Automated Stress Detection in Reddit Posts Using Traditional and Deep Learning Approaches

Ximing Wan

University of Melbourne, Text Analytics for Health (COMP90090-2025-SM1)

ABSTRACT

Objective: Using the Dreaddit data set to find out how well deep learning and traditional models of machine learning perform in distinguishing between stressful and non-stressful Reddit posts.
Materials and Methods: The Dreaddit dataset was analyzed, comprising 3,000 Reddit posts manually annotated as stressful or not-stressful. Cleaning, tokenization, and stopword removal were all part of the preprocessing. Feature representations were generated using TF-IDF, GloVe (100d embeddings from Stanford NLP), DistilBERT embeddings, and the domain-specific "mental-roberta-base" model from Hugging Face. Models tested included SVM, Random Forest, BiLSTM, and fine-tuned transformer models. Accuracy, precision, recall, and F1-score were among the evaluation metrics.
Results: TF-IDF + SVM achieved an accuracy of 72%, while Random Forest + TF-IDF performed comparably (72%). BiLSTM and BERT embeddings + SVM attained around 70% accuracy. Fine-tuning the mental-roberta-base model outperformed others, achieving 77% accuracy and superior macro-averaged F1-scores.
Discussion: The Transformer-based mental-roberta-base model consistently outperforms other models, demonstrating the advantages of domain-specific pre-training. However, while deep learning models can capture richer contextual features, simpler traditional models such as SVM are significantly faster to train in small sample environments, perform comparably well, and offer greater interpretability. For medical institutions without specialized equipment, this method may be more suitable for large-scale deployment.
Conclusion: Although transformer-based models optimised for domain data show promise in stress detection tasks, computational cost is still a major barrier to practical implementation.

Key words: TF-IDF, GloVe, SVM, Random Forest, BiLSTM, Transformer

INTRODUCTION

With the rise of social media platforms, many people openly "vent" their emotions online, making these platforms a valuable source of data for monitoring stress levels in the population. Reddit in particular, a popular forum where users can anonymously discuss personal challenges, often contains posts expressing stress. As for stress, it is a widespread public health issue. If stress is excessive or long-term, it can lead to many negative physical and mental health problems. Some authors even describe stress as a "silent killer", emphasizing the importance of early stress management[1]. Therefore, it is positive to detect psychological stress from personal network communication in a timely manner through machine learning, which can be used for early intervention before it leads to serious diseases such as anxiety, depression or cardiovascular disease.

Research on automatic stress detection emerges at the nexus of mental health and natural language processing (NLP). Simple classifiers and dictionary-based techniques were the mainstays of early approaches. For instance, TensiStrength, a rule-based system created by Thelwall (2017), infers the degree of stress and relaxation in brief texts with a stress dictionary[2]. Additional study has examined the linguistic characteristics linked to stress. Turcan and McKeown (2019) examined lexical patterns in stressed and unstressed texts using the Linguistic Inquiry Dictionary (LIWC)[3]. Subsequently, traditional machine learning techniques trained using bag-of-words models or TF-IDF features, such as logistic regression, naive Bayes, support vector machines (SVM), and random forests, have also been applied to stress classification in social media data. These methods have had some degree of success; for example, a recent study of academic Reddit communities by Oryngozha et al. used a logistic regression classifier on bag-of-words features with an accuracy of approximately 77% to 78%[4].

More recently, deep learning approaches have advanced the state of the art in text classification, including mental health domains. Bidirectional LSTMs (BiLSTMs) and transformer-based models can capture complex language patterns. Transformers like BERT (Bidirectional Encoder Representations from Transformers) have achieved state-of-the-art performance on many NLP tasks by learning rich contextual representations of words in context. In the realm of stress and mental health detection, researchers have begun fine-tuning such models on domain-specific data. For example, models initially pre-trained on general language (e.g. BERT) or on mental health-related corpora (e.g. MentalBERT/MentalRoBERTa) can be adapted to detect stress in social media posts. Ji et al. (2022) introduced MentalBERT and MentalRoBERTa, transformer models pre-trained on Reddit posts from mental health support communities, which improved performance on downstream mental health detection tasks[5].

In this study, we address the task of binary stress detection (stress vs. no stress) in Reddit posts using the Dreaddit dataset. Dreaddit provides a collection of posts from five Reddit domains (such as r/relationships, r/college, etc.), with ground truth labels indicating whether each post contains a stressor or the author is expressing stress. In total, around 3,000 posts are human-labeled (training and test data) for supervised learning. We build and compare multiple models: two traditional classifiers (SVM and Random Forest) using TF-IDF features, and two deep learning models (a BiLSTM and a fine-tuned transformer). By evaluating their performance, we aim to quantify the benefits and drawbacks of each approach. We also generate visualizations (word clouds, learning curves, confusion matrices) to interpret model behavior and important features. Ultimately, we discuss how such models could be deployed in real-world clinical or public health scenarios—for example, as tools for monitoring stress levels in specific communities or for flagging high-stress posts to moderators or health professionals—while considering the practicality, resource requirements, and ethical implications.

