Mental Health Classification

Chue Wai Wai Phyo - Group 20

Agenda

- Dataset Selection & Preprocessing
- Implementation & Use of Pre-trained Model using Hugging Face
- Results & insights
- Conclusion
- Reference

01

Dataset Selection & Preprocessing

```
# Check the structure and summary
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53043 entries, 0 to 53042
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53043 non-null  int64
 1   statement   52681 non-null  object
 2   status      53043 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.2+ MB
None
         Unnamed: 0
count  53043.000000
mean   26521.000000
std    15312.339501
min        0.000000
25%    13260.500000
50%    26521.000000
75%    39781.500000
max    53042.000000
```

```
#Checking how the data look like
df.head()
```

	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

```
df['status'].value_counts()
```

status	count
Normal	16351
Depression	15404
Suicidal	10653
Anxiety	3888
Bipolar	2877
Stress	2669
Personality disorder	1201

dtype: int64

```
df.shape
```

```
(53043, 3)
```

```
#Dropping the unwanted column
data = df.drop(columns=["Unnamed: 0"])
```

```
#Checking the missing data
data.isnull().sum()
```

	0
statement	362
status	0

dtype: int64

```
#Dropping the missing data
data = data.dropna()
```

```
#Checking the duplicated data
data.duplicated().sum()
```

```
1588
```

```
#Dropping the duplicated data
data.drop_duplicates(inplace=True)
```

```
#Checking the Final Shape
data.shape
```

```
(51093, 2)
```

```
#Resets the index of the DataFrame after dropping rows.
data = data.reset_index(drop=True)
```

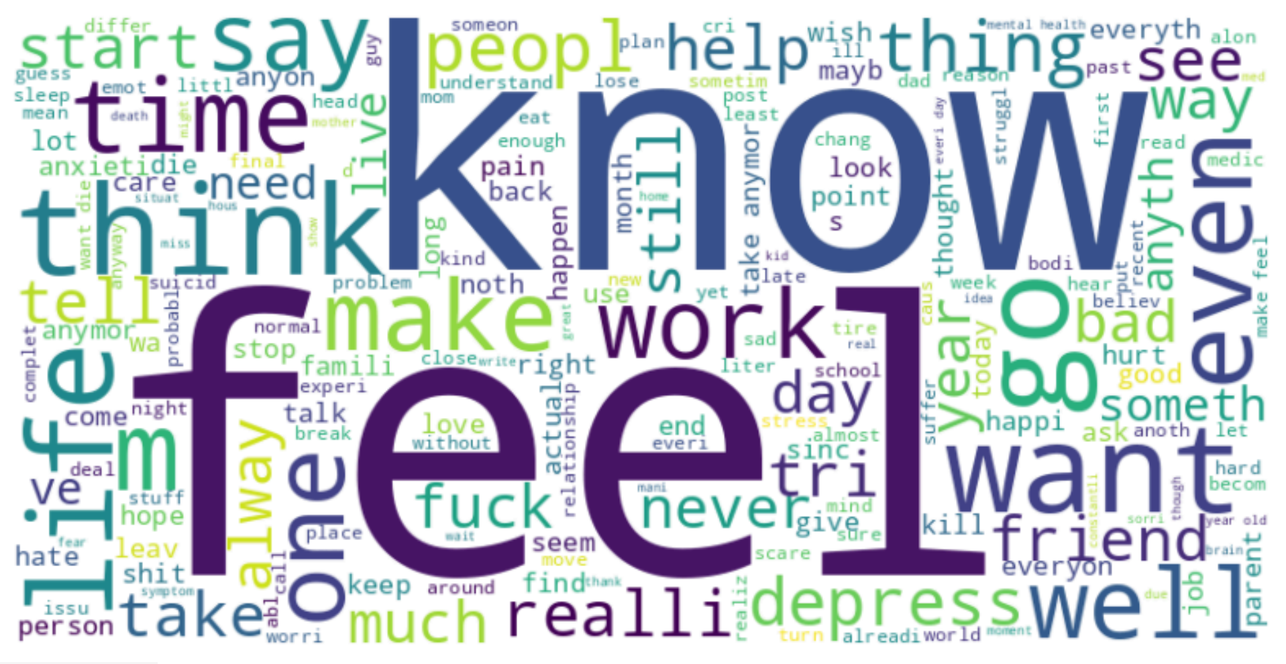
```
[ ] sample_df= data.sample(frac=1/4, random_state = 42).reset_index(drop=True)
```

