NLP Project for Disaster Tweet Classification

NextHikes IT Solution

Submitted by: Kalavathi Alegapalli

Overview of Disaster Tweet Classification

Data Exploration and Preparation

Load and analyze the dataset to understand its structure and content.

Feature Engineering

Extract relevant features from the text data to enhance model performance.

Model Selection and Training

Choose and train suitable classification models for text classification.

Model Evaluation and Validation

Assess the model's performance using appropriate metrics and validate its robustness.

Model Deployment

Integrate the trained model into a web application for real-time tweet classification.

Task 1: Data Exploration and Preparation

 # Features of the dataset:

Id : A unique identifier corresponding to the tweet

Keyword : A highlighting word from the tweet

Location : The location from where the tweet is sent

Text :The textual content of the tweet

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

target	count
0	4342
1	3271

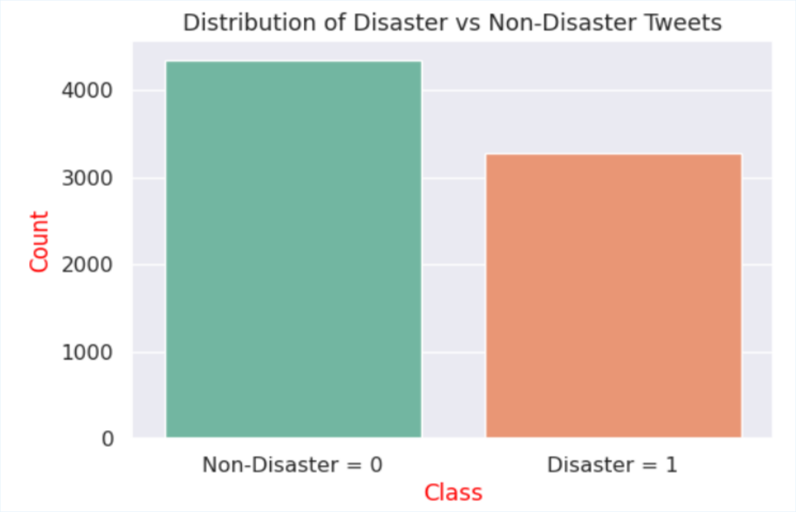
dtype: int64

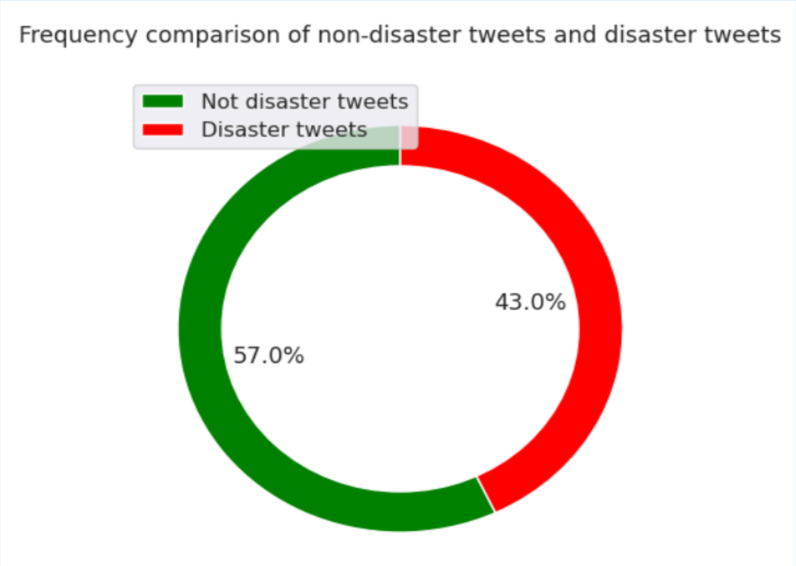
Target : A binary variable, which is 0
if the tweet does not indicate a real disaster and 1 if it does

```
# Dataset Structure
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        7613 non-null   int64
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
```

