

✓ Sentiment Analysis in Mental Health

✓ Import Packages

```
!pip install sacremoses
```

⤵ Show hidden output

```
!pip install transformers datasets
```

⤵ Show hidden output

```
!pip install evaluate
```

⤵ Show hidden output

```
!pip3 install transformers datasets scikit-learn
```

⤵ Show hidden output

```
!pip install --upgrade transformers
```

⤵ Show hidden output

```
!pip install fasttext
```

```
import numpy as np
import pandas as pd
import time
import nltk
import math
import random
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
import seaborn as sns
import tensorflow as tf
import torch
from datasets import Dataset
from matplotlib import pyplot as plt
from sklearn.metrics import (classification_report,
                             confusion_matrix, accuracy_score, auc,
                             roc_auc_score, roc_curve,
                             ConfusionMatrixDisplay, label_binarize,
                             LabelEncoder)
from sklearn.model_selection import (StratifiedKFold, train_test_split)
from sklearn.preprocessing import LabelBinarizer
from tensorflow import keras
from tensorflow.keras.utils import to_categorical
from torch.utils.data import DataLoader, Dataset, random_split
from tqdm import tqdm
from transformers import (XLNetTokenizer,
                           AlbertTokenizer, BioGptTokenizer,
                           BioGptForSequenceClassification, TFBertForSequenceClassification, EarlyStoppingCallback,
                           AdamW, AlbertForSequenceClassification,
                           TXLNetForSequenceClassification,
                           TrainingArguments, Trainer, AutoModel, AutoTokenizer,
                           get_scheduler)

import evaluate
```

✓ Data Cleaning

```
#Connect with drive
from google.colab import drive
drive.mount('/content/drive')
```

→ Mounted at /content/drive

```
#Read dataset
dataset = pd.read_csv('/content/drive/MyDrive/Combined_Data.csv')
```

dataset

	Unnamed: 0	statement	status
0	0	oh my gosh	Anxiety
1	1	trouble sleeping, confused mind, restless hear...	Anxiety
2	2	All wrong, back off dear, forward doubt. Stay ...	Anxiety
3	3	I've shifted my focus to something else but I'...	Anxiety
4	4	I'm restless and restless, it's been a month n...	Anxiety
...
53038	53038	Nobody takes me seriously I've (24M) dealt wit...	Anxiety
53039	53039	selfishness "I don't feel very good, it's lik...	Anxiety
53040	53040	Is there any way to sleep better? I can't slee...	Anxiety
53041	53041	Public speaking tips? Hi, all. I have to give ...	Anxiety
53042	53042	I have really bad door anxiety! It's not about...	Anxiety

53043 rows × 3 columns

```
#Rename column Unnamed: 0 to Id
dataset = dataset.rename(columns={"Unnamed: 0": "Id"})
dataset.head()
```

	Id	statement	status
0	0	oh my gosh	Anxiety
1	1	trouble sleeping, confused mind, restless hear...	Anxiety
2	2	All wrong, back off dear, forward doubt. Stay ...	Anxiety
3	3	I've shifted my focus to something else but I'...	Anxiety
4	4	I'm restless and restless, it's been a month n...	Anxiety

```
#Find number of null values
print(dataset.isnull().sum())
```

```
#Print the null value rows
print(dataset[dataset.isnull().any(axis=1)])
```

	Id	statement	status
293	293	NaN	Anxiety
572	572	NaN	Anxiety
595	595	NaN	Anxiety
1539	1539	NaN	Normal
2448	2448	NaN	Normal
...
52838	52838	NaN	Anxiety
52870	52870	NaN	Anxiety
52936	52936	NaN	Anxiety
53010	53010	NaN	Anxiety
53031	53031	NaN	Anxiety

[362 rows × 3 columns]

```
#Drop the null value rows
dataset.dropna()
```

	Id	statement	status
0	0	oh my gosh	Anxiety
1	1	trouble sleeping, confused mind, restless hear...	Anxiety
2	2	All wrong, back off dear, forward doubt. Stay ...	Anxiety
3	3	I've shifted my focus to something else but I'...	Anxiety
4	4	I'm restless and restless, it's been a month n...	Anxiety
...
53038	53038	Nobody takes me seriously I've (24M) dealt wit...	Anxiety
53039	53039	selfishness "I don't feel very good, it's lik...	Anxiety
53040	53040	Is there any way to sleep better? I can't slee...	Anxiety
53041	53041	Public speaking tips? Hi, all. I have to give ...	Anxiety
53042	53042	I have really bad door anxiety! It's not about...	Anxiety

52681 rows × 3 columns

#Find the unique column values and set the labels for the same.

```
col_values= dataset['status'].unique()
print(col_values)
dataset.head()

→ ['Anxiety' 'Normal' 'Depression' 'Suicidal' 'Stress' 'Bipolar'
 'Personality disorder']
```

	Id	statement	status
0	0	oh my gosh	Anxiety
1	1	trouble sleeping, confused mind, restless hear...	Anxiety
2	2	All wrong, back off dear, forward doubt. Stay ...	Anxiety
3	3	I've shifted my focus to something else but I'...	Anxiety
4	4	I'm restless and restless, it's been a month n...	Anxiety

```
# Create a dictionary to map unique status values to numbers
status_mapping = {status: i for i, status in enumerate(col_values)}
print(status_mapping)
```

```
# Create the new column 'status_numeric' based on the mapping
dataset['labels'] = dataset['status'].map(status_mapping)
```

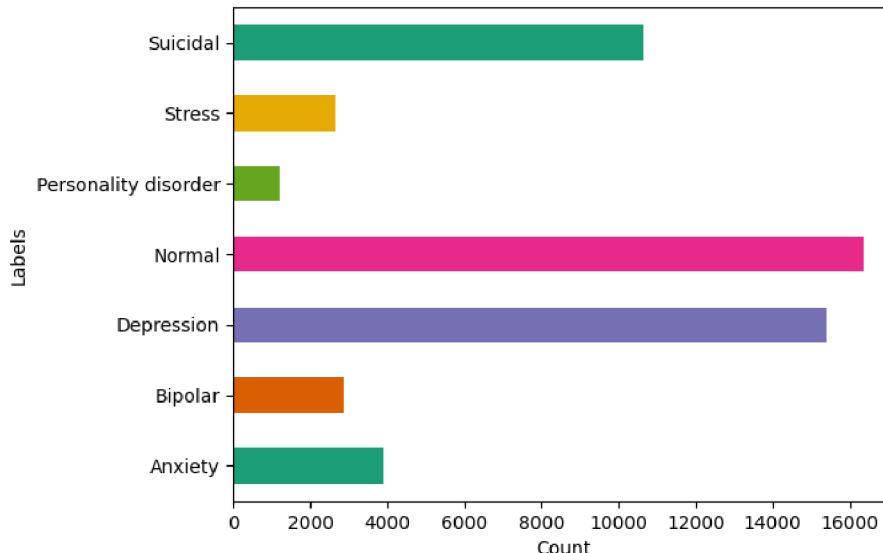
```
# Display the updated DataFrame
dataset.head()
```

	Id	statement	status	labels
0	0	oh my gosh	Anxiety	0
1	1	trouble sleeping, confused mind, restless hear...	Anxiety	0
2	2	All wrong, back off dear, forward doubt. Stay ...	Anxiety	0
3	3	I've shifted my focus to something else but I'...	Anxiety	0
4	4	I'm restless and restless, it's been a month n...	Anxiety	0

```
# Distribution Plot of Labels
dataset.groupby('status').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.title('Distribution of Labels')
plt.xlabel('Count')
plt.ylabel('Labels')
plt.gca().spines[['top', 'right']].set_visible(True)
```



Distribution of Labels



› Data Preprocessing

[] ↴ 3 cells hidden

▼ BioGPT

```
tokenizer = BioGptTokenizer.from_pretrained("microsoft/biogpt")
model = BioGptForSequenceClassification.from_pretrained("microsoft/biogpt", num_labels=len(status_mapping))

→ /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
    warnings.warn(
vocab.json: 100%                                     927k/927k [00:00<00:00, 6.34MB/s]
merges.txt: 100%                                    696k/696k [00:00<00:00, 11.5MB/s]
config.json: 100%                                    595/595 [00:00<00:00, 52.3kB/s]
pytorch_model.bin: 100%                                1.56G/1.56G [00:07<00:00, 223MB/s]
model.safetensors: 100%                               1.56G/1.56G [00:07<00:00, 230MB/s]

Some weights of BioGptForSequenceClassification were not initialized from the model checkpoint at microsoft/biogpt and are n
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

dataset.columns

→ Index(['Id', 'statement', 'status', 'labels', 'clean_text'], dtype='object')

```
X = dataset['clean_text']
y = dataset['labels']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=42)
```

```
word_counts_headline = X_train.astype(str).str.split().apply(len)
```

```
avg_word_count = word_counts_headline.quantile(0.9)
```

```
x_max_len = round(avg_word_count)
```

```
x_max_len
```

278

```
train_x_array = np.array(X_train)
val_x_array = np.array(X_val)
test_x_array = np.array(X_test)

y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)

train_encodings = tokenizer.batch_encode_plus([str(text) for text in train_x_array], truncation=True, padding=True, max_length=x_max
val_encodings = tokenizer.batch_encode_plus([str(text) for text in val_x_array], truncation=True, padding=True, max_length=x_max
test_encodings = tokenizer.batch_encode_plus([str(text) for text in test_x_array], truncation=True, padding=True, max_length=x_max

train_dataset = Dataset.from_dict({
    **train_encodings,
    'labels': y_train
})
val_dataset = Dataset.from_dict({
    **val_encodings,
    'labels': y_val
})
test_dataset = Dataset.from_dict({
    **test_encodings,
    'labels': y_test
})

train_dataset = train_dataset.with_format("torch")
val_dataset = val_dataset.with_format("torch")
test_dataset = test_dataset.with_format("torch")
```