METHODS

Data and Preprocessing

We used the Dreaddit dataset introduced by Turcan and McKeown which were splited into a training set (dreaddit-train.csv) and test set (dreaddit-test.csv)[3]. The dataset consists of roughly 3,000 Reddit posts labeled by annotators for stress or no-stress, and were drawn from five different subreddit categories (relationships, academic, etc.), ensuring a mix of contexts in which users might express stress. Basic characteristics of the dataset included an average post length of a few sentences and an approximate balance between the stress and no-stress classes (Figure 1). For text preprocessing, we applied standard NLP cleaning steps. All text was lowercased, tokenized and removed English stop words which carry little semantic content. We also removed or normalized punctuation, URLs, user mentions, and other non-alphanumeric characters, since these were not directly useful for detecting stress. We then performed POS tagging on each word and lemmatized it, thereby restoring the word to its basic dictionary form to improve semantic consistency and modeling effect.After preprocessing, each post was represented in multiple ways for input into different models, as described below.

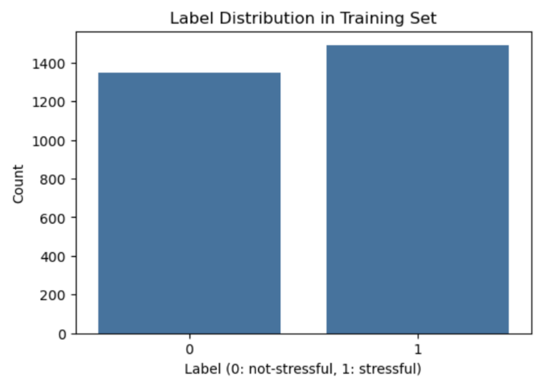

Figure 1. Label Distribution in Training Set.

Feature Extraction

For traditional machine learning models (Support Vector Machine (SVM) and Random Forest), we used Term Frequency-Inverse Document Frequency (TF-IDF). We built a vocabulary based on the preprocessed training set and converted each post into a high-dimensional sparse vector of TF-IDF weights for each word. This approach captures the importance of words (words that appear frequently in a post but rarely in other posts receive higher weights). We limited the size of the vocabulary to the top 5,000 words by frequency to reduce the dimensionality and filter out very rare words. Figure 2 shows the word cloud of the words with the highest weights extracted based on the TF-IDF feature.

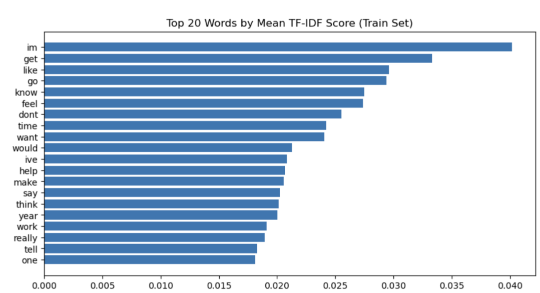

Figure 2. Top 20 Words by Mean TF-IDF Score (Train Set)

We used GloVe vectors to represent each word as a dense vector with 100 dimensions for the BiLSTM model. GloVe (global vectors) encode semantic similarity and are pre-trained on a large corpus(Pennington, Socher, & Manning, 2014)[6]. During LSTM training, we updated (fine-tuned) pre-trained GloVe 6B corpus vectors to fit our domain. Every post was transformed into a series of word vectors that were either padded or truncated to a predetermined length of 100 tokens, and BiLSTM used these sequences as input.

For Transformer-based models, we fine-tune mental-roberta-base. This model belongs to the MentalBERT family (Domain-Specific Language Model for Mental Health), initialized by RoBERTa architecture (12-layer Transformer), and trained on a large corpus of online posts on mental health issues (Ji et al., 2021)[7]. We loaded the mental-roberta-base model and tokenizer via Hugging Face Transformers. During fine-tuning, the model generates context vectors for the entire article, as well as a special CLS token vector that represents the aggregate meaning of the article and feeds it into the classifier layer. We also tried using BERT-base (pre-trained on general English) as a baseline to compare the effect of domain-specific pre-training. Preliminary results show that mental-roberta-base outperforms the general BERT-base, which is consistent with the finding that domain-specific pre-training can improve performance on mental health NLP tasks. Therefore, we report results for mental-roberta-based models as the primary Transformer approach.

Classifiers and Model Training

Four classification models were trained to identify signs of stress in Reddit posts. First, a SVM with a linear kernel and the default value of the regularization parameter C (1.0) was trained on TF-IDF features. This model is robust to high-dimensional sparse text data and is able to quickly and effectively classify "stressed" and "non-stressed" categories.

Second, a random forest model was trained on TF-IDF features with 100 decision trees and default settings (e.g., Gini impurity and maximum number of features as the square root of the total number of features). This model captures nonlinear interactions between words and has some ability to explain feature importance, although this is limited in the case of very high dimensional text.

The third model is a deep learning-based bidirectional LSTM (BiLSTM) neural network. We initialized the embedding layer with pre-trained GloVe word vectors and constructed an LSTM layer with 128 units in the forward and backward passes. The bidirectional structure can capture the contextual information of the sentence at the same time, such as expressions that require contextual judgment, such as "not coping well". The final concatenated hidden state is input into a fully connected layer to complete the binary classification. During the training process, the Adam optimizer (learning rate of 0.001) and dropout (0.3) are used to prevent overfitting, and the training rounds are controlled by the early stopping mechanism.

