Table of Contents

Question 1 a

Import the Dataset

```
[2]  # Mount google drive
     drive.mount('/content/drive')

     # Load the dataset
     data_dir = "/content/drive/MyDrive/Colab Notebooks/DAML Assignment/mentalhealth.csv"
     df = pd.read_csv(data_dir, header=0)
```

```
Mounted at /content/drive
```

Figure 1.1.1 Importing the Dataset

After importing the crucial libraries into the program, the dataset needs to be loaded to conduct the further data analysis. As shown in Figure 1.1.1, the dataset is mounted from Google Drive into Google Colab platform, then it is converted into a dataframe for initial inspection and exploration.

Data Exploration

```
     # Look at the shape of dataset
     print(df.shape)

     # Checking out the data types
     print(df.info())
```

```
(53047, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53047 entries, 0 to 53046
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53044 non-null  object
 1   statement   52681 non-null  object
 2   status      53016 non-null  object
dtypes: object(3)
memory usage: 1.2+ MB
None
```

```
[82]  # Overview of the dataset
      print(df.head(), "\n")
      print(df.tail())
```

```
  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

       Unnamed: 0                                          statement   status
53042      53038  Nobody takes me seriously I’ve (24M) dealt wit...  Anxiety
53043      53039  selfishness  "I don't feel very good, it's lik...  Anxiety
53044      53040  Is there any way to sleep better? I can't slee...  Anxiety
53045      53041  Public speaking tips? Hi, all. I have to give ...  Anxiety
53046      53042  I have really bad door anxiety! It's not about...  Anxiety
```

Figure 1.1.2 Data Exploration Code Snippet and Result

First, we explore the dataset to get an overview of the information, as presented in Figure 1.1.2. Using *.shape* and *.info()* function, the dataset dimension and column details are shown. The *.head()* and *.tail()* function outputs a preview of the first few and last rows in the dataset. This data exploration step allows us to take an overview of the data and its structure.

Missing Values

```python
df_clean = df.copy()

# Total of missing values
print(df_clean.isnull().sum())

# Which columns contain missing values
missing_rows = df_clean[df_clean.isnull().any(axis=1)]
print("\nRows with missing values: ")
missing_rows
```

```
Unnamed: 0      3
statement     366
status         31
dtype: int64

Rows with missing values:
```

	Unnamed: 0	statement	status
6	6	I feel scared, anxious, what can I do? And may...	NaN
7	7	Have you ever felt nervous but didn't know why?	NaN
8	8	I haven't slept well for 2 days, it's like I'm...	NaN
293	293	NaN	Anxiety
572	572	NaN	Anxiety
...	...	...	...
52940	52936	NaN	Anxiety
53014	53010	NaN	Anxiety
53035	53031	NaN	Anxiety
53040	53036	someone help me think rationally I know I soun...	NaN
53041	53037	Lorazepam side effect/after effect In the last...	NaN

```python
# Identify if there are rows that contain more than one missing values
missing_rows_two = df_clean.loc[:, 'Unnamed:0':'status'].isnull().sum(axis=1) >=2

# Filter and display the rows
df_with_two_missing = df_clean[missing_rows_two]

print("Rows that contain at least two missing values:")
df_with_two_missing
```

```
Rows that contain at least two missing values:
```

	Unnamed: 0	statement	status
13294	NaN	NaN	NaN
13295	NaN	NaN	NaN
13296	NaN	NaN	NaN
13297	Suicidal	NaN	NaN

Figure 1.1.3 Identifying Missing Values

After conducting data exploration, we perform data cleaning by examining missing values with *.isnull()* function shown through Figure 1.1.3. In this step, rows with at least one missing value are printed. A further analysis to detect rows with two or more missing values was also performed to identify the reason of having missing values in the *'Unnamed: 0'* column, which act as the unique identifier of the initial dataset. This resulted an irregular empty gap from row 13294 to row 13296, along with an unusual 'Suicidal' word in the index column. After inspecting the original dataset, we have an agreement that it may be a misalignment.

```python
# Fix the misalignment issue
# Relocate the status 'Suicidal' back to row 13293
df_clean.loc[13293, 'status'] = 'Suicidal'

# Drop empty gap rows (where 'statement' and 'status' column are empty)
df_clean = df_clean.dropna(subset=['statement', 'status'], how='all')

# Check again if empty gaps are removed
df_clean.loc[13290:13300]
```

	Unnamed: 0	statement	status
13290	13290	tw: death"Existential dread" I have heard the ...	Depression
13291	13291	You once dreamed of being a writer. You once d...	Suicidal
13292	13292	I play college soccer which is my only accompl...	Suicidal
13293	13293	I cannot TAKE IT ANYMORE. I cannot TAKE IT ANY...	Suicidal
13298	13294	I cannot TAKE IT ANYMORE. I cannot TAKE IT ANY...	Depression
13299	13295	Oh Why are there so many members of this commu...	Suicidal
13300	13296	I am going through a divorce right now, I was ...	Depression

```python
# Print the shape to check the dataset
print("Original dataset shape:", df.shape)
print("Cleaned dataset shape:", df_clean.shape)
```

```
Original dataset shape: (53047, 3)
Cleaned dataset shape: (53043, 3)
```

```python
# Drop the initial index
df_clean = df_clean.drop(columns=['Unnamed: 0'])
df_clean
```

	statement	status
0	oh my gosh	Anxiety
1	trouble sleeping, confused mind, restless hear...	Anxiety
2	All wrong, back off dear, forward doubt. Stay ...	Anxiety
3	I've shifted my focus to something else but I'...	Anxiety
4	I'm restless and restless, it's been a month n...	Anxiety
...	...	...
53042	Nobody takes me seriously I've (24M) dealt wit...	Anxiety
53043	selfishness "I don't feel very good, it's lik...	Anxiety
53044	Is there any way to sleep better? I can't slee...	Anxiety
53045	Public speaking tips? Hi, all. I have to give ...	Anxiety
53046	I have really bad door anxiety! It's not about...	Anxiety

Figure 1.1.4 Handling Missing Values

To address this misalignment issue, we first dropped the empty rows between row 13294 and 13296, then realign the 'Suicidal' status back to row 13293. Rows 13290 to 13300 are printed out to assure the misalignment is solved. Also, the dataset before and after relocation is shown for comparison of the structure.