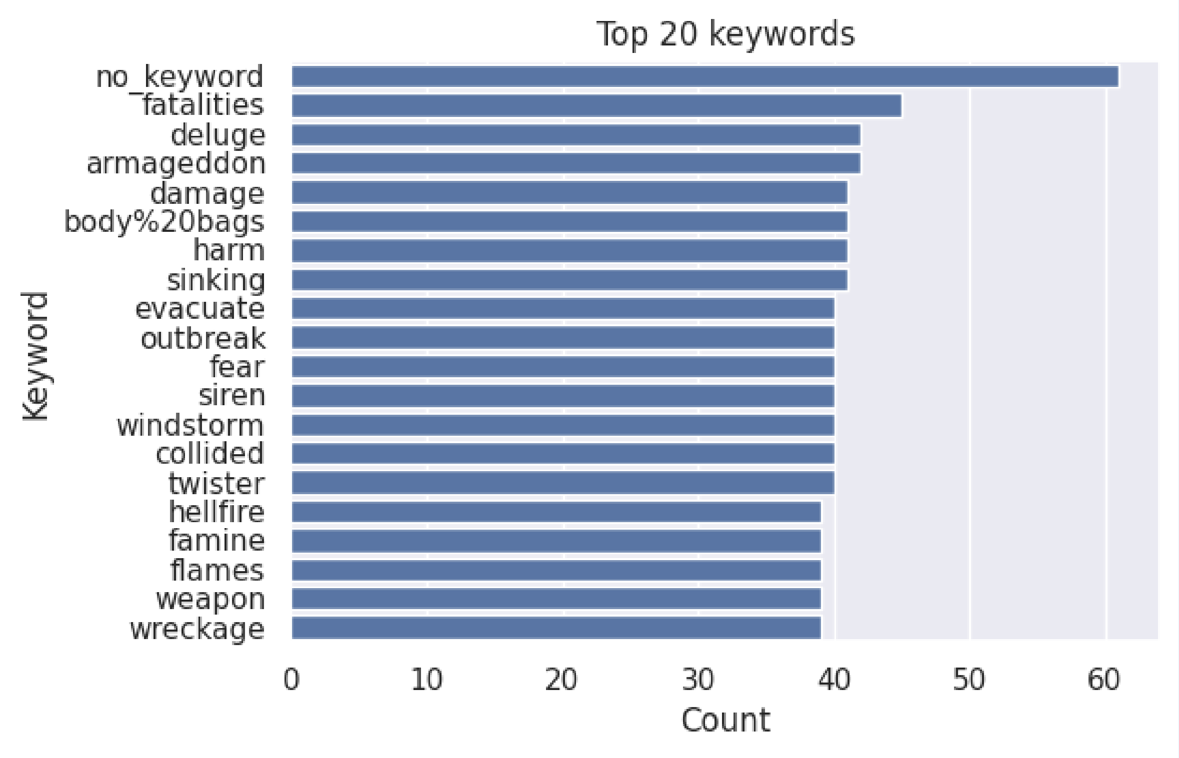
Visualize the distribution of classes (disaster non-disaster tweets):





- ❖ The dataset shows a moderate class imbalance, with 57% non-disaster and 43% disaster tweets.
- ❖ While fairly balanced, models may still lean toward predicting non-disaster tweets.
- ❖ Using metrics like F1-score and techniques like class weighting or resampling can help ensure reliable disaster detection.
- ❖ The distribution supports effective model training with minimal bias.

Distribution by class:



❖ **Both disaster (class 1) and non-disaster (class 0) tweets peak near the 140-character mark, indicating users often use the full tweet length.**

❖ **Non-disaster tweets are more frequent across most lengths.**

❖ **The similar density curves suggest tweet length may be a useful but not strongly discriminative feature.**

❖ **The chart shows the top 20 keywords in the dataset, with "no_keyword" being the most frequent, appearing nearly 60 times.**

❖ **Keywords like "fatalities", "deluge", and "damage" are also common, reflecting disaster-related themes.**

❖ **These terms can enhance model performance by serving as strong indicators of tweet context**

Clean the text data by removing special characters, URLs, and punctuation marks.

```
def clean_text(text):
    # Remove URLs
    text = re.sub(r'http[s]?://\S+', '', text)

    # Remove special characters and punctuation marks
    text = re.sub(r'[^\w\s]', '', text)

    # Optionally, you can remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()

    return text
```

```
#Cleaning the text
df['cleaned_text'] = df['text'].apply(clean_text)

#Convert into lowercase
df['cleaned_text'] = df['cleaned_text'].str.lower()
```

```
df['cleaned_text']
```

Tokenize the text into individual words or tokens

Generate + Code + Markdown

```
    # For tokenizing text
    nltk.download('punkt')
    # For the punkt tokenizer model used by word_tokenize
    nltk.download('punkt_tab')

    # For stopwords
    nltk.download('stopwords')
```

Py

	cleaned_text
0	our deeds are the reason of this earthquake ma...
1	forest fire near la ronge sask canada
2	all residents asked to shelter in place are be...
3	13000 people receive wildfires evacuation orde...
4	just got sent this photo from ruby alaska as s...
...	...
7608	two giant cranes holding a bridge collapse int...
7609	aria_ahrary thetawniest the out of control wil...
7610	m194 0104 utc5km s of volcano hawaii
7611	police investigating after an ebike collided w...
7612	the latest more homes razed by northern califo...