Finally, we fine-tuned the RoBERTa model (mental-roberta-base) based on the Transformer architecture. We added a linear classification layer to the output of its CLS token to determine whether the post expresses stress. Fine-tuning was performed using Hugging Face's Trainer, set to 3 epochs, batch size of 16, learning rate of 2e-5, and a linear learning rate decay strategy. Despite the small amount of training data, the model still showed good generalization ability because it was originally pre-trained on mental health data. For comparison, we also fine-tuned the general bert-base-uncased model with the same settings as RoBERTa (e.g., learning rate 2e-5, batch size 16, training for 3 epochs). Although this model was not specifically pre-trained on mental health corpus, it still performed well in this task, verifying the wide applicability of pre-trained Transformer models in stress detection.

Ethics Statement

This research exclusively uses publicly available Reddit posts from the Dreaddit dataset. The data was originally collected and released in prior work with annotations, and we comply with the terms of use of the dataset. No additional private or identifiable user information was collected. The Reddit platform policy allows the research use of public comments; however, we acknowledge that content can be sensitive as it pertains to mental health. We took care to anonymize the data (removing any usernames or personal references if present) in our processing. Because the data are public and anonymized, this study was exempt from institutional review by the IRB. We do not intend to redistribute the data or any content beyond derived analyses. We discuss only aggregate results and patterns (e.g., common stress indicators) to avoid any potential harm or stigma to individuals. All experiments were carried out according to relevant ethical guidelines for research on human communication. Finally, we note that automated stress detection is intended as a supporting tool for mental health outreach; any deployed system should include human oversight and robust privacy protections.

RESULTS

For the classification performance of each model is summarised in Figure 3-7. First, with precision, recall, and F1 scores balanced at 0.72 across both categories, the SVM model with TF-IDF features achieves 72% accuracy. likewise, the random forest model with TF-IDF features attains the same accuracy (72%), yet it has a slightly higher macro-average accuracy (0.74) and a higher recall for the stress category (0.87). Furthermore, the BiLSTM model with max pooling and a deeper classifier achieves 72% accuracy with a balance of precision and recall (both 0.72).

As can be seen, traditional machine learning models are usually fast and have fairly high accuracy. However, the enhanced domain-specific Transformer model (mental-roberta-base) outperformed traditional methods in this task (achieving an accuracy of 77%). The model has high macro-average precision (0.78) and recall (0.77), indicating that it is able to accurately identify stressful posts. In contrast, the performance of BERT embeddings alone (without task-specific fine-tuning) dropped to 69% overall accuracy and did not outperform traditional methods. This shows that fine-tuning and domain adaptation are critical to fully exploit the power of Transformer models in stress detection.

=== Test Set Performance (SVM + TF-IDF) ===

	precision	recall	f1-score	support
0	0.72	0.68	0.70	346
1	0.72	0.76	0.74	369
accuracy			0.72	715
macro avg	0.72	0.72	0.72	715
weighted avg	0.72	0.72	0.72	715

Figure 3. Test Set Performance (SVM + TF-IDF)

=== Test Set Performance (Random Forest + TF-IDF) ===

	precision	recall	f1-score	support
0	0.80	0.56	0.66	346
1	0.68	0.87	0.76	369
accuracy			0.72	715
macro avg	0.74	0.72	0.71	715
weighted avg	0.74	0.72	0.71	715

Figure 4. Test Set Performance (Random Forest + TF-IDF)

=== Test Set Performance (BiLSTM + Max Pooling + Deeper Classifier) ===

	precision	recall	f1-score	support
0	0.69	0.74	0.72	346
1	0.74	0.69	0.72	369
accuracy			0.72	715
macro avg	0.72	0.72	0.72	715
weighted avg	0.72	0.72	0.72	715

Figure 5. Test Set Performance (BiLSTM + Max Pooling + Deeper Classifier)

=== Test Set Performance (BERT Embeddings + SVM) ===

	precision	recall	f1-score	support
0	0.69	0.66	0.68	346
1	0.69	0.72	0.70	369
accuracy			0.69	715
macro avg	0.69	0.69	0.69	715
weighted avg	0.69	0.69	0.69	715

Figure 6. Test Set Performance (BERT Embeddings + SVM)

=== Classification Report on Test Set (mental-roberta-base) ===

	precision	recall	f1-score	support
0	0.79	0.73	0.76	346
1	0.76	0.82	0.79	369
accuracy			0.77	715
macro avg	0.78	0.77	0.77	715
weighted avg	0.78	0.77	0.77	715

Figure 7. Classification Report on Test Set (mental-roberta-base)

DISCUSSION

The performance differences observed between different models in this experiment may stem from their ability to capture complex stress-related language features. To explain, the excellent performance of mental-roberta-base suggests that pre-training on mental health-related data can enhance contextual understanding. In contrast, the poor performance of individual BiLSTM and BERT embeddings may be due to their limited domain adaptation due to lack of fine-tuning, which may make them misled by the context; for example, ”using stress to make cookies” is understood as psychological stress. However, traditional models such as SVM perform surprisingly well on small datasets, which may benefit from simpler decision boundaries that are able to fit the data without overfitting. As the result, these patterns highlight the importance of domain specific pre-training and the trade-off between model complexity and training data size in stress detection.