Then, the initial index, '*Unnamed: 0*' column, is dropped using the *drop()* function before handling the missing values. As this column does not hold any relevant information, it is removed to ensure the dataset only holds relevant features for further analysis, and does not affect the performance of mode imputation later.

```python
# Impute missing values for 'statement' column
df_clean['statement'] = df_clean.apply(
    lambda row: row['statement'] if pd.notna(row['statement'])  # If 'statement' is filled, make no changes
    else (
        row['statement'] if pd.isna(row['status'])  # If 'status' is missing, make no changes
        # If 'status' is filled, impute mode 'statement' according to 'status' group
        else df_clean[df_clean['status'] == row['status']]['statement'].mode()[0]
    ),axis=1
)

# Check for any missing values after imputation
df_clean.isna().sum()
```

```
              0
statement     0
status       26
dtype: int64
```

```python
# Calculate the overall mode of the 'status' column
status_mode = df_clean['status'].mode()[0]

# Impute missing 'status' values with the overall mode
df_clean['status'] = df_clean['status'].fillna(status_mode)

# Check for any missing values after imputation
df_clean.isna().sum()
```

```
              0
statement     0
status        0
dtype: int64
```

Figure 1.1.5 Mode Imputation for Missing Values

We first conducted a mode imputation to the ‘statement’ column, as the ‘status’ column depends on this column to make classifications. Figure 1.1.5 shows the conditional approach we implemented to handle missing values in ‘statement’ column:

- If ‘statement’ is present, imputation is skipped.
- If both ‘statement’ and ‘status’ are empty, no imputation is made.
- If ‘statement’ is empty but ‘status’ is present, impute the mode statement according to the ‘status’ category.

Similarly, an overall mode imputation is also used to handle missing values in the ‘status’ column, which represents a common and fast approach for categorical data (Memon, Wamala and Kabano, 2023). The total of imputation made to the ‘status’ column is 26/53042 values (0.04%), indicating the risk of potential bias is minimal. While this method may still introduce noise and inconsistencies if the imputed values do not match the actual sentiment of statements, this impact is negligible with the small portion of missing data (Ishaq *et al.*, 2024).

This method maintains data consistency by ensuring that imputed values are contextually relevant to the statuses’ categories. While some statements (e.g. row 13293 and 13294) may appear different statuses, this is still acceptable as it does not significantly impact the dataset’s overall structure.

Text Preprocessing

```
# Set vocabulary size and sequence length
vocab_size = 5000
max_len = 100
```

```
# Download stopwords if not already available
nltk.download("stopwords")

# Load spaCy model for stopwords
nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])
stop_words = nlp.Defaults.stop_words

important_words = {
    'like', 'feel', 'want', 'know', 'get', 'life', 'im', 'even', 'time', 'would',
    'really', 'people', 'cannot', 'one', 'going', 'think', 'go', 'much', 'never', 'day',
    'help', 'dont', 'things', 'could', 'years', 'anymore', 'anxiety', 'back', 'work', 'anything',
    'still', 'make', 'something', 'depression', 'got', 'friends', 'always', 'good', 'take', 'anyone',
    'feeling', 'way', 'everything', 'ive', 'better', 'every', 'need', 'see', 'also', 'happy','fulfilled','break','nervous','death','hate'
}
```

```
def preprocess_text(text):
    """Preprocesses text by removing special characters, numbers & stopwords while keeping important words."""
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'[^a-z\s]', '', text)  # Remove special characters & numbers
    words = text.split()

    # Keep words that are either NOT stopwords OR in the important words list
    filtered_words = [word for word in words if (word not in stop_words) or (word in important_words)]

    return " ".join(filtered_words)  # Join back into a sentence

# Apply preprocessing
df_clean["cleaned_statement"] = df_clean["statement"].astype(str).apply(preprocess_text)
```

Figure 1.1.6 Text Pre-processing

Before proceeding the further analysis, text preprocessing is applied to the *'statement'* column to enhance the relevance of the text data (GeeksforGeeks, 2024). We implemented NLP strategies to eliminate redundant words like special characters (e.g. 'ðÿ') and stopwords. Figure 1.1.6 illustrates the series of text preprocessing carried out to ensure the precision of the natural language. The preprocessing techniques include: lowercasing, special characters removal, tokenization, stopword removal, and tokenization.

The NLTK's predefined stopword list is efficient in removing stopwords, but it also also removes words like 'im' and 'feel', which contribute essential information to sentiment analysis. To resolve this, a custom '*important_words'* list is created with conditional logic to retain all meaningful words if they are either not in the stopword list or the important word list (GeeksforGeeks, 2024). After text pre-processing, the cleaned text is then stored in a separate column called '*cleaned_statement'*.

```
# Initialize and fit tokenizer
tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(df_clean["cleaned_statement"])

# Convert text to sequences and pad them
df_clean["padded_statement"] = list(pad_sequences(
    tokenizer.texts_to_sequences(df_clean["cleaned_statement"]),
    maxlen=max_len,
    padding='post',
    truncating='post'
))

# Display processed data
df_clean[["statement", "cleaned_statement", "padded_statement"]].head()
```

	statement	cleaned_statement	padded_statement
0	oh my gosh	oh gosh	[428, 4471, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
1	trouble sleeping, confused mind, restless hear...	trouble sleeping confused mind restless heart ...	[731, 464, 754, 138, 1005, 165, 1, 0, 0, 0, 0,...
2	All wrong, back off dear, forward doubt. Stay ...	wrong back dear forward doubt stay restless re...	[141, 29, 1971, 414, 918, 204, 1005, 1005, 157...
3	I've shifted my focus to something else but I'...	ive shifted focus something im still worried	[45, 1, 513, 34, 8, 32, 262, 0, 0, 0, 0, 0, 0,...
4	I'm restless and restless, it's been a month n...	im restless restless month boy mean	[8, 1005, 1005, 158, 1129, 222, 0, 0, 0, 0, 0,...

Figure 1.1.7 Tokenization Code Snippet

Then, tokenization is performed using *Tokenizer* from the Keras library, which splits each word and assigns it with a token value, allowing the text data to be compatible with machine learning models for text data processing and analysis. This is because the models can only process numerical values, thus they need to be converted before sent for model training. Additionally, the tokenizer is also equipped with an *'<OOV>'* token to replace any out-of-vocabulary words that are unseen during training, preventing error occurrence.

Padding and truncating techniques are also added to standardize input lengths, by adding zeros at the end *(padding='post')*, or truncating longer sequences *(truncation='post')*. These transformations are stored in *'padded_statement'* for easier comparison with the original column. Figure 1.1.7 displays these pre-processing steps, then outputs the cleaned statements and their corresponding padded (tokenized) statements for the first few rows.