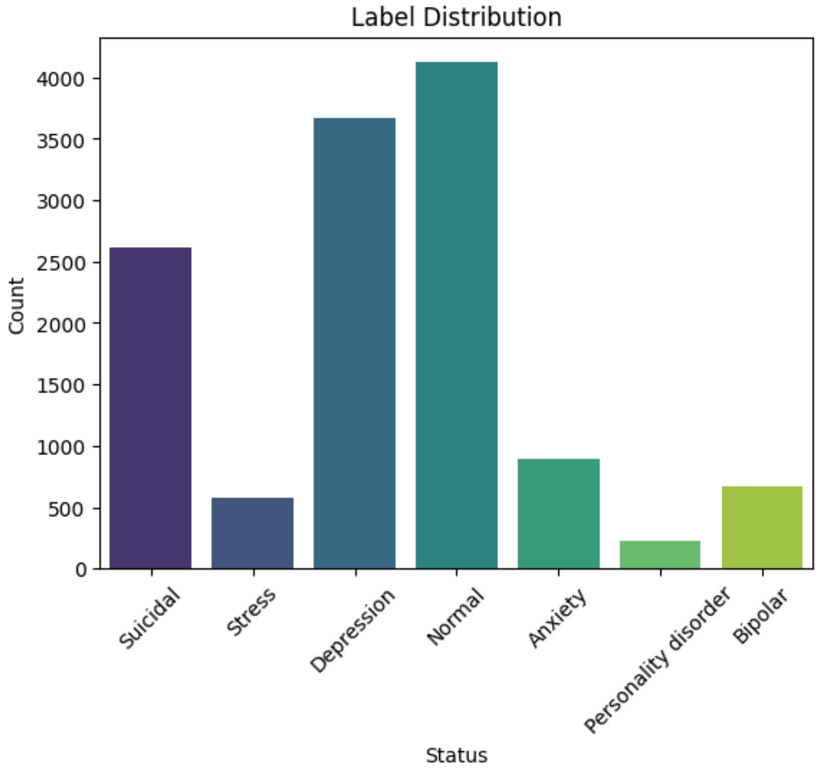
```
import re
def remove_stop_words(text):
    # Tokenize the text and filter out stop words
    words = [word for word in text.split() if word.lower() not in stop_words]
    return ' '.join(words)
def lemmatize_text(text):
    doc = nlp(text)  # Process text with SpaCy
    lemmatized_text = ' '.join([token.lemma_ for token in doc if not token.is_punct])  # Skip
    return lemmatized_text
def stem_word(text):  # Stemming
    stemmer = nltk.stem.PorterStemmer()
    words = [stemmer.stem(word) for word in text.split()]
    return ' '.join(words)
def clean_text(text):
    """Clean text by removing URLs, special characters, and numbers."""
    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)  # Remove URLs
    text = re.sub(r'\@\w+|\#', '', text)  # Remove mentions and hashtags
    text = re.sub(r"[^a-zA-Z\s]", '', text)  # Remove special characters and numbers
    return text.lower().strip()  # Convert to lowercase and strip whitespace

def preprocess_text(text):
    """Apply all preprocessing steps to the text."""
    text = clean_text(text)            # Step 1: Clean the text
    text = remove_stop_words(text)     # Step 2: Remove stop words
    text = lemmatize_text(text)        # Step 3: Lemmatize text
    text = stem_word(text)             # Step 4: Apply stemming
    return text

# Apply preprocessing to the DataFrame
df = sample_df.copy()
for i, sentence in enumerate(tqdm(sample_df["statement"], desc="Processing Text")):
    df.loc[i, "statement"] = preprocess_text(sentence)
```

```
Processing Text: 100%|██████████| 12773/12773 [03:03<00:00, 69.63it/s]
```