CONCLUSION

In summary, this study finds that the fine-tuned mental-roberta-base outperforms traditional methods as well as unspecifically optimized deep learning for stress detection on the Dreaddit dataset; this result highlights the advantages of domain-specific Transformers in capturing complex stressful linguistic cues. However, classical models sacrifice some accuracy for faster training and higher interpretability. Future research should build on these findings by training on larger and more diverse datasets to improve generalization as well as exploring domain adaptation to maintain performance across different social platforms. These measures will contribute to the responsible deployment of automated stress detection systems in real-world environments.

REFERENCES

1. Balwan WK, Kour S. A Systematic Review of Hypertension and Stress – The Silent Killers. Scholars Academic Journal of Biosciences. 2021;9(6):150-4. Available from: https://saspublishers.com/media/articles/SAJB_96_154-158.pdf.

2. Thelwall M. TensiStrength: Stress and relaxation magnitude detection for social media texts. Information Processing & Management. 2017;53(1):106-21. Available from: https://www.sciencedirect.com/science/article/pii/S0306457316302321.

3. Turcan E, McKeown K. Dreaddit: A Reddit Dataset for Stress Analysis in Social Media. In: Proceedings of the Tenth International Workshop on Health Text Mining and Information Analysis (LOUHI 2019). Hong Kong: Association for Computational Linguistics; 2019. p. 97-107. Available from: https://aclanthology.org/D19-6213/.

4. Oryngozha N, Shamoi P, Igali A. Detection and Analysis of Stress-Related Posts in Reddit's Academic Communities. IEEE Access. 2024;12:14932-48. Available from: https://ieeexplore.ieee.org/document/10412045.

5. Ji S, Zhang T, Ansari L, Fu J, Tiwari P, Cambria E. MentalBERT: Publicly Available Pretrained Language Models for Mental Healthcare. In: Proceedings of the Thirteenth Language Resources and Evaluation Conference. European Language Resources Association; 2022. p. 4346-52.

6. Pennington J, Socher R, Manning CD. GloVe: Global Vectors for Word Representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics; 2014. p. 1532-43. Available from: https://aclanthology.org/D14-1162/.

7. Ji S, Zhang T, Ansari L, Fu J, Tiwari P, Cambria E. MentalBERT: Publicly Available Pretrained Language Models for Mental Healthcare. arXiv preprint arXiv:211015621. 2021. Available from: https://huggingface.co/papers/2110.15621.