Question 1 b

Identify Categories

```
# Identify and count total columns of each category
sentiment_count = df_clean['status'].value_counts().sort_values(ascending=True)
sentiment_count
```

status	count
Personality disorder	1199
Stress	2669
Bipolar	2873
Anxiety	3881
Suicidal	10648
Depression	15399
Normal	16374

```
# Plot the Distribution of Sentiments
sentiment_count.plot(kind='bar', title='Distribution of Sentiments')
```

```
<Axes: title={'center': 'Distribution of Sentiments'}, xlabel='status'>
```

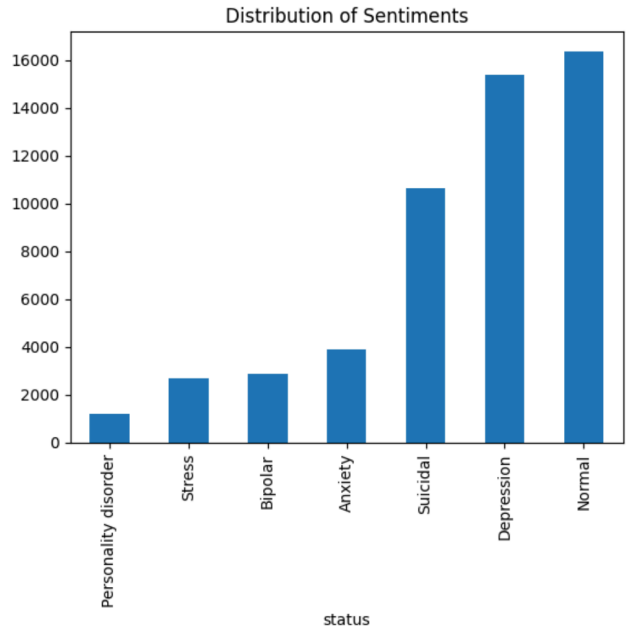

Figure 1.2.1 Identifying Total of each Category

Figure 1.2.1 identifies the categories in the *'status'* column and counts the total instances in each category using *value_counts()*. The dataset contains a total of 7 categories: Personality disorder, Stress, Bipolar, Anxiety, Suicidal, Depression, and Normal. They are sorted in ascending order based on frequency for better readability. Additionally, a bar chart is implemented to see the visual distribution of the categories.

Convert to Binary Classification

```
# Map 'status' column to binary classification
status_map = {
    'Normal': 'Normal',
    'Depression': 'Depression',
    'Suicidal': 'Depression',
    'Anxiety': 'Depression',
    'Stress': 'Depression',
    'Bipolar': 'Depression',
    'Personality disorder': 'Depression'
}
df_clean['status'] = df_clean['status'].map(status_map)

# Check updated counts
df_clean['status'].value_counts().sort_values(ascending=True)
```

status	count
Normal	16374
Depression	36669

Figure 1.2.2 Conversion to Binary Classification

Next, the multiple statuses are converted into a binary classification. As presented in Figure 1.2.2, the 'Depression', 'Suicidal', 'Anxiety', 'Stress', 'Bipolar', 'Personality disorder' statuses are assigned under 'Depression', while the 'Normal' status remains as 'Normal'. This classification strategy aligns with the definition of clinical depression, which refers to depressed mood and loss of interest and pleasure in daily activities (World Health Organization, 2023).

While they vary in severity and symptoms, they are still interconnected to depression. Suicidal thoughts, anxiety and stress are often categorized as direct symptoms or co-occuring conditions with depression. Personality disorders have a high comorbidity rate of 50% with depression, indicating a clinical association with depression (Wongpakaran *et al.*, 2015). On the other hand, Bipolar disorder includes depression episodes, making it a variant of depression (World Health Organization, 2023).

Although these conditions differ in diagnostic aspects, these mental disorders share similar emotional distress, thus we are in agreement to classify them under the same category to maintain clinical relevance and reduce misclassification possibility, ensuring that relevant conditions are grouped together for model performance improvement throughout the learning process.

Question 2 a

Encoding Categorical Data

```
# Use One-hot Encoding to convert categorical variable
df_clean['status'] = df_clean['status'].map({'Normal': 0, 'Depression': 1})
df_clean.head(10)
```

	statement	status	cleaned_statement	padded_statement
0	oh my gosh	1	oh gosh	[428, 4471, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
1	trouble sleeping, confused mind, restless hear...	1	trouble sleeping confused mind restless heart ...	[731, 464, 754, 138, 1005, 165, 1, 0, 0, 0, 0,...
2	All wrong, back off dear, forward doubt. Stay ...	1	wrong back dear forward doubt stay restless re...	[141, 29, 1971, 414, 918, 204, 1005, 1005, 157...
3	I've shifted my focus to something else but I'...	1	ive shifted focus something im still worried	[45, 1, 513, 34, 8, 32, 262, 0, 0, 0, 0, 0, 0,...
4	I'm restless and restless, it's been a month n...	1	im restless restless month boy mean	[8, 1005, 1005, 158, 1129, 222, 0, 0, 0, 0, 0,...
5	every break, you must be nervous, like somethi...	1	every break nervous like something wrong heck	[48, 274, 639, 2, 34, 141, 3056, 0, 0, 0, 0, 0...
6	I feel scared, anxious, what can I do? And may...	0	feel scared anxious family protected	[3, 128, 214, 54, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0...
7	Have you ever felt nervous but didn't know why?	0	felt nervous didnt know	[83, 639, 173, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
8	I haven't slept well for 2 days, it's like I'm...	0	havent slept days like im restless huh	[369, 1120, 87, 2, 8, 1005, 2862, 0, 0, 0, 0, ...
9	I'm really worried, I want to cry.	1	im really worried want cry	[8, 12, 262, 4, 232, 0, 0, 0, 0, 0, 0, 0, 0, 0...

Figure 2.1.1. Label Encoding Code Implementation

Since machine learning models cannot process categorical values directly, the *'status'* column needs to be encoded into numerical format, even after classifying them into binary classifications. Shown in Figure 2.1.1, Label Encoding is applied, assigning each category with a numerical value (0 or 1). Mental health statuses such as 'anxiety', 'depression', and 'suicidal' are classified as Depression (1), whereas the "normal" state are assigned a value of 0.