7613 rows × 1 columns

BERT or Other Transformer Embedding's:

```
torch.cuda.empty_cache() if torch.cuda.is_available() else None
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 2.62kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 2.53MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 5.55MB/s]
config.json: 100% 570/570 [00:00<00:00, 15.2kB/s]
model.safetensors: 100% 440M/440M [00:11<00:00, 50.8MB/s]
Processing batches: 100%|██████████| 238/238 [13:17<00:00, 3.35s/it]
```

Sentiment Analysis:

```python
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')

sia = SentimentIntensityAnalyzer()
df['sentiment'] = df['cleaned_text'].apply(lambda x: sia.polarity_scores(x)['compound'])
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

Summary of the image content:

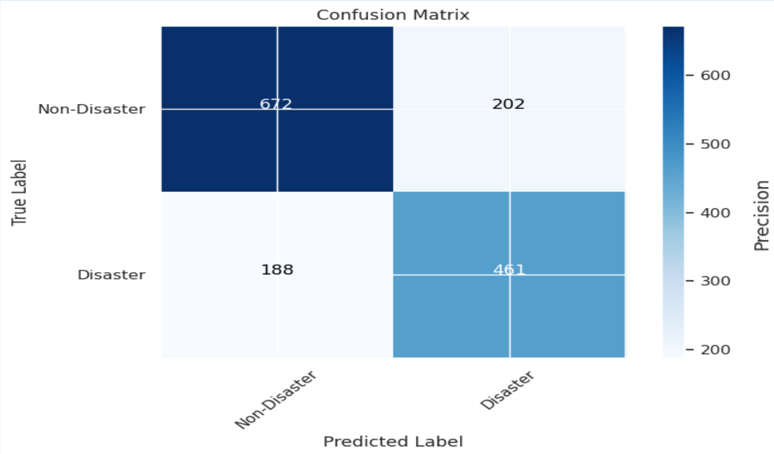