APPENDIX: KEY JUPYTER NOTEBOOK CODE

GitHub:https://github.com/Nooob233/nlp4health-a3.git

```python
#  ------------ Import necessary libraries ------------
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn.model_selection import train_test_split
import nltk
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk import pos_tag


# === Read the dataset ===
# Load Dreaddit training and test sets
train_path = "CORPORA/REDDIT/DREADDIT/dreaddit-train.csv"
test_path = "CORPORA/REDDIT/DREADDIT/dreaddit-test.csv"

train_raw = pd.read_csv(train_path)
test_raw = pd.read_csv(test_path)


# Keep only the 'text' and 'label' columns
train_df = train_raw[['text', 'label']].copy()
test_df = test_raw[['text', 'label']].copy()

print("Number of training samples:", train_df.shape[0])
print("Number of test samples:", test_df.shape[0])
print("\nFirst 3 rows of training data:\n", train_df.head(3))


#  ------------Exploratory Data Analysis ------------
# Visualize the label distribution in the training set
plt.figure(figsize=(6,4))
sns.countplot(data=train_df, x='label')
plt.title("Label Distribution in Training Set")
plt.xlabel("Label (0: not-stressful, 1: stressful)")
plt.ylabel("Count")
plt.show()


#  ------------ Text Cleaning and Preprocessing  ------------
# Download stopword list and WordNet resources
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Function to map Penn Treebank tags to WordNet POS tags
def get_wordnet_pos(treebank_tag):
    """
    Map Penn Treebank POS tags to WordNet POS tags
    """
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
```

```python
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # fallback to noun

# Load stop words and initialize lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Define a function to clean the text data
def clean_text(text):
    """
    Full cleaning process:Remove HTML tags and URLs, Remove non-alphabetic characters, Lowercase text,
    Tokenize and remove stop words, POS tagging and lemmatization
    """
    text = re.sub(r'<.*?>', '', str(text))  # Remove HTML tags
    text = re.sub(r'http\S+|www\S+|https\S+', '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove non-alphabetic characters
    text = text.lower()  # Lowercase

    tokens = word_tokenize(text)  # Tokenize
    tokens = [word for word in tokens if word not in stop_words]  # Remove stopwords
    pos_tags = pos_tag(tokens)  # POS tagging

    # Lemmatization with accurate POS
    lemmatized_tokens = [
        lemmatizer.lemmatize(word, get_wordnet_pos(tag))
        for word, tag in pos_tags
    ]
    return ' '.join(lemmatized_tokens)

# Apply the cleaning function to training and test data
train_df['clean_text'] = train_df['text'].apply(clean_text)
test_df['clean_text'] = test_df['text'].apply(clean_text)

# Display a few cleaned examples for sanity check
print("\nExample of cleaned text:\n", train_df[['text', 'clean_text']].head(3))


# ------------ Split the training data into train and validation sets (80/20) ------------
# Stratify ensures the label distribution is preserved
train_final, valid_final = train_test_split(
    train_df[['clean_text', 'label']],
    test_size=0.2,
    random_state=42,
    stratify=train_df['label']
)

print("\nSize of training set after split:", train_final.shape)
print("Size of validation set after split:", valid_final.shape)

# === Save cleaned datasets for future use ===
# Save cleaned training, validation, and test sets to CSV files
train_final.to_csv("clean_train.csv", index=False)
valid_final.to_csv("clean_valid.csv", index=False)
test_df[['clean_text', 'label']].to_csv("clean_test.csv", index=False)

print("\nCleaned datasets saved successfully.")


# ------------ Visualize the distribution of text lengths before and after cleaning in the same plot, with annotations ------------
# Compute the number of words before and after cleaning
train_df['text_length'] = train_df['text'].apply(lambda x: len(str(x).split()))
train_df['clean_text_length'] = train_df['clean_text'].apply(lambda x: len(str(x).split()))

plt.figure(figsize=(8,5))

# Plot original text length distribution
sns.histplot(train_df['text_length'], bins=30, kde=True, color='blue', label='Before Cleaning', alpha=0.5)
# Plot cleaned text length distribution
sns.histplot(train_df['clean_text_length'], bins=30, kde=True, color='green', label='After Cleaning', alpha=0.5)

# Calculate most frequent text lengths (
```

```
    mode)
mode_before = train_df['text_length'].mode
    ()[0]
mode_after = train_df['clean_text_length'
    ].mode()[0]

# Calculate maximum text lengths
max_before = train_df['text_length'].max()
max_after = train_df['clean_text_length'].
    max()

# Annotate mode (most frequent)
plt.axvline(mode_before, color='blue',
    linestyle='--', label=f"Most Frequent (
    Before): {mode_before}")
plt.axvline(mode_after, color='green',
    linestyle='--', label=f"Most Frequent (
    After): {mode_after}")

# Annotate maximum length
plt.scatter(max_before, 0, color='blue',
    marker='o', s=100, label=f"Max (Before)
    : {max_before}")
plt.scatter(max_after, 0, color='green',
    marker='o', s=100, label=f"Max (After):
     {max_after}")

plt.title("Comparison of Text Length
    Distribution (Before vs After Cleaning)
    ")
plt.xlabel("Number of Words")
plt.ylabel("Number of Samples")
plt.legend()
plt.show()


#  ------------ Generate word cloud
    visualization for high-frequency words
     ------------
from wordcloud import WordCloud

# Concatenate all cleaned text into a
    single string
all_text = ' '.join(train_df['clean_text'
    ].dropna().tolist())

# Create and configure the word cloud
    object
wordcloud = WordCloud(width=800, height
    =400,
                      background_color='
                          white',
                      max_words=100,  #
                          show top 100
                          words
                      contour_width=1,
                      contour_color='
