

Model 1: EmotionClassifier

- A simple feedforward neural network with two fully connected layers, ReLU activation, dropout for regularization, and 13 output classes for emotion classification.

```
class EmotionClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(EmotionClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Model Parameters
input_dim = X_train_vec.shape[1]
hidden_dim = 128
output_dim = len(label_encoder.classes_)

# Initialize Model, Loss, and Optimizer
model = EmotionClassifier(input_dim, hidden_dim, output_dim)
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=0.001)
```

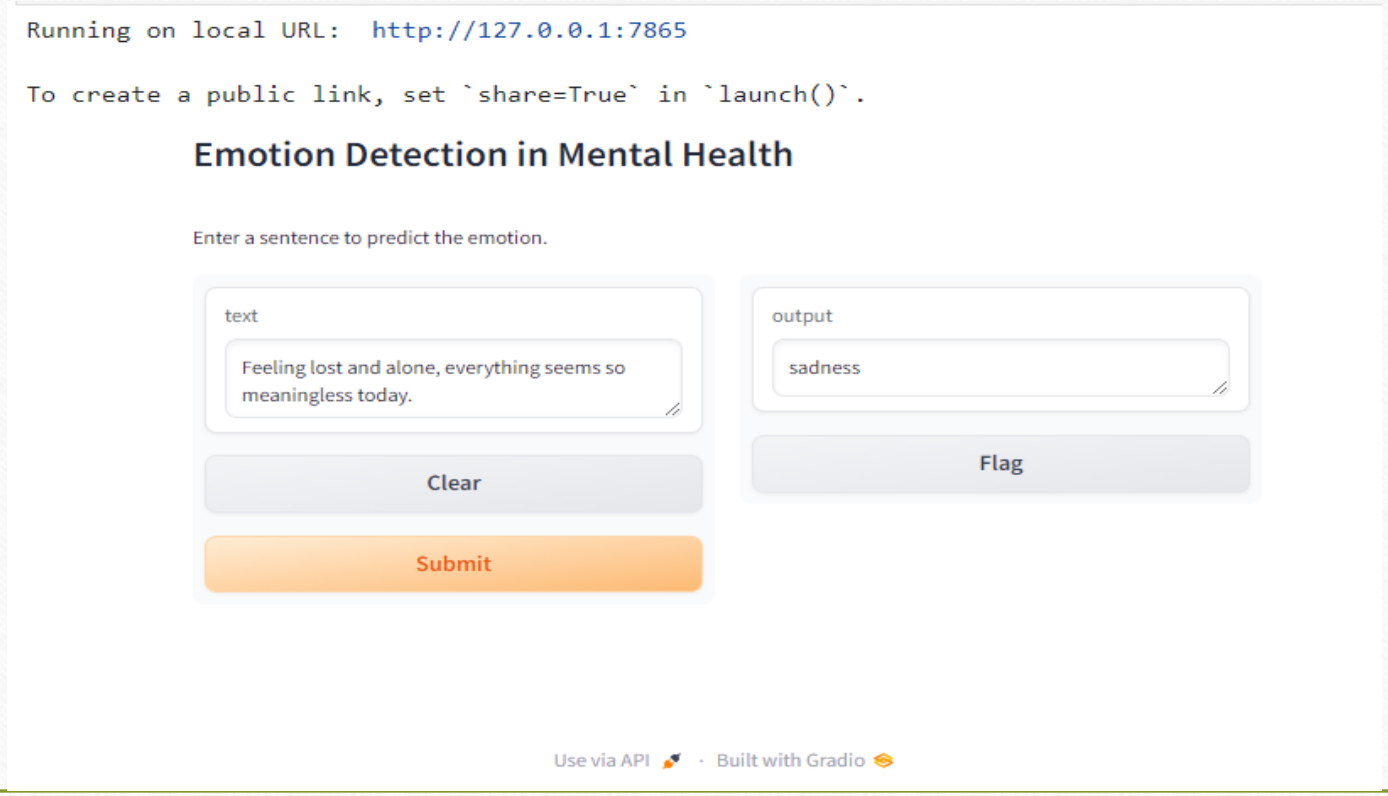
Model 1: Performance

- The classification report shows low precision, recall, and F1-scores across most emotions, with overall accuracy of 30%, indicating room for improvement in the model's performance and class balance.

Classification Report:

	precision	recall	f1-score	support
anger	0.00	0.00	0.00	11
boredom	0.25	0.06	0.09	18
empty	0.00	0.00	0.00	82
enthusiasm	0.00	0.00	0.00	76
fun	0.12	0.08	0.10	177
happiness	0.27	0.32	0.29	521
hate	0.24	0.15	0.19	133
love	0.40	0.38	0.39	384
neutral	0.35	0.45	0.39	864
relief	0.09	0.05	0.07	153
sadness	0.27	0.27	0.27	517
surprise	0.16	0.09	0.12	218
worry	0.33	0.36	0.35	846
accuracy			0.30	4000
macro avg	0.19	0.17	0.17	4000
weighted avg	0.28	0.30	0.29	4000