Label Encoding chosen over One-Hot Encoding because One-Hot Encoding will create two separate columns (e.g. [1,0] for Normal and [0,1] for Depression) for its binary solution. This approach creates unnecessary dimensions and redundancy that increase complexity to the model instead. One-Hot Encoding is more suitable for multiclass classification, where ordinal correlation exists among the categories (Manai et al., 2023).

Data Shuffling

```
df_shuffled = shuffle(df_clean, random_state=42)

# Display the first few rows of the shuffled DataFrame
df_shuffled.head()
```

	statement	status	cleaned_statement	padded_statement
22265	Just as the the title says. I feel like one is...	1	title says feel like one hyper alot attention ...	[951, 313, 3, 2, 15, 2618, 1033, 535, 3713, 1,...
41404	a blackened sky encroached tugging behind it m...	1	blackened sky encroached tugging depression	[1, 2760, 1, 1, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
20069	It gives you insomnia, which in turn makes you...	1	gives insomnia turn makes depression worse day...	[740, 1498, 377, 111, 35, 98, 21, 1, 97, 9, 74...
30040	Hello all, I'm a new submitter to this channel...	0	hello im new submitter channel went nights ago...	[614, 8, 101, 1, 3609, 104, 1025, 95, 229, 623...
780	Thank God the CB is over for Eid	0	thank god cb eid	[268, 279, 3852, 2125, 0, 0, 0, 0, 0, 0, 0,...

Figure 2.1.2 Shuffling the Data

Figure 2.1.2 demonstrates random data shuffling and its output, an essential step used to prevent biases during model training before splitting the data into training and testing datasets. This process involves randomly rearranging the dataset to eliminate any inherent order, such as sequential patterns, that could influence the model's learning. The *shuffle()* function, applied with *random_state*, is implemented to maintain reproducibility while achieving randomness in the dataset.

Question 2 b

Data Splitting

```
# Extract features (padded sequences) and labels
X = df_shuffled["padded_statement"].tolist()  # Convert padded sequences to a list
y = df_shuffled["status"]  # Target labels

# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Print dataset sizes
print("Training set size:", len(X_train))
print("Testing set size:", len(X_test))
```

```
Training set size: 42434
Testing set size: 10609
```

Figure 2.2.1. Code Implementation and Output of Splitting Dataset

Finally, X (independent variable) and Y (dependent variable) can be declared. The question identifies X as the "statement" column and Y as the "status" column. Using the Pareto theory, which states that the training dataset must be more than the testing dataset, the dataset is split into 80% for training and 20% for testing for a balanced model evaluation. Therefore, the code and output is shown in Figure 2.2.1.

Decision Tree Model

```
[ ]
# Train Decision Tree Classifier
clf = DecisionTreeClassifier(
    max_depth=5,
    min_samples_split=10,
    min_samples_leaf=5,
    class_weight={0: 1, 1: 4},  # Increase weight for class 1
    random_state=42
)

clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)
```

Figure 2.2.2. Decision Tree Model Code Snippet

Figure 2.2.3. Decision Tree Graph

Now, the Decision tree model can be built and trained. The code in Figure 2.2.2 demonstrates the process of constructing and evaluating the decision tree model. The decision tree model's structure is shown in Figure 2.2.3. After placing a statement into a category as either “depression” or “normal”, the root node then splits into different sides depending on that decision. By taking into account new situations, each following node (leaf node) improves the classification. In the figure, blue represents "Normal," while red represents “Depression”.

However, due to the imbalance between “Normal” and “Depression” cases, the model favors the majority class, “Normal”, while mostly ignoring the minority class. To solve the issue,

a class weight modification of {0:1, 1:4} was used, which increases the emphasis on the minority class and boosts the model's capacity to spot cases of depression. Additionally, to solve any overfitting issues, we have limited the decision tree to have a *max_depth* of 5, *min_samples_split* of 10 and *min_samples_leaf* of 5, in order to prevent the tree from becoming too complex and the possible case of redundancies.

Neural Network

```
# Define the model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=64, input_length=max_len),
    GaussianNoise(0.15),  # Balanced noise for smoother training

    LSTM(10, return_sequences=False, kernel_regularizer=l2(0.002)),  # Moderate LSTM size
    BatchNormalization(),
    Dropout(0.55),  # Slightly higher dropout to slow learning

    Dense(1, activation='sigmoid', kernel_regularizer=l2(0.002))
])

# Slightly lower learning rate
optimizer = Adam(learning_rate=0.0002)

# Label smoothing to prevent overconfidence
loss_fn = tf.keras.losses.BinaryCrossentropy(label_smoothing=0.05)

# Compile the model
model.compile(optimizer=optimizer, loss=loss_fn, metrics=['accuracy'])
```

Figure 2.2.4. Code implementation of Neural Network model

Since this is a sentiment analysis model, we chose an RNN model which is well-suited for processing sequential data, such as textual data. Since sentiment analysis relies on understanding the context and meaning of words within a sentence, an RNN can capture dependencies between words and retain information from previous inputs (Pal et al., 2018). Figure 2.2.4 and Table 2.1 displays and justifies the model for our neural network and its compiler.

Table 1.Overview of Neural Network Model

Layer	Type	Explanation
1	Embedding	Transforms words into dense vector representations, helping the model capture semantic relationships between words. The output dimension of 64 provides a balance between computational efficiency and representation quality.
2	GaussianNoise	Adds controlled noise (standard deviation of 0.15) to improve generalization and reduce overfitting, ensuring the model does not become too sensitive to training data.
3	LSTM	Retains long-term dependencies, which is crucial for understanding context in sentiment analysis. The L2 regularization (0.002) helps prevent overfitting, and using only 10 units keeps the model lightweight while maintaining effectiveness.
4	BatchNormalization	Stabilizes training, normalizes activations, and accelerates convergence, improving model performance.
5	Dropout	Reduces overfitting by randomly deactivating 55% of neurons during training, forcing the model to learn more robust features.
6	Dense	Uses a sigmoid activation function to produce a probability output for binary classification (Normal vs. Depression). The L2

		regularization (0.002) prevents excessive reliance on specific features.