                          steelblue').
                          generate(all_text
                          )

# Plot the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='
    bilinear')
plt.axis('off')
plt.title("Word Cloud of Cleaned Training
    Text")
plt.show()


# ----------------Re-import pandas and
    read the cleaned datasets ---------
import pandas as pd

# Read cleaned training, validation, and
    test sets
clean_train = pd.read_csv("clean_train.csv
    ")
clean_valid = pd.read_csv("clean_valid.csv
    ")
clean_test = pd.read_csv("clean_test.csv")

# Display the first few rows of each to
    confirm successful loading
print("First 3 rows of clean_train:")
print(clean_train.head(3))

print("\nFirst 3 rows of clean_valid:")
print(clean_valid.head(3))

print("\nFirst 3 rows of clean_test:")
print(clean_test.head(3))


# *-------------------- TF-IDF Feature
    Extraction ----------
from sklearn.feature_extraction.text
    import TfidfVectorizer

# Initialize TF-IDF vectorizer for
    unigrams
tfidf_vectorizer = TfidfVectorizer(
    max_features=5000)

# Fit on the training set and transform
    train/valid/test sets
X_train_tfidf = tfidf_vectorizer.
    fit_transform(clean_train['clean_text'
    ])
X_valid_tfidf = tfidf_vectorizer.transform
    (clean_valid['clean_text'])
X_test_tfidf = tfidf_vectorizer.transform(
    clean_test['clean_text'])

print("\nTF-IDF features generated
    successfully.")
print("Shape of training TF-IDF matrix:",
    X_train_tfidf.shape)
print("Shape of validation TF-IDF matrix:"
```

```python
    , X_valid_tfidf.shape)
print("Shape of test TF-IDF matrix:", X_test_tfidf.shape)


# ---------   Visualize top 20 high-frequency words in training data
import numpy as np
import matplotlib.pyplot as plt

feature_names = tfidf_vectorizer.get_feature_names_out()
mean_tfidf = np.array(X_train_tfidf.mean(axis=0)).flatten()
top_indices = mean_tfidf.argsort()[::-1][:20]

top_words = feature_names[top_indices]
top_scores = mean_tfidf[top_indices]

plt.figure(figsize=(10, 5))
plt.barh(top_words[::-1], top_scores[::-1])
plt.xlabel("Mean TF-IDF Score")
plt.title("Top 20 Words by Mean TF-IDF Score (Train Set)")
plt.show()


# ------------- SVM model with TFIDF training & validation
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

# Prepare target labels
y_train = clean_train['label']
y_valid = clean_valid['label']
y_test = clean_test['label']

# Initialize and train SVM
svm_tfidf = LinearSVC(random_state=42)
svm_tfidf.fit(X_train_tfidf, y_train)

# Predict on validation set
y_valid_pred = svm_tfidf.predict(X_valid_tfidf)

# Evaluate validation set
print("\n=== Validation Set Performance (SVM + TF-IDF) ===")
print(classification_report(y_valid, y_valid_pred))


# Visualize confusion matrix
cm = confusion_matrix(y_valid, y_valid_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title("Confusion Matrix: SVM + TF-IDF on Validation Set")
plt.show()


# ------------ Test set evaluation
y_test_pred = svm_tfidf.predict(X_test_tfidf)
print("\n=== Test Set Performance (SVM + TF-IDF) ===")
print(classification_report(y_test, y_test_pred))

# Load cleaned data for TF-IDF features
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

clean_train = pd.read_csv("clean_train.csv")
clean_valid = pd.read_csv("clean_valid.csv")
clean_test = pd.read_csv("clean_test.csv")

# TF-IDF Feature Extraction
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = tfidf_vectorizer.fit_transform(clean_train['clean_text'])
X_valid_tfidf = tfidf_vectorizer.transform(clean_valid['clean_text'])
X_test_tfidf = tfidf_vectorizer.transform(clean_test['clean_text'])

# Train Random Forest on TF-IDF
y_train = clean_train['label']
y_valid = clean_valid['label']
y_test = clean_test['label']

rf_tfidf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_tfidf.fit(X_train_tfidf, y_train)

# Evaluate on test set
y_test_pred_tfidf = rf_tfidf.predict(X_test_tfidf)
print("\n=== Test Set Performance (Random Forest + TF-IDF) ===")
print(classification_report(y_test, y_test_pred_tfidf, digits=2))


# -------   Load Cleaned Data & Build
```

```
Vocab-------
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from collections import Counter

# Load cleaned datasets
clean_train = pd.read_csv("clean_train.csv")
clean_valid = pd.read_csv("clean_valid.csv")
clean_test = pd.read_csv("clean_test.csv")

# Build vocab manually
word_counter = Counter()
for text in clean_train['clean_text']:
    word_counter.update(text.split())

vocab = {"<pad>": 0, "<unk>": 1}
for idx, (word, _) in enumerate(word_counter.most_common(), start=2):
    vocab[word] = idx

def tokenize(text):
    return text.split()

def text_to_ids(text, vocab, max_len=100):
    tokens = tokenize(text)
    ids = [vocab.get(token, vocab["<unk>"]) for token in tokens]
    return ids[:max_len]

# ------- Load GloVe and Build Embedding Matrix-------
def load_glove_vectors(glove_file_path, embedding_dim=100):
    word_vectors = {}
    with open(glove_file_path, 'r', encoding='utf-8') as f:
        for line in f:
            parts = line.strip().split()
            word = parts[0]
            vector = np.asarray(parts[1:], dtype='float32')
            word_vectors[word] = vector
    return word_vectors

glove_vectors = load_glove_vectors("glove.6B.100d.txt", embedding_dim=100)
print("Loaded GloVe vectors. Example:", list(glove_vectors.items())[:1])

vocab_size = len(vocab)
embedding_dim = 100
embedding_matrix = np.random.normal(scale=0.6, size=(vocab_size, embedding_dim))
for word, idx in vocab.items():
    vector = glove_vectors.get(word)
    if vector is not None:
        embedding_matrix[idx] = vector
embedding_matrix = torch.tensor(embedding_matrix, dtype=torch.float32)

# ------- Dataset & DataLoader -------
class RedditDataset(Dataset):
    def __init__(self, texts, labels, vocab, max_len=100):
        self.texts = texts
        self.labels = labels
        self.vocab = vocab