```
accuracy_metric = evaluate.load("accuracy")
```

Downloading builder script: 100%

4.20k/4.20k [00:00<00:00, 466kB/s]

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    references = np.argmax(labels, axis=-1)
    return accuracy_metric.compute(predictions=predictions, references=references)

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=8,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    greater_is_better=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
)

trainer.train()
```

```
→ wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize?ref=models
wandb: Paste an API key from your profile and hit enter: .....
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: kr67 (kr67-rice-university) to https://api.wandb.ai. Use `wandb login --relogin` to force re
Tracking run with wandb version 0.19.10
Run data is saved locally in /content/wandb/run-20250430_014113-4cpl9njv
Syncing run /results to Weights & Biases (docs)
View project at https://wandb.ai/kr67-rice-university/huggingface
View run at https://wandb.ai/kr67-rice-university/huggingface/runs/4cpl9njv
```

[15918/21224 2:33:34 < 51:11, 1.73 it/s, Epoch 6/8]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.145000	0.144304	0.782805
2	0.125400	0.139692	0.782994
3	0.092700	0.136660	0.797511
4	0.061000	0.184026	0.803544
5	0.031800	0.248568	0.799585
6	0.018900	0.295055	0.795249

```
TrainOutput(global_step=15918, training_loss=0.08125931088207565, metrics={'train_runtime': 9241.5256,
'train_samples_per_second': 36.733, 'train_steps_per_second': 2.297, 'total_flos': 1.2838850002634342e+17, 'train_loss':
0.08125931088207565, 'epoch': 6.0})
```

```
predictions = trainer.predict(test_dataset)
predicted_labels = np.argmax(predictions.predictions, axis=1)
```

→

```
y_test_labels = np.argmax(y_test, axis=1)
```

```
y_test_labels
```

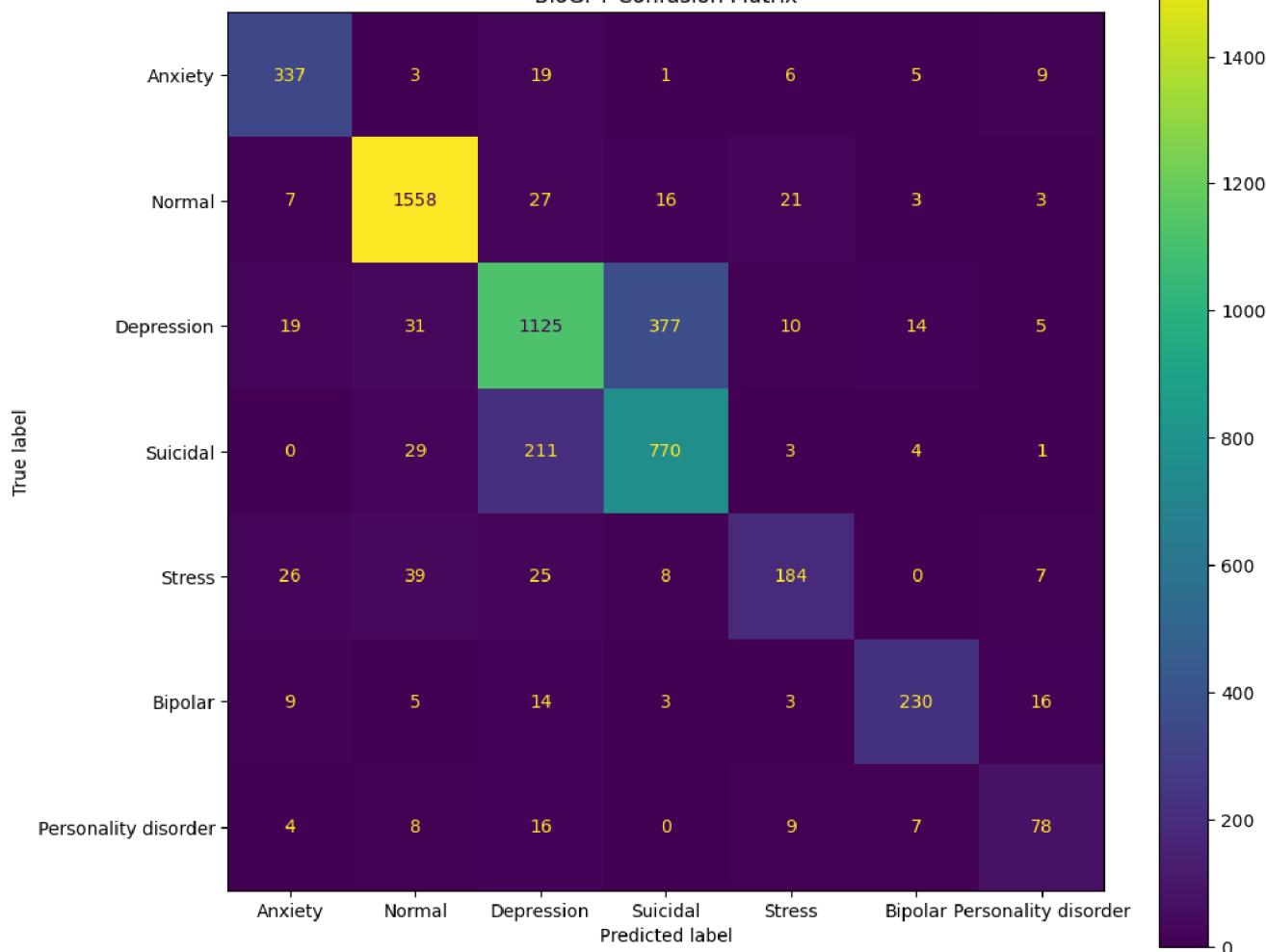
→ array([2, 2, 1, ..., 0, 2, 1])

```
confusion_matx = confusion_matrix(y_test_labels, predicted_labels)
```

```
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=confusion_matx, display_labels=status_mapping)
fig, ax = plt.subplots(figsize=(11, 10))
confusion_matrix_display.plot(ax=ax)
plt.title("BioGPT Confusion Matrix")
plt.show()
```



BioGPT Confusion Matrix



```
accuracy = np.trace(confusion_matx) / np.sum(confusion_matx)
```

```
print(f"Accuracy: {accuracy}")
```

→ Accuracy: 0.8071630537229029

```
micro_roc_auc_ovr = roc_auc_score(
    y_test,
    predictions.predictions,
    multi_class="ovr",
    average="micro",
)
```

```
print(f"One-vs-Rest AUROC score: {micro_roc_auc_ovr:.2f}")
```

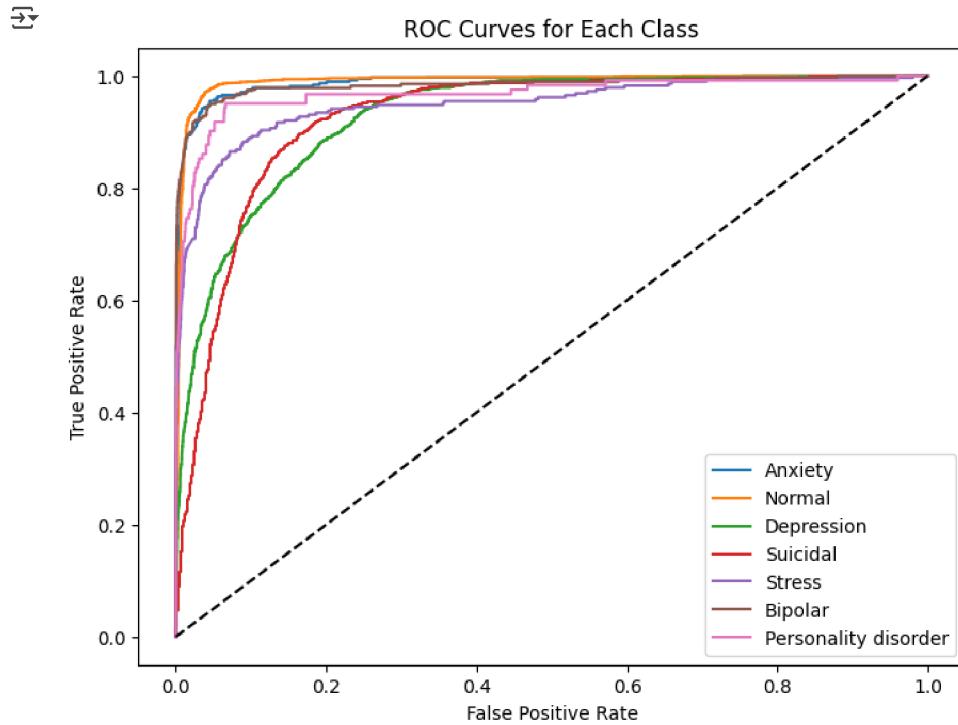
→ One-vs-Rest AUROC score: 0.98

```
num_labels = len(status_mapping)
status_keys = list(status_mapping.keys())
```

```
fpr = dict()
tpr = dict()
plt.figure(figsize=(8,6))
for i in range(num_labels):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], predictions.predictions[:, i])
    plt.plot(fpr[i], tpr[i], label=f'{status_keys[i]}')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Each Class')
```

```
plt.legend(loc="lower right")
plt.show()
```