While training the model, many cases of overfitting were encountered. To address this issue, several strategies were implemented, such as lowering the learning rate of the Adam optimizer to 0.0002 (instead of the default 0.001) to ensure stable training and prevent overshooting. Binary Cross-Entropy (BCE) was used as the loss function since the task requires binary categorization (Normal vs. Depression). To further enhance generalization, label smoothing (0.05) was applied, preventing the model from becoming overconfident by slightly adjusting the target labels (e.g. 0.95 instead of 1, and 0.05 instead of 0). (Srivastava et al., 1970)

```
# Callbacks for controlled learning
early_stopping = EarlyStopping(monitor='val_loss', patience=4, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)  # Increase patience
```

Figure 2.2.5. Code implementation of Early Stopping and ReducLROnPlateau.

```
nltk.download("averaged_perceptron_tagger_eng")

# Define text augmentation methods
syn_aug = naw.SynonymAug(aug_src="wordnet")  # Synonym Replacement
swap_aug = naw.RandomWordAug(action="swap")  # Random Word Swap
delete_aug = naw.RandomWordAug(action="delete")  # Random Deletion

def augment_text(text):
    """Applies a random augmentation technique to the given text."""
    aug = random.choice([syn_aug, swap_aug, delete_aug])
    return aug.augment(text)

# Apply augmentation to the cleaned text
augmented_texts = [augment_text(text) for text in df_shuffled["cleaned_statement"].astype(str).tolist()]

# Convert both original and augmented text to sequences and pad them
x_train_padded = pad_sequences(tokenizer.texts_to_sequences(df_shuffled["cleaned_statement"]), maxlen=max_len, padding="post", truncating="post")
x_aug_padded = pad_sequences(tokenizer.texts_to_sequences(augmented_texts), maxlen=max_len, padding="post", truncating="post")

# Combine original and augmented data
X_train_combined = np.vstack([x_train_padded, x_aug_padded])
y_train_combined = np.concatenate([y_train, y_train])  # Duplicate labels for augmented data
```

Figure 2.2.6. Code implementation of Data Augmentation.

Additional strategies to mitigate overfitting before training included early stopping, which halts training when validation loss (*val_loss*) no longer improves, and learning rate reduction on plateau, which decreases the learning rate when validation loss stagnates, allowing for finer adjustments. Lastly, data augmentation techniques, such as synonym replacement, random word swapping, and random word deletion, were applied to increase data diversity and reduce overfitting. (Srivastava et al., 1970). This is illustrated in Figures 2.2.5 and 2.2.6.

```
# Train the model
history = model.fit(np.array(X_train), y_train, epochs=50, batch_size=64, callbacks=[early_stopping], validation_data=(np.array(X_test), y_test))
```

Figure 2.2.7. Code implementation of Training the model.

The choice of batch size, epochs, and other hyperparameters for training neural networks for sentiment analysis is an important consideration. Referencing Figure 2.2.7, batch size of 64 was chosen to balance between computational efficiency and model performance (Masters and

Luschi, 2018). Different values of epochs, such as 128 and 256, were tested, with the results showing that 64 epochs provided the most stable growth in the training process of the model.

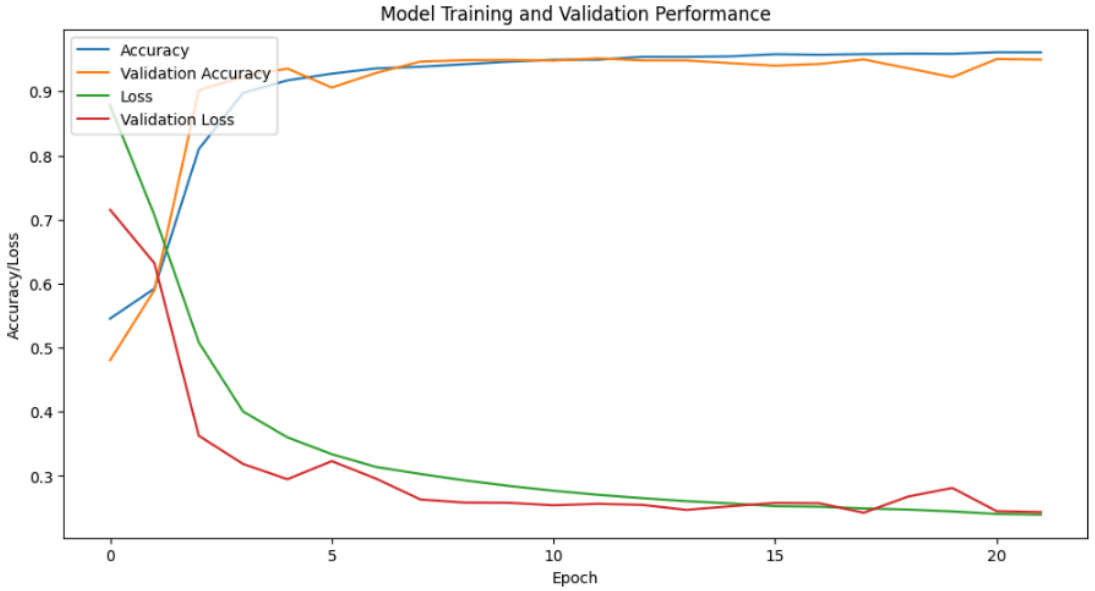

Figure 2.2.8. Output of the training.

The final graph for model training and validation performance shows a steady increase in accuracy, followed by validation accuracy (*val_acc*), along with a steady decline in loss and validation loss (*val_loss*).

Question 3 a

This question covers the evaluation of performance metrics of both models. Figure 3.1.1 below displays the code and output while Table 2 compares the decision tree and neural network to determine which one is more reliable and accurate.

```
# Calculate metrics
def evaluate_model(name, y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)

    print(f"--- {name} Performance ---")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(classification_report(y_true, y_pred))
    print("\n")

# Evaluate both models
evaluate_model("Decision Tree", y_test, y_pred_tree)
evaluate_model("Neural Network", y_test, y_pred_nn_binary)
```

```
--- Decision Tree Performance ---
Accuracy: 0.8250
Precision: 0.8092
Recall: 0.9753
              precision    recall  f1-score   support

           0       0.90      0.49      0.64      3315
           1       0.81      0.98      0.88      7294

    accuracy                           0.82     10609
   macro avg       0.86      0.73      0.76     10609
weighted avg       0.84      0.82      0.81     10609


--- Neural Network Performance ---
Accuracy: 0.9499
Precision: 0.9529
Recall: 0.9753
              precision    recall  f1-score   support

           0       0.94      0.89      0.92      3315
           1       0.95      0.98      0.96      7294

    accuracy                           0.95     10609
   macro avg       0.95      0.93      0.94     10609
weighted avg       0.95      0.95      0.95     10609
```

Figure 3.1.1. Code implementation & Output of Performance Metrics

The Neural Network is the better model due to its higher accuracy and precision across both classes. While the Decision Tree has a slightly better recall for class 1, it struggles with precision and overall balance, making the neural network the better option for predictions. This could be due to several factors, such as the imbalanced number of normal and depression cases or the fact that decision trees typically perform better with TF-IDF for text vectorization rather than raw tokenized input. (Prati et al., 2008). Table 3 below compares and evaluates the performance metrics between both models.