        self.max_len = max_len

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

    def __getitem__(self, idx):
        token_ids = text_to_ids(self.texts[idx], self.vocab, self.max_len)
        return torch.tensor(token_ids), torch.tensor(self.labels[idx])

def collate_fn(batch):
    texts, labels = zip(*batch)
    texts_padded = pad_sequence(texts, batch_first=True, padding_value=vocab["<pad>"])
    return texts_padded, torch.tensor(labels)

train_dataset = RedditDataset(clean_train['clean_text'].tolist(), clean_train['label'].tolist(), vocab)
valid_dataset = RedditDataset(clean_valid['clean_text'].tolist(), clean_valid['label'].tolist(), vocab)
test_dataset = RedditDataset(clean_test['clean_text'].tolist(), clean_test['label'].tolist(), vocab)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True, collate_fn=collate_fn)
valid_loader = DataLoader(valid_dataset, batch_size=32, collate_fn=collate_fn)
test_loader = DataLoader(test_dataset, batch_size=32, collate_fn=collate_fn)


#------- BiLSTM Model with Max Pooling and Deeper Classifier-------
class BiLSTMMaxPoolClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim,
```

```
                 output_dim, pad_idx, embedding_matrix):
        super(BiLSTMMaxPoolClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=pad_idx)
        self.embedding.weight.data.copy_(embedding_matrix)
        self.embedding.weight.requires_grad = True

        self.lstm = nn.LSTM(embedding_dim, hidden_dim, bidirectional=True, batch_first=True)

        self.classifier = nn.Sequential(
            nn.Linear(hidden_dim * 2, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, output_dim)
        )

    def forward(self, text):
        embedded = self.embedding(text)
        output, _ = self.lstm(embedded)
        pooled, _ = torch.max(output, dim=1)
        return self.classifier(pooled)

hidden_dim = 128
output_dim = 2
pad_idx = vocab["<pad>"]
model = BiLSTMMaxPoolClassifier(len(vocab), embedding_dim, hidden_dim, output_dim, pad_idx, embedding_matrix)

# -------  Training & Evaluation -------
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

def train(model, iterator):
    model.train()
    total_loss = 0
    for texts, labels in iterator:
        texts, labels = texts.to(device), labels.to(device)
        optimizer.zero_grad()
        predictions = model(texts)
        loss = criterion(predictions, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(iterator)

def evaluate(model, iterator):
    model.eval()
    total_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for texts, labels in iterator:
            texts, labels = texts.to(device), labels.to(device)
            predictions = model(texts)
            loss = criterion(predictions, labels)
            total_loss += loss.item()
            preds = predictions.argmax(dim=1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)
    return total_loss / len(iterator), correct / total

for epoch in range(5):
    train_loss = train(model, train_loader)
    valid_loss, valid_acc = evaluate(model, valid_loader)
    print(f"Epoch {epoch+1}: Train Loss={train_loss:.4f}, Valid Loss={valid_loss:.4f}, Valid Acc={valid_acc:.4f}")


# -------  Test Set Performance (Classification Report) -------
from sklearn.metrics import classification_report

def get_preds_and_labels(model, iterator):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for texts, labels in iterator:
            texts = texts.to(device)
            predictions = model(texts)
            preds = predictions.argmax(dim=1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.numpy())
    return all_labels, all_preds

y_test_true, y_test_pred = get_preds_and_labels(model, test_loader)
print("\n=== Test Set Performance (BiLSTM + Max Pooling + Deeper Classifier) ===")
```

```python
print(classification_report(y_test_true, y_test_pred, digits=2))

# Load Cleaned Data
import pandas as pd

clean_train = pd.read_csv("clean_train.csv")
clean_valid = pd.read_csv("clean_valid.csv")
clean_test = pd.read_csv("clean_test.csv")
# Load pretrained BERT tokenizer & model
from transformers import AutoTokenizer, AutoModel
import torch

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Using DistilBERT (lighter, faster)
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
bert_model = AutoModel.from_pretrained("distilbert-base-uncased").to(device)

# Function to get sentence embeddings from BERT
def get_bert_embeddings(texts, tokenizer, model, batch_size=32, max_length=128):
    """
    Given a list of texts, compute the [CLS] token embeddings from BERT.
    """
    all_embeddings = []

    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i+batch_size]
        encoded = tokenizer(batch_texts, padding=True, truncation=True, max_length=max_length, return_tensors="pt")
        input_ids = encoded['input_ids'].to(device)
        attention_mask = encoded['attention_mask'].to(device)

        with torch.no_grad():
            outputs = model(input_ids, attention_mask=attention_mask)
            last_hidden_state = outputs.last_hidden_state
            cls_embeddings = last_hidden_state[:, 0, :].cpu().numpy()  # Take [CLS] token embedding
            all_embeddings.append(cls_embeddings)

    return np.vstack(all_embeddings)
# Generate sentence embeddings
X_train_bert = get_bert_embeddings(clean_train['clean_text'].tolist(), tokenizer, bert_model)
X_valid_bert = get_bert_embeddings(clean_valid['clean_text'].tolist(), tokenizer, bert_model)
X_test_bert = get_bert_embeddings(clean_test['clean_text'].tolist(), tokenizer, bert_model)

y_train = clean_train['label']
y_valid = clean_valid['label']
y_test = clean_test['label']


from sklearn.svm import SVC

# Initialize a SVM classifier
clf_svm = SVC(kernel='linear', C=1, random_state=42)
clf_svm.fit(X_train_bert, y_train)

# Predict labels for the test set using the trained SVM
y_test_pred_svm = clf_svm.predict(X_test_bert)