Model 1: Deployment



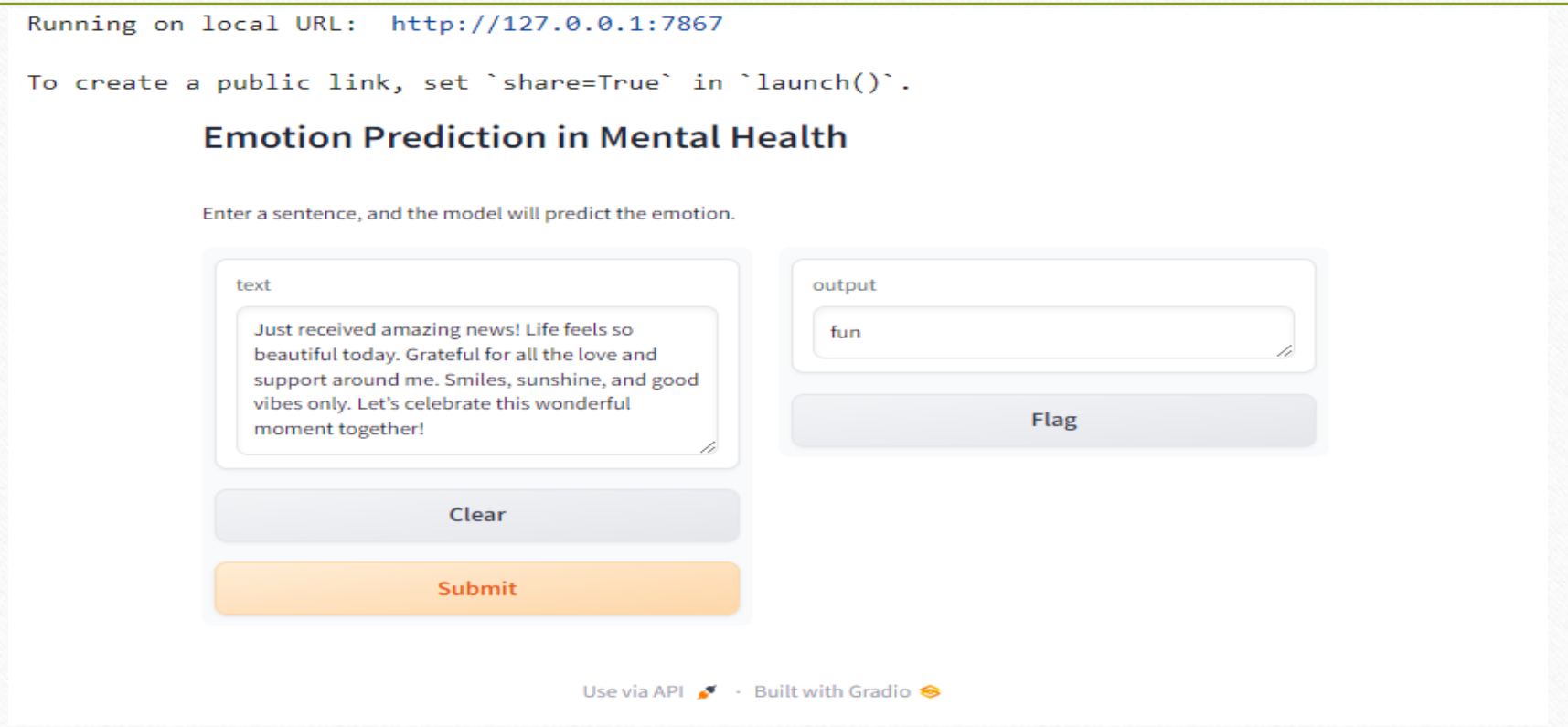
Model 2: DistilBERT

- The **DistilBERT** model is a lightweight, distilled version of BERT, offering faster inference while maintaining high accuracy.
- It is ideal for real-time emotion prediction, leveraging transformer-based embeddings for nuanced text understanding.

```
# The pretrained DistilBERT model and tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=13)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

emotion_labels = [
    'happy', 'angry', 'sad', 'surprised', 'fearful', 'disgusted',
    'neutral', 'worry', 'happiness', 'sadness', 'love', 'surprise',
    'fun', 'relief', 'hate', 'empty', 'enthusiasm', 'boredom', 'anger'
]
```

Model 2: DistilBERT Deployment



Model 3: BERT Model

- The BertForSequenceClassification model is built upon the BERT architecture, which uses 12 transformer layers for deep contextual understanding of text.
- It includes a classification head comprising a dropout layer and a fully connected layer for 13 output classes, ideal for emotion prediction.

```
23]: BertForSequenceClassification(
       (bert): BertModel(
         (embeddings): BertEmbeddings(
           (word_embeddings): Embedding(30522, 768, padding_idx=0)
           (position_embeddings): Embedding(512, 768)
           (token_type_embeddings): Embedding(2, 768)
           (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
           (dropout): Dropout(p=0.1, inplace=False)
         )
         (encoder): BertEncoder(
           (layer): ModuleList(
             (0-11): 12 x BertLayer(
               (attention): BertAttention(
                 (self): BertSelfAttention(
                   (query): Linear(in_features=768, out_features=768, bias=True)
                   (key): Linear(in_features=768, out_features=768, bias=True)
                   (value): Linear(in_features=768, out_features=768, bias=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
                 (output): BertSelfOutput(
                   (dense): Linear(in_features=768, out_features=768, bias=True)
                   (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                   (dropout): Dropout(p=0.1, inplace=False)
                 )
               )
               (intermediate): BertIntermediate(
                 (dense): Linear(in_features=768, out_features=3072, bias=True)
                 (intermediate_act_fn): GELUActivation()
               )
               (output): BertOutput(
                 (dense): Linear(in_features=3072, out_features=768, bias=True)
                 (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                 (dropout): Dropout(p=0.1, inplace=False)
               )
             )
           )
         )
         (pooler): BertPooler(
           (dense): Linear(in_features=768, out_features=768, bias=True)
           (activation): Tanh()
         )
       )
       (dropout): Dropout(p=0.1, inplace=False)
       (classifier): Linear(in_features=768, out_features=13, bias=True)
     )
```

Model 3: BERT Deployment