Table 3.1. Comparison of Decision Tree and Neural Network's Performance Metrics

Metric	Decision Tree	Neural Network	Better Model	Evaluation
Accuracy	82.51%	94.99%	Neural Network	The neural network outperforms the decision tree in overall accuracy, making it more reliable.
Precision	80.93%	95.29%	Neural Network	The neural network has better precision, meaning fewer false positives.
Recall	97.53%	97.53%	Both the same	Both have the same score for predicting false negatives.

Confusion Matrix

To further justify our results, we have plotted the confusion matrix for both models. Shown in the figures below.

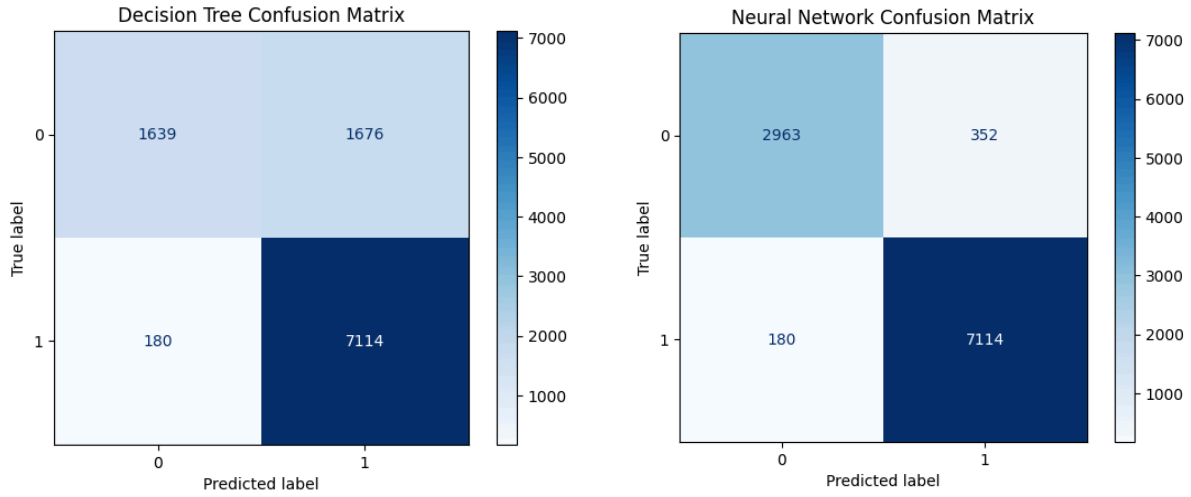

Figure 3.1.2. Confusion Matrix for both Models

The decision tree's confusion matrix shows 1,676 false positives, which explains its precision of 80.93%, while the neural network has only 352 false positives and has a higher precision score (95.29%). Similarly, for recall, the decision tree predicted 7,114 true positives which is the same as the neural network's 7,114 true positives. This further justifies why both models have the same recall. Overall, the neural network model performed better than the decision tree model in terms of accuracy and precision.

Question 3 b

ROC & AUC Graph

```
[ ]  1 # Compute ROC curve & AUC
     2 fpr_tree, tpr_tree, _ = roc_curve(y_test, clf.predict_proba(X_test)[:, 1])
     3 auc_tree = auc(fpr_tree, tpr_tree)
     4
     5 fpr_nn, tpr_nn, _ = roc_curve(y_test, y_pred_nn)  # No need to use proba, NN outputs sigmoid
     6 auc_nn = auc(fpr_nn, tpr_nn)
```

```
     1 # Plot ROC curves
     2 plt.figure(figsize=(8, 6))
     3 plt.plot(fpr_tree, tpr_tree, label=f"Decision Tree (AUC = {auc_tree:.4f})", linestyle="--")
     4 plt.plot(fpr_nn, tpr_nn, label=f"Neural Network (AUC = {auc_nn:.4f})", linestyle="-")
     5
     6 # Reference line for random classifier
     7 plt.plot([0, 1], [0, 1], color="gray", linestyle="dotted")
     8
     9 # Labels and legend
    10 plt.xlabel("False Positive Rate (FPR)")
    11 plt.ylabel("True Positive Rate (TPR)")
    12 plt.title("ROC Curve Comparison: Decision Tree vs Neural Network")
    13 plt.legend()
    14 plt.show()
```

Figure 3.2.1. Code Implementation to calculate and plot ROC and AUC graph

To plot the ROC and AUC graph, the code in Figure 3.2.1 computes the ROC curve and the AUC for both the Decision Tree (*clf.predict_proba*) and the Neural Network (*y_pred_nn*). It calculates the False Positive Rate (FPR) and True Positive Rate (TPR) to measure how well the models distinguish between classes.