02

Implementation & Use of Pre-trained Model using Hugging Face

```
[ ] from transformers import AutoTokenizer,AutoModelForCausalLM,pipeline
    import torch
```

```
GPU is available: Tesla T4
```

```
[ ] # Check GPU availability
    import torch
    import tensorflow as tf
    import spacy
    from transformers import AutoModel, AutoTokenizer

    if torch.cuda.is_available():
        print(f"PyTorch GPU: {torch.cuda.get_device_name(0)}")
    else:
        print("PyTorch GPU not available.")

    print(f"TensorFlow GPUs: {len(tf.config.list_physical_devices('GPU'))}")
    spacy.prefer_gpu()
    print("SpaCy is using GPU:", spacy.require_gpu())

    # PyTorch Example
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    tensor = torch.randn(3, 3).to(device)
    print("PyTorch Tensor:", tensor)

    # TensorFlow Example
    with tf.device('/GPU:0'):
        result = tf.matmul(tf.constant([[1.0, 2.0]]), tf.constant([[3.0], [4.0]]))
    print("TensorFlow Result:", result)

    # Hugging Face Example
    tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
    model = AutoModel.from_pretrained("bert-base-uncased").to(device)
    inputs = tokenizer("Hello, GPU!", return_tensors="pt").to(device)
    outputs = model(**inputs)
    print("Hugging Face Output:", outputs.last_hidden_state.shape)

    # SpaCy Example
    nlp = spacy.load("en_core_web_sm")
    doc = nlp("GPU-powered text processing!")
    print("SpaCy Tokens:", [token.text for token in doc])
```

```
[ ] PyTorch GPU: Tesla T4
    TensorFlow GPUs: 1
    SpaCy is using GPU: True
    PyTorch Tensor: tensor([[-1.3681, -0.7114,  0.6280],
            [-0.7865,  2.0975, -1.6050],
            [-1.6355,  0.5405, -1.4189]], device='cuda:0')
    TensorFlow Result: tf.Tensor([[11.]], shape=(1, 1), dtype=float32)
    tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 3.43kB/s]
    config.json: 100% 570/570 [00:00<00:00, 36.5kB/s]
    vocab.txt: 100% 232k/232k [00:00<00:00, 2.58MB/s]
    tokenizer.json: 100% 466k/466k [00:00<00:00, 27.8MB/s]
    model.safetensors: 100% 440M/440M [00:02<00:00, 187MB/s]
    Hugging Face Output: torch.Size([1, 7, 768])
    SpaCy Tokens: ['GPU', '-', 'powered', 'text', 'processing', '!']
```

```
# Downloading model
model_id = "kingabzpro/Llama-3.1-8B-Instruct-Mental-Health-Classification"

tokenizer = AutoTokenizer.from_pretrained(model_id)

model = AutoModelForCausalLM.from_pretrained(
        model_id,
        return_dict=True,
        low_cpu_mem_usage=True,
        torch_dtype=torch.float16,
        device_map="auto",
        trust_remote_code=True,
)
```

```
tokenizer_config.json: 100% 50.9k/50.9k [00:00<00:00, 994kB/s]
tokenizer.json: 100% 9.09M/9.09M [00:00<00:00, 39.9MB/s]
special_tokens_map.json: 100% 296/296 [00:00<00:00, 6.99kB/s]
config.json: 100% 914/914 [00:00<00:00, 19.2kB/s]
model.safetensors.index.json: 100% 23.9k/23.9k [00:00<00:00, 500kB/s]
Downloading shards: 100% 4/4 [25:44<00:00, 332.96s/it]
model-00001-of-00004.safetensors: 100% 4.98G/4.98G [07:58<00:00, 10.5MB/s]
```

```python
# Downloading model
model_id = "kingabzpro/Llama-3.1-8B-Instruct-Mental-Health-Classification"

tokenizer = AutoTokenizer.from_pretrained(model_id)

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    return_dict=True,
    low_cpu_mem_usage=True,
    torch_dtype=torch.float16,
    device_map="auto",
    trust_remote_code=True,
)


text = "I can't sleep at night. I think about my past decisions and blame myself for it."
prompt = f"""Classify the text into Normal, Depression, Suicidal, Anxiety, Bipolar, Stress,
Personality disorder and return the answer as the corresponding mental health disorder label.
text: {text}
label: """.strip()

# Update: Using 'generate' directly instead of pipeline
# We call 'generate' method directly instead of using pipeline
# with 'text-generation' task which appears to be causing issues with 'prefix'.
input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to(model.device)
outputs = model.generate(input_ids, max_new_tokens=2, do_sample=True, temperature=0.1)
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

print(generated_text)
```

```
label: Depression
```