✓ ALBERT

```
# 1. Load your CSV dataset
df = pd.read_csv("preprocessed_data.csv")
df = df[['clean_text', 'labels']]
df.dropna(inplace=True)

# 2. Load ALBERT tokenizer and model
tokenizer = AlbertTokenizer.from_pretrained("albert-base-v2")
model = AlbertForSequenceClassification.from_pretrained("albert-base-v2", num_labels=len(df['labels'].unique()))

# 3. Tokenize the text
encodings = tokenizer(
    list(df['clean_text']),
    truncation=True,
    padding='max_length',
    max_length=128,
    return_tensors='pt'
)

labels = torch.tensor(df['labels'].values)

# 4. Create a custom Dataset class
class MentalHealthDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

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

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

dataset = MentalHealthDataset(encodings, labels)

# 5. Split into train/val/test sets
total_size = len(dataset)
train_size = int(0.8 * total_size)
```

```
val_size = int(0.10 * total_size)
test_size = total_size - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])

# 6. Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)
test_loader = DataLoader(test_dataset, batch_size=16)

# 7. Define optimizer
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)
epoch = 5
num_training_steps = epoch * len(train_loader)
lr_scheduler = get_scheduler(
    name="linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=num_training_steps
)

# 8. Training function
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

def train(model, dataloader):
    model.train()
    total_loss = 0
    for batch in tqdm(dataloader, desc="Training"):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        total_loss += loss.item()
    return total_loss / len(dataloader)

# 9. Evaluation function
def evaluate(model, dataloader):
    model.eval()
    preds, true_labels = [], []
    with torch.no_grad():
        for batch in tqdm(dataloader, desc="Evaluating"):
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            logits = outputs.logits
            preds.extend(torch.argmax(logits, axis=1).cpu().numpy())
            true_labels.extend(batch["labels"].cpu().numpy())
    acc = accuracy_score(true_labels, preds)
    report = classification_report(true_labels, preds)
    return acc, report

# 10. Train the model
for epoch in range(8):
    print(f"\nEpoch {epoch + 1}")
    train_loss = train(model, train_loader)
    val_acc, val_report = evaluate(model, val_loader)
    print(f"Train loss: {train_loss:.4f}")
    print(f"Validation Accuracy: {val_acc:.4f}")
    print("Validation Report:\n", val_report)
```

```
→ /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
    warnings.warn(
tokenizer_config.json: 100%                                         25.0/25.0 [00:00<00:00, 2.71kB/s]
spiece.model: 100%                                              760k/760k [00:00<00:00, 12.3MB/s]
tokenizer.json: 100%                                             1.31M/1.31M [00:00<00:00, 27.0MB/s]
config.json: 100%                                              684/684 [00:00<00:00, 61.3kB/s]

Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falli
model.safetensors: 100%                                         47.4M/47.4M [00:00<00:00, 183MB/s]

Some weights of AlbertForSequenceClassification were not initialized from the model checkpoint at albert-base-v2 and are new
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Epoch 1

Training: 100%|██████████| 2634/2634 [03:43<00:00, 11.79it/s]
Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.44it/s]

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

Train loss: 0.8388  

Validation Accuracy: 0.6202  

Validation Report:
```

	precision	recall	f1-score	support
0	0.44	0.78	0.56	391
1	0.84	0.90	0.87	1650
2	0.51	0.85	0.64	1514
3	0.58	0.01	0.03	1087
4	0.00	0.00	0.00	250
5	0.69	0.53	0.60	279
6	1.00	0.11	0.20	97
accuracy			0.62	5268
macro avg	0.58	0.46	0.41	5268
weighted avg	0.62	0.62	0.54	5268

Epoch 2

Training: 100%|██████████| 2634/2634 [03:42<00:00, 11.84it/s]
Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.59it/s]

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

Train loss: 1.2037  

Validation Accuracy: 0.5634  

Validation Report:
```

	precision	recall	f1-score	support
0	0.24	0.37	0.29	391
1	0.75	0.82	0.78	1650
2	0.55	0.44	0.49	1514
3	0.49	0.74	0.59	1087
4	0.00	0.00	0.00	250
5	0.00	0.00	0.00	279
6	0.00	0.00	0.00	97
accuracy			0.56	5268
macro avg	0.29	0.34	0.31	5268
weighted avg	0.51	0.56	0.53	5268

Epoch 3

Training: 100%|██████████| 2634/2634 [03:42<00:00, 11.85it/s]
Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.49it/s]

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

Train loss: 1.4238
```

Validation Accuracy: 0.3132

Validation Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	391
1	0.31	1.00	0.48	1650
2	0.00	0.00	0.00	1514
3	0.00	0.00	0.00	1087
4	0.00	0.00	0.00	250
5	0.00	0.00	0.00	279
6	0.00	0.00	0.00	97
accuracy			0.31	5268
macro avg	0.04	0.14	0.07	5268
weighted avg	0.10	0.31	0.15	5268

Epoch 4

Training: 100%|██████████| 2634/2634 [03:42<00:00, 11.84it/s]

Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.54it/s]

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
```

Train loss: 1.6382

Validation Accuracy: 0.3132

Validation Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	391
1	0.31	1.00	0.48	1650
2	0.00	0.00	0.00	1514
3	0.00	0.00	0.00	1087
4	0.00	0.00	0.00	250
5	0.00	0.00	0.00	279
6	0.00	0.00	0.00	97
accuracy			0.31	5268
macro avg	0.04	0.14	0.07	5268
weighted avg	0.10	0.31	0.15	5268

Epoch 5

Training: 100%|██████████| 2634/2634 [03:42<00:00, 11.85it/s]

Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.55it/s]

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
```

Train loss: 1.6354

Validation Accuracy: 0.3132

Validation Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	391
1	0.31	1.00	0.48	1650
2	0.00	0.00	0.00	1514
3	0.00	0.00	0.00	1087
4	0.00	0.00	0.00	250
5	0.00	0.00	0.00	279
6	0.00	0.00	0.00	97
accuracy			0.31	5268
macro avg	0.04	0.14	0.07	5268
weighted avg	0.10	0.31	0.15	5268

Epoch 6

Training: 100%|██████████| 2634/2634 [03:42<00:00, 11.84it/s]

Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.55it/s]

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined when all predicted labels in a class are the same. Use 'average' and 'macro' to resolve this warning.
```

Train loss: 1.5918

Validation Accuracy: 0.5376

Validation Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.00	0.00	0.00	391
1	0.84	0.85	0.85	1650
2	0.40	0.95	0.56	1514
3	0.00	0.00	0.00	1087
4	0.00	0.00	0.00	250
5	0.00	0.00	0.00	279
6	0.00	0.00	0.00	97
accuracy			0.54	5268
macro avg	0.18	0.26	0.20	5268
weighted avg	0.38	0.54	0.43	5268

Epoch 7

Training: 100%|██████████| 2634/2634 [03:42<00:00, 11.85it/s]

Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.60it/s]

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
```

Train loss: 1.3660

Validation Accuracy: 0.4991

Validation Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	391
1	0.85	0.72	0.78	1650
2	0.37	0.95	0.53	1514
3	0.00	0.00	0.00	1087
4	0.00	0.00	0.00	250
5	0.00	0.00	0.00	279
6	0.00	0.00	0.00	97
accuracy			0.50	5268
macro avg	0.18	0.24	0.19	5268
weighted avg	0.37	0.50	0.40	5268

Epoch 8

Training: 100%|██████████| 2634/2634 [03:42<00:00, 11.85it/s]

Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.58it/s] Train loss: 1.3574

Validation Accuracy: 0.5321

Validation Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	391
1	0.73	0.90	0.81	1650
2	0.41	0.87	0.55	1514
3	0.00	0.00	0.00	1087
4	0.00	0.00	0.00	250
5	0.00	0.00	0.00	279
6	0.00	0.00	0.00	97
accuracy			0.53	5268
macro avg	0.16	0.25	0.19	5268
weighted avg	0.35	0.53	0.41	5268