- **The code uses VADER from NLTK to perform sentiment analysis on a DataFrame column called cleaned_text.**

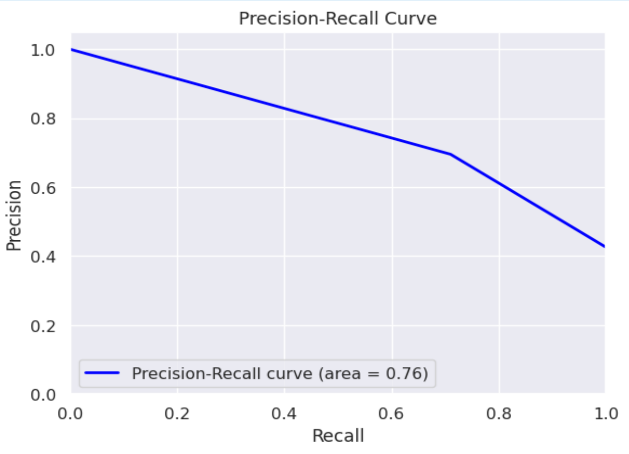
- **It calculates the compound sentiment score and stores it in a new column named sentiment.**

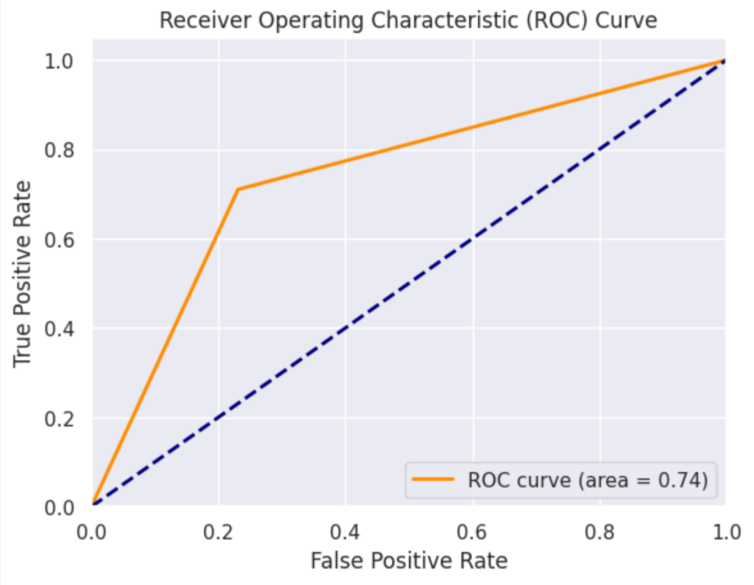
- **The vader_lexicon is downloaded to enable the SentimentIntensityAnalyzer.**

- **All configurations and model components shown a bove the code are fully loaded (100%).**

Logistic Regression Classification



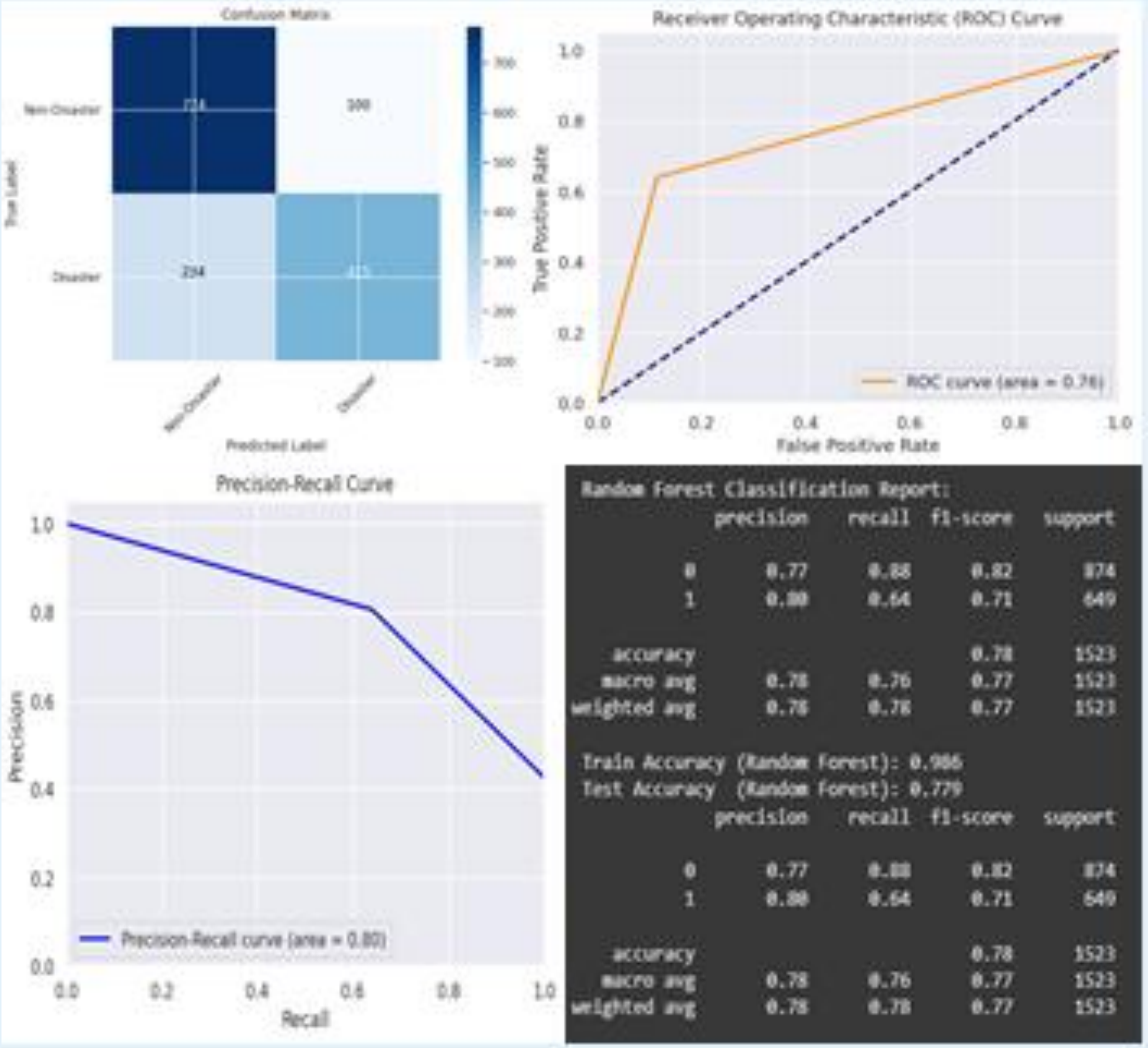





```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       874
           1       0.82      0.67      0.74       649

    accuracy                           0.80      1523
   macro avg       0.80      0.78      0.78      1523
weighted avg       0.80      0.80      0.79      1523