# Print the classification report (precision, recall, f1-score) for each class
from sklearn.metrics import classification_report
print("\n=== Test Set Performance (BERT Embeddings + SVM) ===")
print(classification_report(y_test, y_test_pred_svm, digits=2))


# Loading data & preprocessing
import pandas as pd
from datasets import Dataset

# Loading the cleaned Reddit dataset
train_df = pd.read_csv("clean_train.csv")
valid_df = pd.read_csv("clean_valid.csv")
test_df = pd.read_csv("clean_test.csv")

# Convert to Hugging Face Dataset format
train_dataset = Dataset.from_pandas(train_df)
valid_dataset = Dataset.from_pandas(valid_df)
test_dataset = Dataset.from_pandas(test_df)

# Load the pre-trained tokenizer for the mental-roberta-base model
```

```
from transformers import AutoTokenizer
# Initialize the tokenizer
tokenizer = AutoTokenizer.from_pretrained(
    "mental/mental-roberta-base")

# Define a tokenization function for the
    dataset
def tokenize(batch):
    """
    Tokenize each example in the dataset
        batch.
    - batch["clean_text"]: list of text
        samples to be tokenized
    - truncation: cuts text to max_length
        if needed
    - padding: ensures all sequences in
        the batch have the same length (
        max_length)
    - max_length: 128 tokens (fixed for
        all samples)
    """
    return tokenizer(batch["clean_text"],
        truncation=True, padding="
        max_length", max_length=128)

# Apply tokenization to the training,
    validation, and test sets
train_dataset = train_dataset.map(tokenize
    , batched=True)
valid_dataset = valid_dataset.map(tokenize
    , batched=True)
test_dataset = test_dataset.map(tokenize,
    batched=True)

# Remove the original text column as it's
    no longer needed after tokenization
train_dataset = train_dataset.
    remove_columns(["clean_text"])
valid_dataset = valid_dataset.
    remove_columns(["clean_text"])
test_dataset = test_dataset.remove_columns
    (["clean_text"])

# Rename the label column to 'labels' that
     required by Hugging Face Trainer API
train_dataset = train_dataset.
    rename_column("label", "labels")
valid_dataset = valid_dataset.
    rename_column("label", "labels")
test_dataset = test_dataset.rename_column(
    "label", "labels")

# Set dataset format to PyTorch tensors
    for compatibility with Trainer
train_dataset.set_format("torch")
valid_dataset.set_format("torch")
test_dataset.set_format("torch")

#Load the pre-trained mental-roberta-base
    model with a classification head
from transformers import
    AutoModelForSequenceClassification

# Load the pre-trained RoBERTa-based model
    .
# Since doing binary classification set
    num_labels=2.
model = AutoModelForSequenceClassification
    .from_pretrained("mental/mental-roberta
    -base", num_labels=2)

# Set up training arguments
from transformers import TrainingArguments
    , Trainer

# Initialize TrainingArguments with basic
    training configuration.
training_args = TrainingArguments(
    output_dir="./results_mental_roberta",
          # Directory to save model
        checkpoints and logs
    do_train=True,
                                # Enable
        training
    do_eval=True,
                                 # Enable
        evaluation during training
    learning_rate=2e-5,
                          # Common
        learning rate for fine-tuning
        transformers
    per_device_train_batch_size=16,
                # Batch size for training
    per_device_eval_batch_size=16,
                 # Batch size for
        evaluation
    num_train_epochs=3,
                          # Number of
        training epochs
    weight_decay=0.01,
                           # Weight decay
         for regularization
    logging_dir="./logs"
                         # Directory to
        store logs
)

# Define the evaluation metrics
import numpy as np
from sklearn.metrics import accuracy_score
    , precision_recall_fscore_support

def compute_metrics(eval_pred):
    """
    Compute standard classification
        metrics (accuracy, precision,
        recall, f1).

    Args:
    - eval_pred: a tuple (logits, labels)
```

```
        from the Trainer

    Returns:
    - A dictionary containing the metrics for logging and evaluation.
    """
    logits, labels = eval_pred
    # Convert logits to predicted labels
    predictions = np.argmax(logits, axis=-1)
    # Compute precision, recall, f1-score with macro averaging (equal weight for each class)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions, average="macro")
    # Compute overall accuracy
    acc = accuracy_score(labels, predictions)
    # Return metrics in a dictionary
    return {"accuracy": acc, "precision": precision, "recall": recall, "f1": f1}
# Create a Trainer instance and start fine-tuning
trainer = Trainer(
    model=model,                            # The model to fine-tune
    args=training_args,                     # Training parameters
    train_dataset=train_dataset,            # Dataset for training
    eval_dataset=valid_dataset,             # Dataset for validation
    tokenizer=tokenizer,                    # Tokenizer (needed for saving & loading pipeline)
    compute_metrics=compute_metrics         # Function to compute evaluation metrics
)

# Start the training process
trainer.train()

# Evaluate the model on the test dataset
print("\n=== Test Set Evaluation (mental-roberta-base) ===")
# Evaluate the model on the held-out test dataset
metrics = trainer.evaluate(test_dataset)
print(metrics)


# Generate a classification report on the test dataset
from sklearn.metrics import classification_report

# Predict the labels for the test dataset
preds_output = trainer.predict(test_dataset)
y_true = preds_output.label_ids
y_pred = np.argmax(preds_output.predictions, axis=1)

# Print detailed classification report
print("\n=== Classification Report on Test Set (mental-roberta-base) ===")
print(classification_report(y_true, y_pred, digits=2))
```