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
```

```
# 11. Final test evaluation
test_acc, test_report = evaluate(model, test_loader)
print(f"\nFinal Test Accuracy: {test_acc:.4f}")
print("Test Report:\n", test_report)
```

Evaluating: 100%|██████████| 330/330 [00:09<00:00, 33.29it/s]

Final Test Accuracy: 0.5288

Test Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	339
1	0.73	0.90	0.80	1676
2	0.40	0.85	0.54	1503
3	0.00	0.00	0.00	1117
4	0.00	0.00	0.00	257
5	0.00	0.00	0.00	257
6	0.00	0.00	0.00	120
accuracy			0.53	5269
macro avg	0.16	0.25	0.19	5269
weighted avg	0.35	0.53	0.41	5269

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for labels with zero samples, return 0.0
  _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for labels with zero samples, return 0.0
  _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for labels with zero samples, return 0.0
  _warn_prf(average, modifier, f"{{metric.capitalize()}} is", len(result))
```

```
status_keys = list(status_mapping.keys())
```

```
def evaluate(model, dataloader, device):
    model.eval()
    true_labels = []
    predictions = []
    probs = []

    with torch.no_grad():
        for batch in dataloader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            logits = outputs.logits
            softmax_probs = torch.softmax(logits, dim=1)

            probs.extend(softmax_probs.cpu().numpy())
            predicted_labels = torch.argmax(softmax_probs, dim=1)
            predictions.extend(predicted_labels.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())

    print("Classification Report:")
    print(classification_report(true_labels, predictions))

    print("Confusion Matrix:")
    cm = confusion_matrix(true_labels, predictions)
    confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=status_mapping)
    fig, ax = plt.subplots(figsize=(11, 10))
    confusion_matrix_display.plot(ax=ax)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("ALBERT Confusion Matrix")
    plt.show()

# AUC Score (for multi-class)
try:

    n_classes = model.config.num_labels
    y_true_bin = label_binarize(true_labels, classes=list(range(n_classes)))
    auc = roc_auc_score(y_true_bin, probs, multi_class='ovr', average="micro")
    print(f"One-vs-Rest AUROC score: {auc:.4f}")
except Exception as e:
    print("AUC computation failed:", e)

# Optional: Plot ROC curves per class
try:
```

```
fpr = {}
tpr = {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], [p[i] for p in probs])

plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'{status_keys[i]}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for Each Class")
plt.legend(loc="lower right")
plt.show()

except Exception as e:
    print("ROC curve plotting failed:", e)

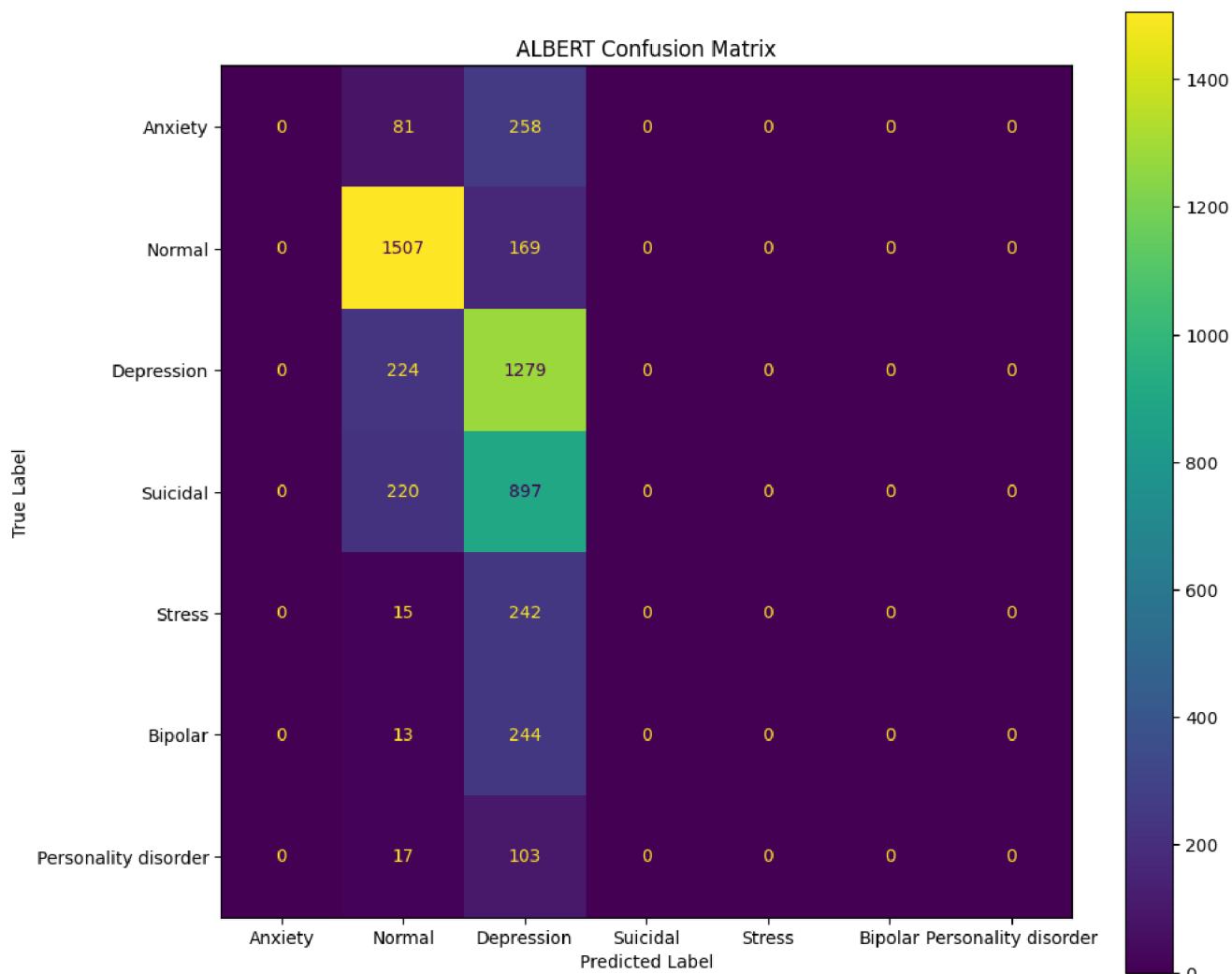
evaluate(model, test_loader, device)
```

Classification Report:

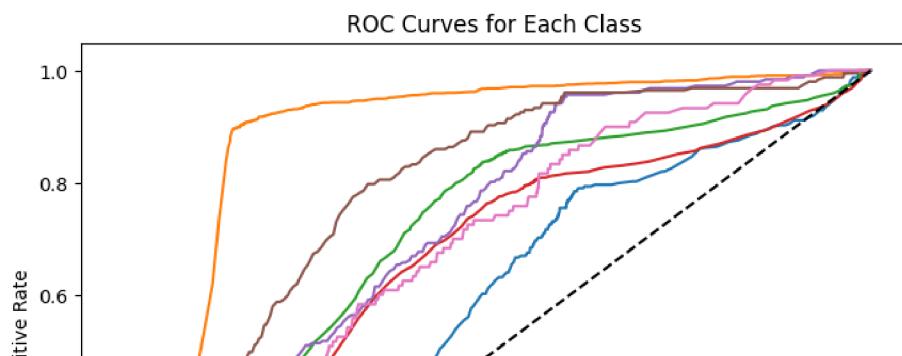
	precision	recall	f1-score	support
0	0.00	0.00	0.00	339
1	0.73	0.90	0.80	1676
2	0.40	0.85	0.54	1503
3	0.00	0.00	0.00	1117
4	0.00	0.00	0.00	257
5	0.00	0.00	0.00	257
6	0.00	0.00	0.00	120
accuracy			0.53	5269
macro avg	0.16	0.25	0.19	5269
weighted avg	0.35	0.53	0.41	5269

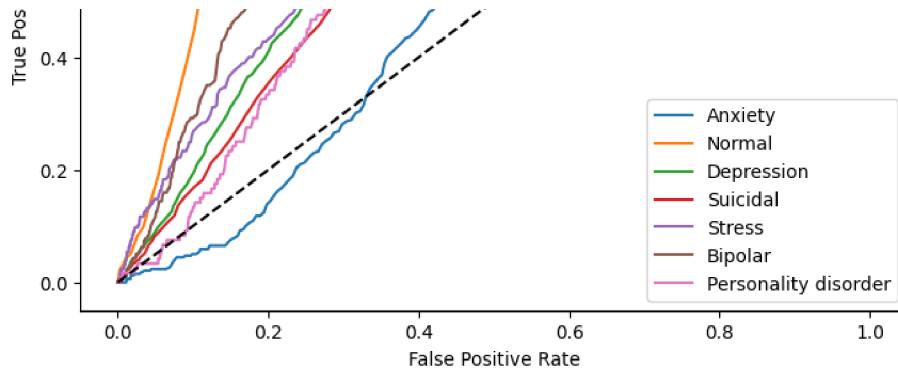
Confusion Matrix:

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for labels after binarization
  _warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for labels after binarization
  _warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined for labels after binarization
  _warn_prf(average, modifier, f'{metric.capitalize()} is', len(result))
```



One-vs-Rest AUROC score: 0.8404





▼ MentalBERT

Start coding or generate with AI.

```
# drop rows with 'NaN'
dataset = dataset.dropna(subset=['clean_text','labels'])

# truncate size per row to only 50 words
data_clean = [" ".join(entry.split(' ')[0:50]) for entry in list(dataset['clean_text'])]

# split data into training, validation, and testing datasets
trainX, testX, trainY, testY = train_test_split(data_clean, list(dataset['labels']),
                                                stratify=list(dataset['labels']), test_size=0.2, random_state=42)
valX, testX, valY, testY = train_test_split(testX, testY, stratify=testY, test_size=0.5, random_state=42)

# limit size for training
limit_size = 10000
print('original dataset size:',len(dataset), ' dataset size used for training',limit_size)
trainX = list(trainX[:limit_size])
trainY = list(trainY[:limit_size])
valX = list(valX[:limit_size])
valY = list(valY[:limit_size])

→ 52681 10000

# Create a tf dataset
tokenizer = AutoTokenizer.from_pretrained("mental/mental-bert-base-uncased", token='hf_FlaQEyISrChpurohLsxVf0WqCbAkmgUPiw')
train_encodings = tokenizer(trainX, is_split_into_words=False, return_offsets_mapping=True, padding=True, truncation=True, max_l
val_encodings = tokenizer(valX, is_split_into_words=False, return_offsets_mapping=True, padding=True, truncation=True, max_lengt

→ tensorflow is using GPU

#train_labels = encode_tags(tags, train_encodings)
train_labels = [[i] for i in trainY]
val_labels = [[i] for i in valY]

train_encodings.pop("offset_mapping") # we don't want to pass this to the model
val_encodings.pop("offset_mapping")

train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    train_labels
))
val_dataset = tf.data.Dataset.from_tensor_slices(
    dict(val_encodings),
    val_labels
))

mentalBERT_model = TFBertForSequenceClassification.from_pretrained("mental/mental-bert-base-uncased", from_pt=True,
                                                               num_labels=len(status_mapping), token='hf_FlaQEyISrChpurohLsxVf0WqCbAkmg
```

→ Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertForSequenceClassification: ['bert.e'
 - This IS expected if you are initializing TFBertForSequenceClassification from a PyTorch model trained on another task or w
 - This IS NOT expected if you are initializing TFBertForSequenceClassification from a PyTorch model that you expect to be ex
 Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not initialized from the PyTorch model and
 You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