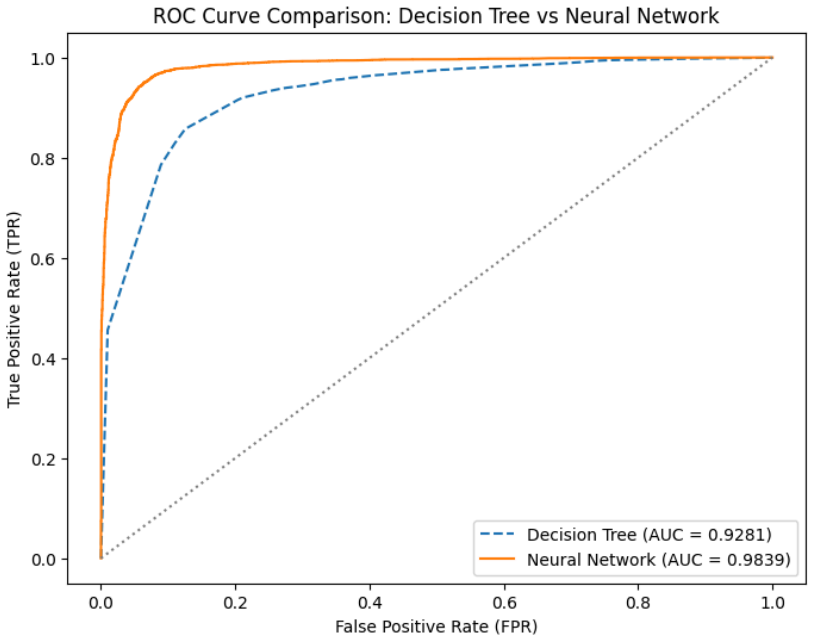

Figure 3.2.2. ROC and AUC Graph

Figure 3.2.2 shows the final graph, where the orange line represents the Neural Network and the blue dashed line represents the Decision Tree. The ROC curve compares their classification performance, with the y-axis (TPR) showing correctly predicted positive cases and the x-axis (FPR) indicating false positives. A good model curves toward the top-left corner, indicating higher accuracy. The Neural Network (AUC = 0.9839) outperforms the Decision Tree (AUC = 0.9281), meaning it better distinguishes between classes. The dotted diagonal line (AUC = 0.5) represents random guessing, and both models perform significantly better. However, the Neural Network consistently achieves a higher TPR at the same FPR, making it the better model for predictions with fewer errors.

Question 4

Test Statements Using Decision Tree and Neural Network Models

This question examines how a Decision Tree model and a Neural Network model classify five different statements as either 'Normal' or 'Depressed'. Both models use different ways to understand text and predict mental health conditions. As a result of this question, next we will mainly highlight the steps for classification and its results, then explain the possible reasons behind the discrepancies.

Both Decision Tree and Neural Network models use very similar methods to classify statements based on their likelihood of indicating depression. The classification process consists of several key steps, including text preprocessing, sequence conversion, padding, and final classification.

Step 1 : Text Preprocessing

The first step involves preparing the text for numerical processing. The input text is converted to lowercase, and all punctuation and special characters are removed to ensure consistency. The text is then tokenized, breaking it down into individual words, making it easier for the model to analyze patterns.

Step 2 : Convert Text to Sequences

Once tokenized, the words are mapped to unique integer values using a trained tokenizer. Each word's numerical representation is based on its frequency in the training dataset, allowing both models to process the text in a structured manner.

Step 3 : Padding Sequences

To ensure all inputs have a consistent length, shorter sequences are padded with zeros, and longer sequences are truncated to fit a predefined size. Additionally, for the Neural Network model, the padded sequences are converted into a NumPy array to ensure compatibility with the deep learning framework.

Step 4 : Classification Process

a) **Decision Tree Model:** The preprocessed and padded text is passed into the Decision Tree classifier, which applies predefined decision rules to analyze word patterns and determine the probability of the statement indicating depression.

b) **Neural Network Model:** The formatted text is fed into a trained Neural Network, which processes the input through multiple layers to extract features, detect contextual meaning, and refine predictions. The final layer produces a probability score between 0 and 1, indicating the likelihood of depression.

Step 5 : Final Classification

a) **Decision Tree Model:** If the probability score is 0.5 or higher, the statement is classified as 'Depressed' (1); otherwise, it is classified as 'Normal' (0).

b) **Neural Network Model:** If the probability score is 0.7 or higher, the statement is classified as 'Depressed' (1); otherwise, it is classified as 'Normal' (0).

Test Statements and Classification Results

Table 3. Test Statements and Classification Results

Statement	Decision Tree Prediction	Decision Tree Probability	Neural Network Prediction	Neural Network Probability
"I can't stop worrying about everything"	Depressed (1)	0.68	Depressed (1)	0.83
"I've been working hard, and seeing the results makes me feel incredibly happy and fulfilled"	Depressed (1)	0.88	Normal (0)	0.65
Even the smallest things feel like too much right now"	Depressed (1)	0.80	Depressed (1)	0.77
"I can’t stop smiling"	Normal (0)	0.64	Normal (0)	0.15
"Today has been amazing!"	Normal (0)	0.64	Normal (0)	0.03

Classification Code and Output - Decision Tree

```python
# Example input texts
new_texts = [
    "I can't stop worrying about everything",
    "I've been working hard, and seeing the results makes me feel incredibly happy and fulfilled",
    "Even the smallest things feel like too much right now",
    "I can’t stop smiling",
    "Today has been amazing!"
]

# Preprocess each text
cleaned_texts = [preprocess_text(text) for text in new_texts]

# Convert text to sequences
new_texts_sequences = tokenizer.texts_to_sequences(cleaned_texts)

# Pad sequences to match model's input shape
new_texts_padded = pad_sequences(new_texts_sequences, maxlen=max_len, padding='post', truncating='post')

# Ensure input format is correct (convert to list if needed)
new_texts_padded = new_texts_padded.tolist()

# Make predictions using the trained model
predictions = clf.predict(new_texts_padded)  # Get predictions for all statements
probabilities = clf.predict_proba(new_texts_padded)  # Get probabilities

# Print results
for i, text in enumerate(new_texts):
    print(f"\nStatement: {text}")
    print("Prediction Probabilities:", probabilities[i])

    if predictions[i] >= 0.5:
        print("Prediction: The person may have Depression (1).")
    else:
        print("Prediction: The person is Normal (0).")
```

Figure 4.1. Code Implementation of Classification - Decision Tree

```
Statement: I can't stop worrying about everything
Prediction Probabilities: [0.32177845 0.67822155]
Prediction: The person may have Depression (1).

Statement: I've been working hard, and seeing the results makes me feel incredibly happy and fulfilled
Prediction Probabilities: [0.11966854 0.88033146]
Prediction: The person may have Depression (1).

Statement: Even the smallest things feel like too much right now
Prediction Probabilities: [0.20440118 0.79559882]
Prediction: The person may have Depression (1).

Statement: I can’t stop smiling
Prediction Probabilities: [0.64097915 0.35902085]
Prediction: The person is Normal (0).

Statement: Today has been amazing!
Prediction Probabilities: [0.64097915 0.35902085]
Prediction: The person is Normal (0).
```