03

Results & insights

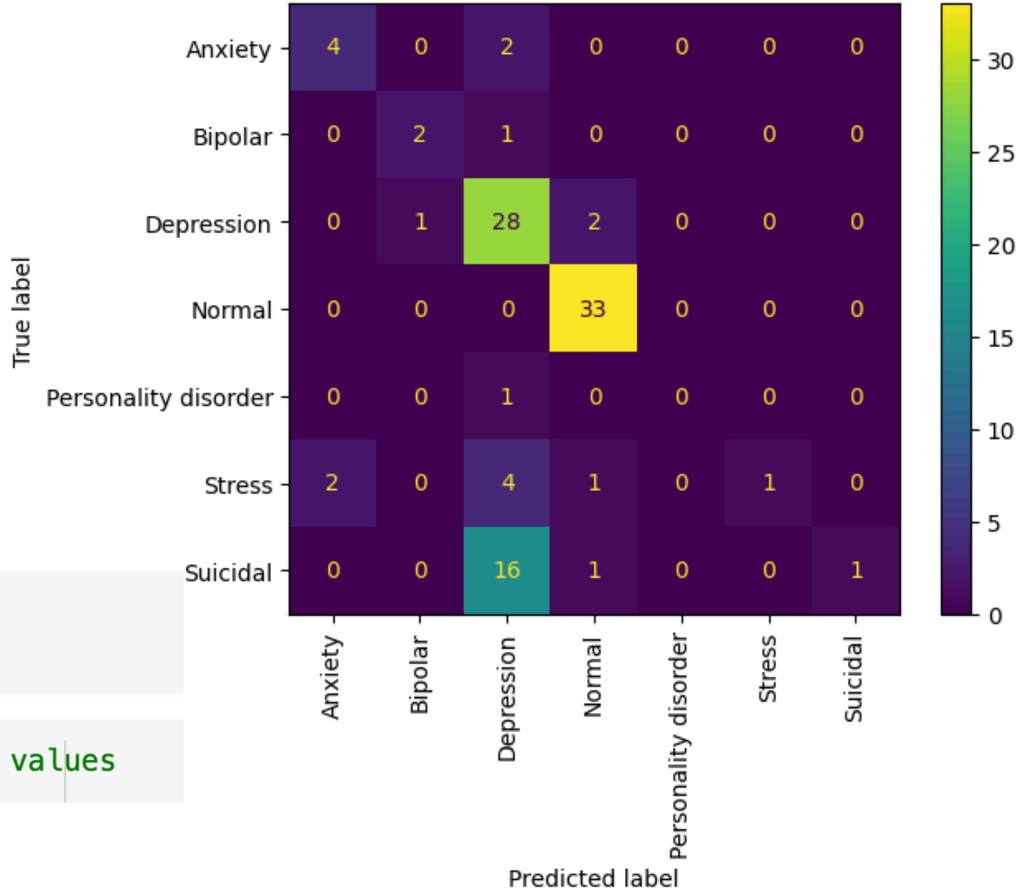
```python
# Replace incorrect predictions
y_pred = ["Suicidal" if label == "Suic" else label for label in y_pred]
```

```python
# Accuracy check
print("Accuracy score:", accuracy_score(sample_df["status"][:100], y_pred))
print("Classification report: \n", classification_report(sample_df["status"][:100], y_pred))
```

```
Accuracy score: 0.69
Classification report:
                      precision    recall  f1-score   support

             Anxiety       0.67      0.67      0.67         6
             Bipolar       0.67      0.67      0.67         3
          Depression       0.54      0.90      0.67        31
              Normal       0.89      1.00      0.94        33
Personality disorder       0.00      0.00      0.00         1
              Stress       1.00      0.12      0.22         8
            Suicidal       1.00      0.06      0.11        18

            accuracy                           0.69       100
           macro avg       0.68      0.49      0.47       100
        weighted avg       0.78      0.69      0.62       100
```




```python
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score, classification_report
```

```python
y_pred = df["statement"][:100].apply(get_label) # Only predict the first 100 values
```

The classification model was evaluated on a random 25% sample of the data, resulting in an accuracy of 69%.

- **Normal is Predicted Well**: (33 correct predictions)
- **Depression Shows Good Results**: Out of 31 true instances of "Depression," 28 are classified correctly. 1 case is misclassified as "Bipolar," and 2 as "Normal."
- **Misclassification in "Suicidal" Cases**: Of the true "Suicidal" cases, 16 are misclassified as "Depression." Only 1 instance of "Suicidal" is correctly identified.
- **Stress and Personality Disorder**: "Stress" and "Personality Disorder" have several misclassifications. For example, "Stress" cases are distributed across various other labels.
- **Rare Categories**: "Anxiety," "Bipolar," and "Suicidal" show relatively low correct predictions and high misclassifications. Smaller sample sizes in these categories might be contributing to the lower accuracy.

04

Conclusion

- The model shows reasonable accuracy (69%)
- The model performed well for some classes, particularly Normal and Depression, but struggled with others, especially for Personality Disorder, Stress, and Suicidal.
- This indicates that the model is sensitive to the distribution of classes(class imbalances), with stronger performance on majority classes and weaker results for minority ones.
- The classification report reveals imbalanced precision and recall values across different classes, highlighting areas that need improvement for better predictive performance.
- The model needs improvements, particularly for minority classes.

05

Reference

Hamza, R. (2024). *Large Language Models Driven Projects in the Real World* [Lecture]. Python for Data Science, Tokyo International University.

Hamza, R. (2024). *Sentiment Analysis* [Lecture]. Python for Data Science, Tokyo International University.

Soliman, A. S. (n.d.). *Sentiment analysis for mental health: Combined data* [Dataset]. Hugging Face. Retrieved from https://huggingface.co/datasets/AhmedSSoliman/sentiment-analysis-for-mental-health-Combined-Data

Kingabzpro. (n.d.). *Llama-3.1-8B-Instruct-Mental-Health-Classification* [Machine learning model]. Hugging Face. Retrieved from https://huggingface.co/kingabzpro/Llama-3.1-8B-Instruct-Mental-Health-Classification

OpenAI. (2024). *ChatGPT* (November 2024 version) [Large language model]. Retrieved from https://chat.openai.com/

Thank You!

Any Questions?