```
optimizer = tf.optimizers.Adam(learning_rate=5e-5) # note the low learning rate
mentalBERT_model.compile(optimizer=optimizer, loss=mentalBERT_model.hf_compute_loss, metrics=['accuracy'], run_eagerly=True)
```

```
mentalBert_history = mentalBERT_model.fit(train_dataset.batch(8),
                                             epochs=8,
                                             batch_size=8,
                                             validation_data = val_dataset.batch(8))
```

→ Epoch 1/8
 1250/1250 [=====] - 1112s 873ms/step - loss: 0.8337 - accuracy: 0.6778 - val_loss: 0.7560 - val_acc
 Epoch 2/8
 1250/1250 [=====] - 1083s 867ms/step - loss: 0.5538 - accuracy: 0.7845 - val_loss: 0.7280 - val_acc
 Epoch 3/8
 1250/1250 [=====] - 1084s 867ms/step - loss: 0.3820 - accuracy: 0.8571 - val_loss: 0.8216 - val_acc
 Epoch 4/8
 1250/1250 [=====] - 1086s 869ms/step - loss: 0.2557 - accuracy: 0.9072 - val_loss: 0.8698 - val_acc
 Epoch 5/8
 1250/1250 [=====] - 1084s 867ms/step - loss: 0.1813 - accuracy: 0.9372 - val_loss: 1.0514 - val_acc
 Epoch 6/8
 1250/1250 [=====] - 1084s 868ms/step - loss: 0.1581 - accuracy: 0.9460 - val_loss: 1.1084 - val_acc
 Epoch 7/8
 1250/1250 [=====] - 1081s 865ms/step - loss: 0.1276 - accuracy: 0.9594 - val_loss: 1.1215 - val_acc
 Epoch 8/8
 1250/1250 [=====] - 1087s 870ms/step - loss: 0.0914 - accuracy: 0.9696 - val_loss: 1.1399 - val_acc

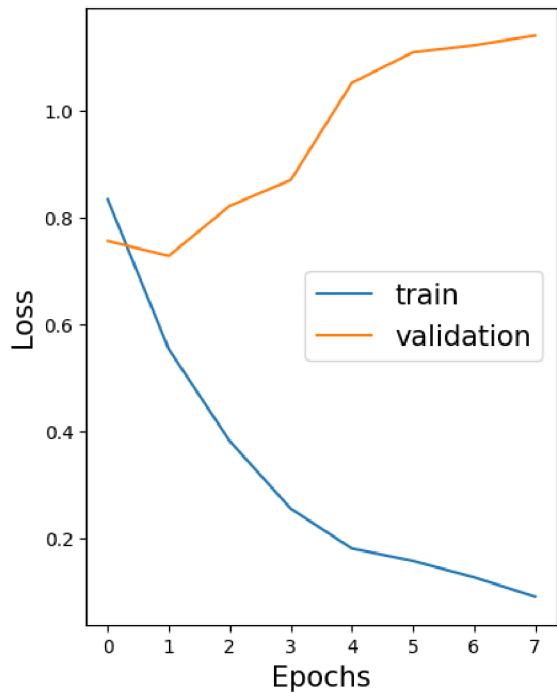
Plot results

```
# borrowed from Module 5 notebook
def summarize_training(history, ylim_loss=None, ylim_acc=None):
    """ plots loss curves and accuracy"""
    fig, ax = plt.subplots(1,2, figsize=(10, 6))
    # plot loss
    ax[0].set_title('Loss Curves', fontsize=20)
    ax[0].plot(history.history['loss'], label='train')
    ax[0].plot(history.history['val_loss'], label='validation')
    ax[0].set_xlabel('Epochs', fontsize=15)
    ax[0].set_ylabel('Loss', fontsize=15)
    if ylim_loss is not None: ax[0].set_ylim(ylim_loss[0], ylim_loss[1])
    ax[0].legend(fontsize=15)
    # plot accuracy
    ax[1].set_title('Classification Accuracy', fontsize=20)
    ax[1].plot(history.history['accuracy'], label='train')
    ax[1].plot(history.history['val_accuracy'], label='validation')
    ax[1].set_xlabel('Epochs', fontsize=15)
    ax[1].set_ylabel('Accuracy', fontsize=15)
    if ylim_acc is not None: ax[1].set_ylim(ylim_acc[0], ylim_acc[1])
    ax[1].legend(fontsize=15)

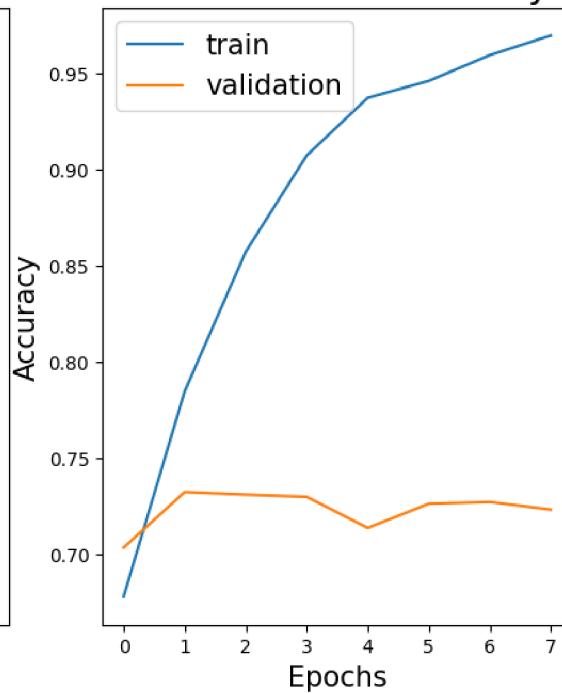
summarize_training(mentalBert_history)
```



Loss Curves



Classification Accuracy



```
# predict test results

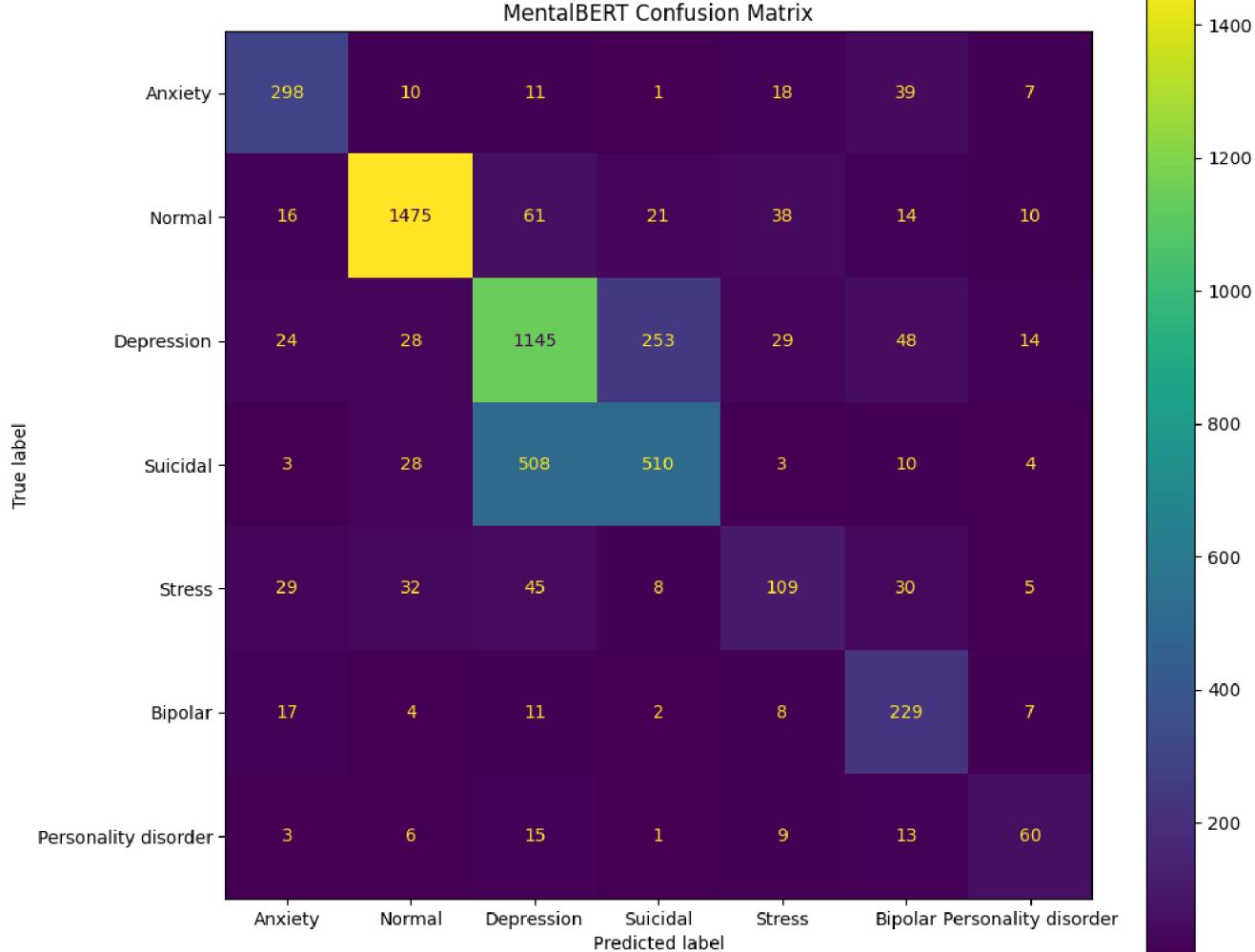
def softmax_prob_distribution(values):
    evals = [math.exp(v) for v in values]
    esum = sum(evals)
    return [eval/esum for eval in evals]

mentalBERTpreds_for_cm, mentalBERTpreds_for_auc = [], []
for string in testX:
    input_ids = tf.constant(tokenizer.encode(string[:512], add_special_tokens=True))[None, :]
    outputs = mentalBERT_model(input_ids)
    mentalBERTpreds_for_auc.append(softmax_prob_distribution(outputs[0].numpy()[0]))
    mentalBERTpreds_for_cm.append(np.argmax(outputs[0].numpy()[0]))