Figure 4.2. Output of Classification - Decision Tree

Classification Code and Output - Neural Network

```python
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Example input texts
new_texts = [
    "I can't stop worrying about everything",
    "I've been working hard, and seeing the results makes me feel incredibly happy and fulfilled",
    "Even the smallest things feel like too much right now",
    "I can’t stop smiling",
    "Today has been amazing!"
]

# Preprocess each text
cleaned_texts = [preprocess_text(text) for text in new_texts]

# Convert texts to sequences
new_texts_sequences = tokenizer.texts_to_sequences(cleaned_texts)

# Pad sequences to match model's input shape
new_texts_padded = pad_sequences(new_texts_sequences, maxlen=max_len, padding='post', truncating='post')

# Convert to NumPy array before prediction
new_texts_padded = np.array(new_texts_padded)

# Make predictions using the trained model
predictions = model.predict(new_texts_padded)  # Get predictions for all statements

# Print results
for i, text in enumerate(new_texts):
    prob = predictions[i][0]  # Extract probability
    print(f"\nStatement: {text}")
    print("Prediction Probability:", prob)

    # Output classification
    if prob >= 0.7:  # Threshold set to 0.7 for classifying depression
        print("Prediction: The person may have Depression (1).")
    else:
        print("Prediction: The person is Normal (0).")
```

Figure 4.3. Code Implementation of Classification - Neural Network

```
Statement: I can't stop worrying about everything
Prediction Probability: 0.83109164
Prediction: The person may have Depression (1).

Statement: I've been working hard, and seeing the results makes me feel incredibly happy and fulfilled
Prediction Probability: 0.6544109
Prediction: The person is Normal (0).

Statement: Even the smallest things feel like too much right now
Prediction Probability: 0.772475
Prediction: The person may have Depression (1).

Statement: I can’t stop smiling
Prediction Probability: 0.15198001
Prediction: The person is Normal (0).

Statement: Today has been amazing!
Prediction Probability: 0.037440002
Prediction: The person is Normal (0).
```

Figure 4.4. Output of Classification - Neural Network

Explanation of Classifications

1. *"I can't stop worrying about everything"*

The statement was classified as **Depression** by both models, with probabilities of 0.68 (Decision Tree) and 0.83 (Neural Network). Both recognized the strong negative sentiment and overwhelming nature of the phrase.

2. *"I've been working hard, and seeing the results makes me feel incredibly happy and fulfilled"*

The statement was misclassified as **Depression** by the Decision Tree model (0.88), likely due to associations between "working hard" and stress in the training data. In contrast, the Neural Network correctly classified it as **Normal** (0.65), capturing the overall positive sentiment. This highlights a key difference between the models where the Decision Tree relies on rigid word associations, the Neural Network better understands context and emotional nuance.

3. "Even the smallest things feel like too much right now"

The statement was also classified as **Depression**, with probabilities of 0.80 (Decision Tree) and 0.77 (Neural Network), as both models identified signs of emotional exhaustion and distress.

4. *"I can't stop smiling"*

For "I can't stop smiling," both models correctly classified it as **Normal**, with probabilities of 0.64 (Decision Tree) and 0.15 (Neural Network), recognizing the clear expression of happiness.

5. *"Today has been amazing!"*

Similarly, "Today has been amazing!" was classified as **Normal**, with probabilities of 0.64 (Decision Tree) and 0.03 (Neural Network), correctly detecting the strong positive sentiment.

Conclusion

Mental health issues have become increasingly prevalent in today's fast-paced and high-pressure society. There are all sorts of things that can make anxiety and depression happen. Stress at work, people's expectations, having a lot to spend money on, and of course, it's always handling life's difficulties pretty hard. If left undetected, these conditions can lead to severe consequences, including decreased productivity, emotional distress, and potential self-harm. Therefore, identifying mental health issues early is super important because that allows us to jump in with quick support and help right away.

This assignment compared two machine learning models which are Decision Tree and Neural Networks in classifying text statements as either Normal or Depressed. While both models managed to spot clear signs of happiness and depression, one of them distinguished sharply from the other when classifying subtle statements as well. The Neural Network model demonstrated superior contextual understanding, correctly identifying positive sentiment in complex sentences, whereas the Decision Tree model misclassified one positive statement as Depressed due to over-reliance on specific words without considering overall sentiment.

In summary, there are advantages and disadvantages to both strategies for categorising text-based mental health indicators. While the Decision Tree model relies on specific words and pre-learned rules, it sometimes misclassifies positive statements due to its keyword-based decision-making approach. The Neural Network model, on the other hand, is better at assessing sentiment in general and is therefore more reliable at distinguishing between positive and negative expressions.

References

GeeksforGeeks. (2024, Jan 03). Removing stop words with NLTK in Python. *GeeksforGeeks*. https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

Ishaq, M., Zahir, S., Iftikhar, L., Bulbul, M. F., Rho, S., & Lee, M. Y. (2024b). Machine learning based missing data imputation in categorical datasets. *IEEE Access*, *12*, 88332–88344. https://doi.org/10.1109/access.2024.3411817

Manai, E., Mejri, M., & Fattahi, J. (2023, July 10). *Impact of feature encoding on malware classification explainability*. arXiv.Org. https://arxiv.org/abs/2307.05614

Masters, D., & Luschi, C. (2018, April 20). *Revisiting small batch training for deep neural networks*. arXiv.Org. https://arxiv.org/abs/1804.07612

Memon, S. MZ., Wamala, R., & Kabano, I. H. (2023b). A comparison of imputation methods for categorical data. *Informatics in Medicine Unlocked*, *42*, 101382. https://doi.org/10.1016/j.imu.2023.101382

Pal, S., Ghosh, S., & Nag, A. (2018, January 1). *Sentiment analysis in the light of LSTM recurrent neural networks*. IGI Global. https://www.researchgate.net/publication/326242364_Sentiment_Analysis_in_the_Light_of_LSTM_Recurrent_Neural_Networks

Pargent, F., Pfisterer, F., Thomas, J., & Bischl, B. (2022). Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, *37*(5), 2671–2692. https://doi.org/10.1007/s00180-022-01207-6

Prati, R. C., Batista, G. E., & Monard, M.-C. (2008, July 10). *A study with class imbalance and random sampling for a decision tree learning system*. Unknown. http://researchgate.net/publication/226275222_A_Study_with_Class_Imbalance_and_Random_Sampling_for_a_Decision_Tree_Learning_System

Setu, S. (2024, July 30). Sentiment analysis for mental health monitoring. *Kaggle*. https://www.kaggle.com/code/sahityasetu/sentiment-analysis-for-mental-health-monitoring/notebook

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Wongpakaran, N., Boonyanaruthee, V., Pinyopornpanish, M., Intaprasert, S., & Wongpakaran, T. (2015). Comorbid personality disorders among patients with depression. *Neuropsychiatric Disease and Treatment*, 1091. https://doi.org/10.2147/ndt.s82884

World Health Organization: WHO. (2023, March 31). Depressive disorder (depression). *World Health Organization: WHO*. https://www.who.int/news-room/fact-sheets/detail/depression