# confusion matrix
cm = confusion_matrix(testY[:len(mentalBERTpreds_for_cm)], mentalBERTpreds_for_cm, labels=list(status_mapping.values()))
fig, ax = plt.subplots(figsize=(11, 10))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=list(status_mapping.keys()))
disp.plot(ax=ax)
plt.title("MentalBERT Confusion Matrix")
plt.show()
```



MentalBERT Confusion Matrix



```
# test accuracy
accuracy = np.trace(cm) / np.sum(cm)

print(f"Accuracy: {accuracy}")

→ Accuracy: 0.7261339912696907
```

#AUC score

```
micro_roc_auc_ovr = roc_auc_score(
    testY[:len(mentalBERTpreds_for_auc)],
    mentalBERTpreds_for_auc,
    multi_class="ovr",
    average="micro",
)
```

```
print(f"One-vs-Rest AUROC score: {micro_roc_auc_ovr:.2f}")
```

```
→ One-vs-Rest AUROC score: 0.95
```

AUROC

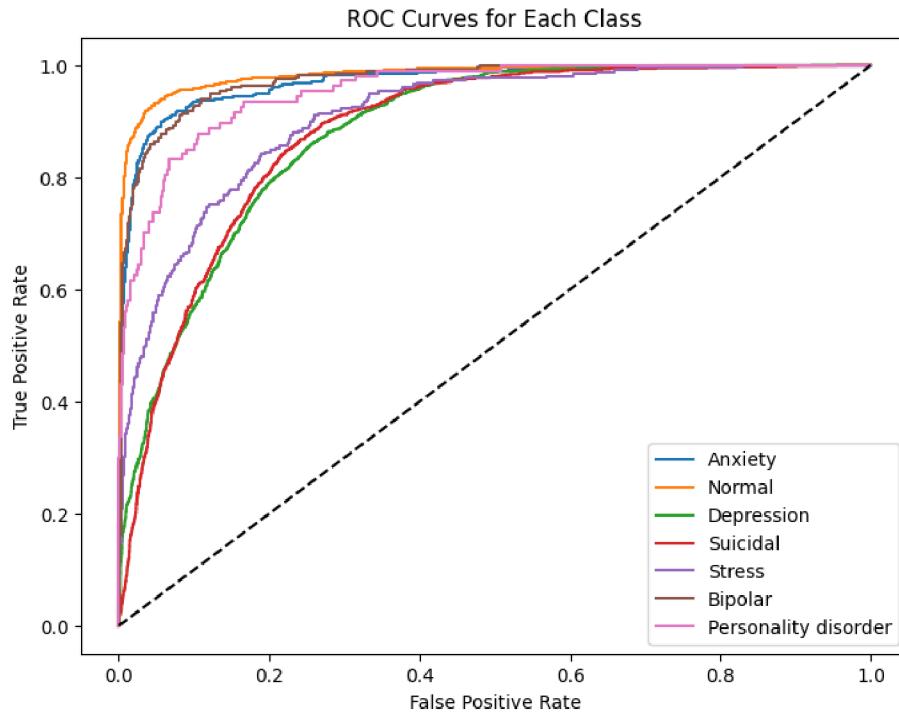
```
num_labels = len(status_mapping)
status_keys = list(status_mapping.keys())

label_binarizer = LabelBinarizer().fit(testY)
true_y = label_binarizer.transform(testY[:len(mentalBERTpreds_for_auc)])
# ROC Curve
fpr = dict()
tpr = dict()
plt.figure(figsize=(8,6))
for i in range(num_labels):
    print(tvpe(true_y[:,i]).tvpe(np.array(mentalBERTpreds_for_auc)[:,i]))
```

```
fpr[i], tpr[i], _ = roc_curve(true_y[:,i], np.array(mentalBERTpreds_for_auc)[:,i])
plt.plot(fpr[i], tpr[i], label=f'{status_keys[i]}')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Each Class')
plt.legend(loc="lower right")
plt.show()
```

→ <class 'numpy.ndarray'> <class 'numpy.ndarray'>
 <class 'numpy.ndarray'> <class 'numpy.ndarray'>



fastText

Start coding or generate with AI.

```
# Connect with drive
from google.colab import drive
drive.mount('/content/drive')

#Read dataset
folder_path = '/content/drive/MyDrive/COMP 652/Final Project/Colab Notebooks/Combined_Data.csv'
dataset = pd.read_csv(folder_path)
# dataset

→ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

# preprocess data
fasttext_dataset = dataset.dropna(subset=['clean_text']) # drop where clean_text is na
fasttext_dataset = fasttext_dataset.dropna(subset=['status']) # drop where status is na
fasttext_dataset['clean_text'] = fasttext_dataset['clean_text'].astype(str).replace('\n','') # replace any new lines in "clean_t
fasttext_dataset['status'] = fasttext_dataset['status'].astype(str).replace('\n','\r',regex=True) # replace any new lines in "stat
fasttext_dataset['status'] = fasttext_dataset['status'].replace(' ','_',regex=True) # replace any new lines in "status" column
fasttext_dataset = fasttext_dataset[fasttext_dataset['clean_text'] != ''] # remove any rows where "clean_text" has blank value

# declare the data and its labels
x = fasttext_dataset['clean_text']
y = fasttext_dataset['status']
```

```

# split the training and test datasets. 80% for training, 20% for test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# function to flatten nested lists. used for y_predictions
def flatten_list(x:list):
    lst = []
    for element in x:
        if isinstance(element, list):
            lst.extend(flatten_list(element))
        else:
            lst.append(element)
    return lst

# Hyperparameters
# 1st item: learning rate
# 2nd item: wordNgrams
# 3rd item: loss function (ns = negative sampling, hs = hierarchical softmax)
hyperparameters = {
    0:[.1, 1, 'ns'],
    1:[.5, 1, 'ns'],
    2:[1.0, 1, 'ns'],
    3:[.1, 2, 'ns'],
    4:[.5, 2, 'ns'],
    5:[1.0, 2, 'ns'],
    6:[.1, 3, 'ns'],
    7:[.5, 3, 'ns'],
    8:[1.0, 3, 'ns'],
    9:[.1, 1, 'hs'],
    10:[.5, 1, 'hs'],
    11:[1.0, 1, 'hs'],
    12:[.1, 2, 'hs'],
    13:[.5, 2, 'hs'],
    14:[1.0, 2, 'hs'],
    15:[.1, 3, 'hs'],
    16:[.5, 3, 'hs'],
    17:[1.0, 3, 'hs'],
}
}

# Validation Models with K Folds

kfolds = StratifiedKFold(n_splits=18, shuffle=True, random_state=42)

accuracies_negativesampling = []
accuracies_hierarchicalsoftmax = []

for idx, (train_index, val_index) in enumerate(kfolds.split(x_train, y_train)):

    # training sets
    x_train_fold = x_train.iloc[train_index]
    y_train_fold = y_train.iloc[train_index]

    # validation sets
    x_val_fold = x_train.iloc[val_index]
    y_val_fold = y_train.iloc[val_index]

    # concatenate the y and x data to match the fasttext supervised parameter format
    fasttext_model_data = '__label__' + y_train.astype(str) + ' ' + x_train.astype(str)

    # substitute any NaN values with ''
    fasttext_model_data = fasttext_model_data.fillna('')

    # output a text file of the fasttext training data, since the fasttext model can only read files
    fasttext_model_data.to_csv('/content/drive/MyDrive/COMP 652/Final Project/Colab Notebooks/fasttext_model_data.txt', index=False)

    # select parameters for hypertuning
    learning_rate = hyperparameters[idx][0]
    wordNgram = hyperparameters[idx][1]
    loss_function = hyperparameters[idx][2]

    # train the fasttext model with the text file
    fasttext_model = fasttext.train_supervised(input = '/content/drive/MyDrive/COMP 652/Final Project/Colab Notebooks/fasttext_model_data.txt',
                                                lr = learning_rate,
                                                wordNgrams = wordNgram,
                                                loss = loss_function)

```

```
# get predicted labels
y_predictions, _ = fasttext_model.predict(list(x_val_fold))
y_predictions = flatten_list(y_predictions)
y_predictions = [prediction.replace('__label__', '') for prediction in y_predictions]

# calculate the accuracy and append results to accuracies variable
num_correct = np.sum(y_predictions == y_val_fold) # number of matches found
accuracy = float(num_correct) / len(y_val_fold) # calculate the accuracy

if idx < 9:
    accuracies_negativesampling.append(accuracy)
else:
    accuracies_hierarchicalsoftmax.append(accuracy)

print(f"Fold {idx+1} learning rate={learning_rate}, wordNgram={wordNgram}, loss_function={loss_function}, accuracy={accuracy}
```

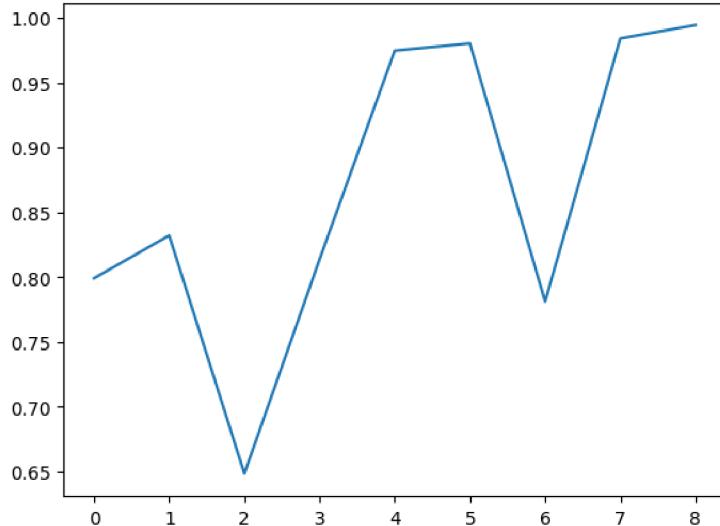
→ Fold 1 learning rate=0.1, wordNgram=1, loss_function=ns, accuracy=0.7992
 Fold 2 learning rate=0.5, wordNgram=1, loss_function=ns, accuracy=0.8322
 Fold 3 learning rate=1.0, wordNgram=1, loss_function=ns, accuracy=0.6485
 Fold 4 learning rate=0.1, wordNgram=2, loss_function=ns, accuracy=0.8138
 Fold 5 learning rate=0.5, wordNgram=2, loss_function=ns, accuracy=0.9747
 Fold 6 learning rate=1.0, wordNgram=2, loss_function=ns, accuracy=0.9803
 Fold 7 learning rate=0.1, wordNgram=3, loss_function=ns, accuracy=0.7812
 Fold 8 learning rate=0.5, wordNgram=3, loss_function=ns, accuracy=0.9842
 Fold 9 learning rate=1.0, wordNgram=3, loss_function=ns, accuracy=0.9944
 Fold 10 learning rate=0.1, wordNgram=1, loss_function=hs, accuracy=0.7791
 Fold 11 learning rate=0.5, wordNgram=1, loss_function=hs, accuracy=0.8292
 Fold 12 learning rate=1.0, wordNgram=1, loss_function=hs, accuracy=0.8253
 Fold 13 learning rate=0.1, wordNgram=2, loss_function=hs, accuracy=0.7984
 Fold 14 learning rate=0.5, wordNgram=2, loss_function=hs, accuracy=0.9717
 Fold 15 learning rate=1.0, wordNgram=2, loss_function=hs, accuracy=0.9782
 Fold 16 learning rate=0.1, wordNgram=3, loss_function=hs, accuracy=0.7987
 Fold 17 learning rate=0.5, wordNgram=3, loss_function=hs, accuracy=0.9829
 Fold 18 learning rate=1.0, wordNgram=3, loss_function=hs, accuracy=0.9949

```
# Accuracies for each loss function
# negative sampling
plt.plot(accuracies_negativesampling)
plt.title('Fasttext Validation Results: Negative Sampling Accuracy Results')
plt.show()

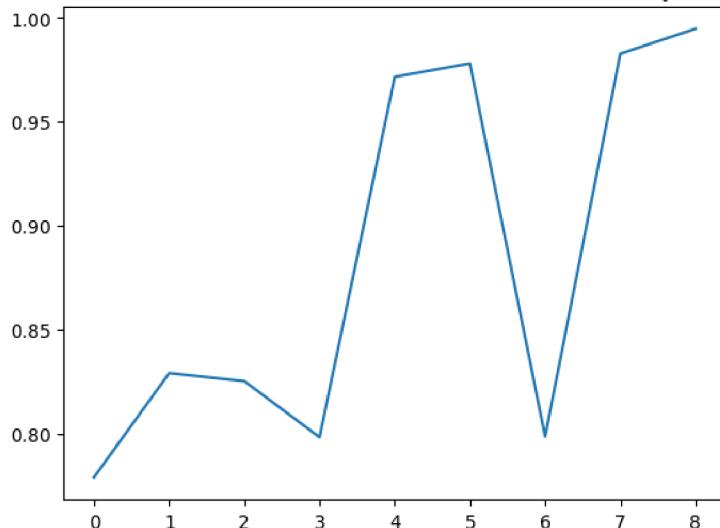
# hierarchical softmax
plt.plot(accuracies_hierarchicalsoftmax)
plt.title('Fasttext Validation Results: Hierarchical Softmax Accuracy Results')
plt.show()
```



Fasttext Validation Results: Negative Sampling Accuracy Results



Fasttext Validation Results: Hierarchical Softmax Accuracy Results



```
# Test Data Results
# With an accuracy of .9949, the best results came from fold 18, where the learning rate = 1.0, ngram = 3, and loss function = hs

# train the fasttext model with the text file
fasttext_model = fasttext.train_supervised(input = '/content/drive/MyDrive/COMP 652/Final Project/Colab Notebooks/fasttext_model'
                                            lr = 1.0,
                                            wordNgrams = 3,
                                            loss = 'hs')

# get predicted labels
y_predictions, _ = fasttext_model.predict(list(x_test))
y_predictions = flatten_list(y_predictions)
y_predictions = [prediction.replace('__label__', '') for prediction in y_predictions]

# calculate the accuracy and append results to accuracies variable
num_correct = np.sum(y_predictions == y_test) # number of matches found
test_accuracy = float(num_correct) / len(y_test) # calculate the accuracy

print(f"Test Accuracy Result: {test_accuracy:.4f}")
```

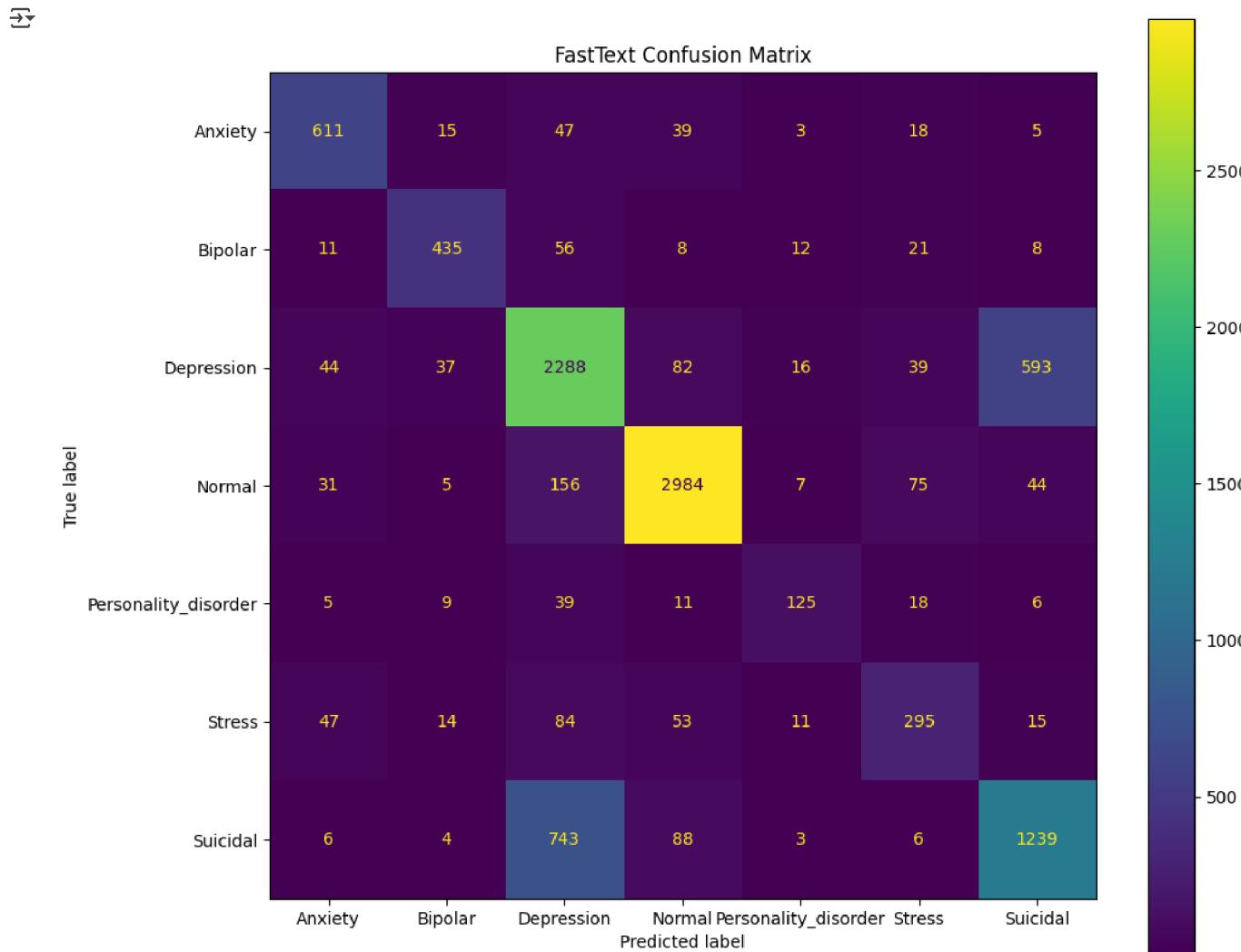
→ Test Accuracy Result: 0.7589

#Fasttext Confusion Matrix Results

```
# # calculate confusion matrix
labels=['Anxiety','Bipolar','Depression','Normal','Personality_disorder','Stress','Suicidal']

# confusion matrix
cm = confusion_matrix(y_test,y_predictions,labels=labels)
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=labels)
fig, ax = plt.subplots(figsize=(11, 10))
disp.plot(ax=ax)
plt.title("FastText Confusion Matrix")
plt.show()
```



```
# fastText AUC Score

statuses = ['Anxiety','Bipolar','Depression','Normal','Personality_disorder','Stress','Suicidal']

y_test = np.array(list(y_test))
y_predictions = np.array(list(y_predictions))

# encode string labels
le = LabelEncoder()
y_test_encoded = le.fit_transform(y_test)
y_predictions_bin = label_binarize(y_predictions, classes=le.classes_) # Use le.classes_ to ensure consistency

micro_roc_auc_ovr = roc_auc_score(
    y_test_encoded,
    y_predictions_bin,
    multi_class="ovr",
    average="micro",
)

print(f"One-vs-Rest AUROC Score: {micro_roc_auc_ovr:.4f}")

# Plot ROC curves per class

# add a key value pair to replace space character
status_mapping['Personality_disorder'] = 6
```

```
# encode y_test for the 1vrest AUROC plot
y_test_encoded = []
for label in y_test:
    encoded_label = status_mapping[label]
    y_test_encoded.append(encoded_label)
y_test_encoded = np.array(y_test_encoded)

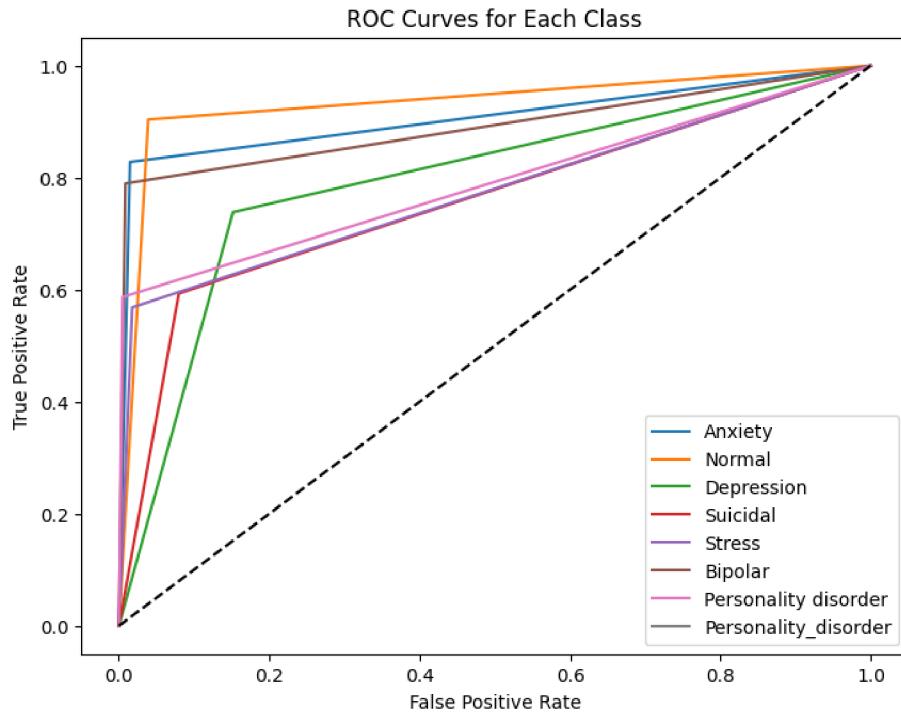
# encode y_predictions for the 1vrest AUROC plot
y_predictions_encoded = []
for label in y_predictions:
    encoded_label = status_mapping[label]
    y_predictions_encoded.append(encoded_label)
y_predictions_encoded = np.array(y_predictions_encoded)

num_labels = len(status_mapping)
status_keys = list(status_mapping.keys())

fpr = dict()
tpr = dict()
plt.figure(figsize=(8,6))
for i in range(num_labels):
    fpr[i], tpr[i], _ = roc_curve(
        (y_test_encoded == i).astype(int), # filter for specific class
        (y_predictions_encoded == i).astype(int) # filter for specific class
    )
    plt.plot(fpr[i], tpr[i], label=f'{status_keys[i]}')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Each Class')
plt.legend(loc="lower right")
plt.show()
```

→ /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning: No positive samples in y_t
warnings.warn(



XLNet

Start coding or generate with AI.

```
tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased')

→ /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
  The secret `HF_TOKEN` does not exist in your Colab secrets.
  To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set
  You will be able to reuse this secret in all of your notebooks.
  Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn()

# Subsample 20000 rows
SAMPLE_SIZE = 20000

dataset_small = dataset.sample(n=SAMPLE_SIZE, random_state=42).reset_index(drop=True)

# Prepare texts and labels
texts = dataset_small['clean_text'].astype(str).tolist()
labels = dataset_small['labels'].tolist()

# Train-test split
train_texts, test_texts, train_labels, test_labels = train_test_split(
    texts, labels, test_size=0.2, random_state=42
)

# Force to string
train_texts = [str(x) for x in train_texts]
test_texts = [str(x) for x in test_texts]

# Tokenize
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=64)
test_encodings = tokenizer(test_texts, truncation=True, padding=True, max_length=64)

# TensorFlow Datasets
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    train_labels
)).shuffle(1000).batch(8)

test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    test_labels
)).batch(8)

# Compile
model = TFXLNetForSequenceClassification.from_pretrained('xlnet-base-cased', num_labels=len(status_mapping))

optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = ['accuracy']

model.compile(optimizer=optimizer, loss=loss, metrics=metrics)

→ /usr/local/lib/python3.11/dist-packages/tf_keras/src/initializers/initializers.py:121: UserWarning: The initializer Truncate
  warnings.warn()
Some layers from the model checkpoint at xlnet-base-cased were not used when initializing TFXLNetForSequenceClassification:
- This IS expected if you are initializing TFXLNetForSequenceClassification from the checkpoint of a model trained on anothe
- This IS NOT expected if you are initializing TFXLNetForSequenceClassification from the checkpoint of a model that you expe
Some layers of TFXLNetForSequenceClassification were not initialized from the model checkpoint at xlnet-base-cased and are n
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

# Train

# Start timing
start_time = time.time()

# Train Model with history tracking
history = model.fit(
    train_dataset,
    epochs=8,
    validation_data=test_dataset,
    verbose=1
)

# End timing
end_time = time.time()
total_training_time = end_time - start_time
```

```

for i in range(len(history.history['accuracy'])):
    train_acc = history.history['accuracy'][i]
    val_acc = history.history['val_accuracy'][i]
    print(f"Epoch {i+1}: Train Acc = {train_acc:.4f}, Val Acc = {val_acc:.4f}")

final_train_acc = history.history['accuracy'][-1]
final_val_acc = history.history['val_accuracy'][-1]

print(f"Final Training Accuracy: {final_train_acc:.4f}")
print(f"Final Validation Accuracy: {final_val_acc:.4f}")

Epoch 1/8
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_for_sequence_classification/transformer/mask_emb:0'] when
2000/2000 [=====] - 179s 67ms/step - loss: 0.8909 - accuracy: 0.6612 - val_loss: 0.7180 - val_accuracy: 0.7180
Epoch 2/8
2000/2000 [=====] - 118s 59ms/step - loss: 0.6024 - accuracy: 0.7629 - val_loss: 0.6818 - val_accuracy: 0.6818
Epoch 3/8
2000/2000 [=====] - 116s 58ms/step - loss: 0.4663 - accuracy: 0.8142 - val_loss: 0.6874 - val_accuracy: 0.6874
Epoch 4/8
2000/2000 [=====] - 118s 59ms/step - loss: 0.3648 - accuracy: 0.8579 - val_loss: 0.7098 - val_accuracy: 0.7098
Epoch 5/8
2000/2000 [=====] - 117s 59ms/step - loss: 0.2753 - accuracy: 0.8933 - val_loss: 0.8087 - val_accuracy: 0.8087
Epoch 6/8
2000/2000 [=====] - 116s 58ms/step - loss: 0.2206 - accuracy: 0.9165 - val_loss: 0.8684 - val_accuracy: 0.8684
Epoch 7/8
2000/2000 [=====] - 116s 58ms/step - loss: 0.1703 - accuracy: 0.9373 - val_loss: 0.9370 - val_accuracy: 0.9370
Epoch 8/8
2000/2000 [=====] - 117s 59ms/step - loss: 0.1413 - accuracy: 0.9511 - val_loss: 0.9664 - val_accuracy: 0.9664
Epoch 1: Train Acc = 0.6612, Val Acc = 0.7147
Epoch 2: Train Acc = 0.7629, Val Acc = 0.7483
Epoch 3: Train Acc = 0.8142, Val Acc = 0.7607
Epoch 4: Train Acc = 0.8579, Val Acc = 0.7535
Epoch 5: Train Acc = 0.8933, Val Acc = 0.7480
Epoch 6: Train Acc = 0.9165, Val Acc = 0.7480
Epoch 7: Train Acc = 0.9373, Val Acc = 0.7470
Epoch 8: Train Acc = 0.9511, Val Acc = 0.7600
Final Training Accuracy: 0.9511
Final Validation Accuracy: 0.7600

```

Total Training Time: 997.61 seconds

--- Summarize Training Curves ---

```

def summarize_training(history):
    num_epochs = len(history.history['loss'])
    fig, ax = plt.subplots(1, 2, figsize=(14, 5))

    ax[0].plot(range(1, num_epochs+1), history.history['loss'], label='Training Loss')
    ax[0].plot(range(1, num_epochs+1), history.history['val_loss'], label='Validation Loss')
    ax[0].set_title('Epochs vs Loss', fontsize=16)
    ...  
...  
...

```