Train Accuracy: 0.852
Test Accuracy : 0.796
```

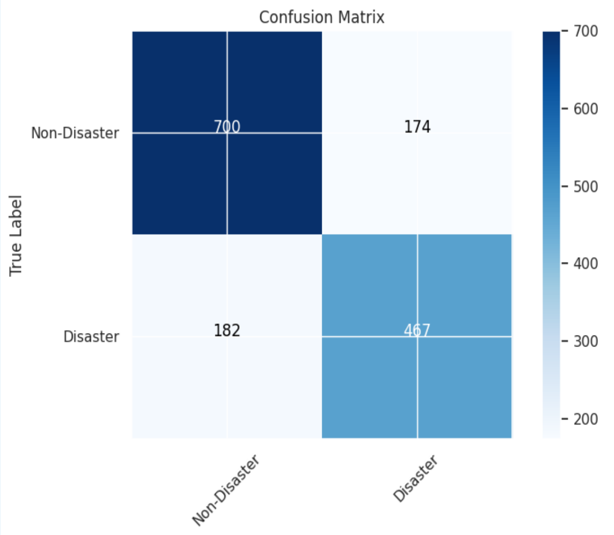
- **The image presents evaluation metrics for a logistic regression model, including a confusion matrix, precision-recall curve, ROC curve, and classification report.**
- **The classification report shows precision, recall, and f1-score for two classes, with an overall test accuracy of 79.6%.**
- **Class 1 has higher recall (0.87) and f1-score (0.84) compared to class 0.**
- **The ROC and precision-recall curves visualize the model's ability to distinguish between classes.**
- **Train accuracy is 85.2%, indicating slight overfitting compared to the test accuracy.**

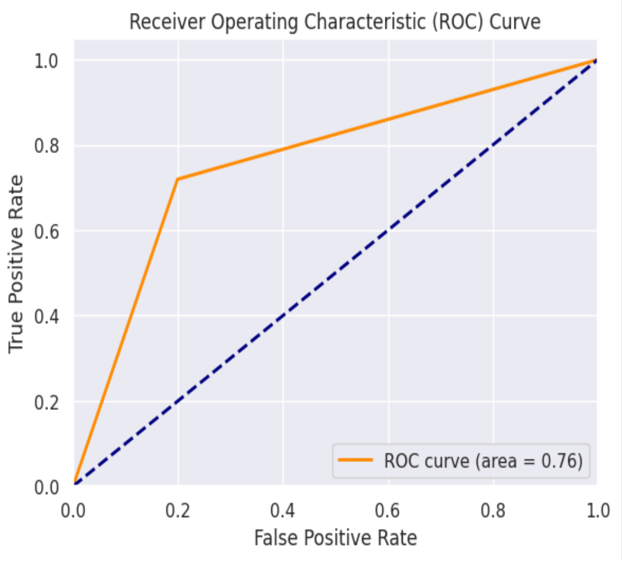
Random Forest Classification

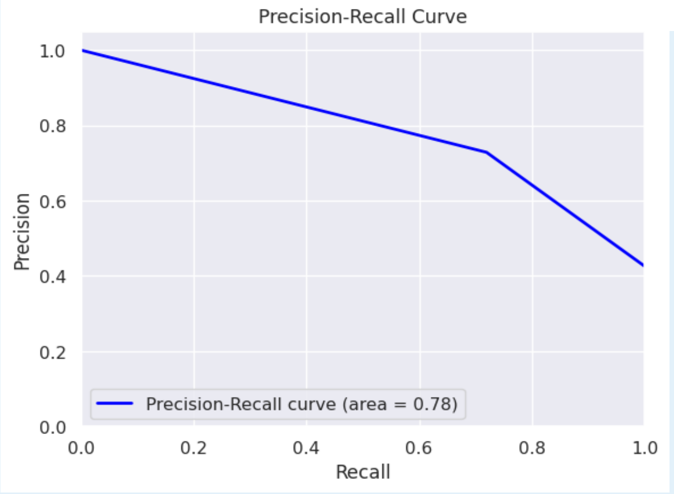


- **The image showcases performance metrics f or a Random Forest classifier used to predict "Disaster" vs. "Non-Disaster" tweets.**
- **It includes a confusion matrix, ROC curve, Precision-Recall curve, and a classification r eport.**
- **The classification report details precision, re call, and F1-score for both classes, with over all accuracy around 78%.**
- **The ROC and Precision-Recall curves indicate the model's ability to separate the two classes effectively.**
- **These visualizations help assess the model's r eliability and potential areas for improvement.**

NLP model MLP Classifier



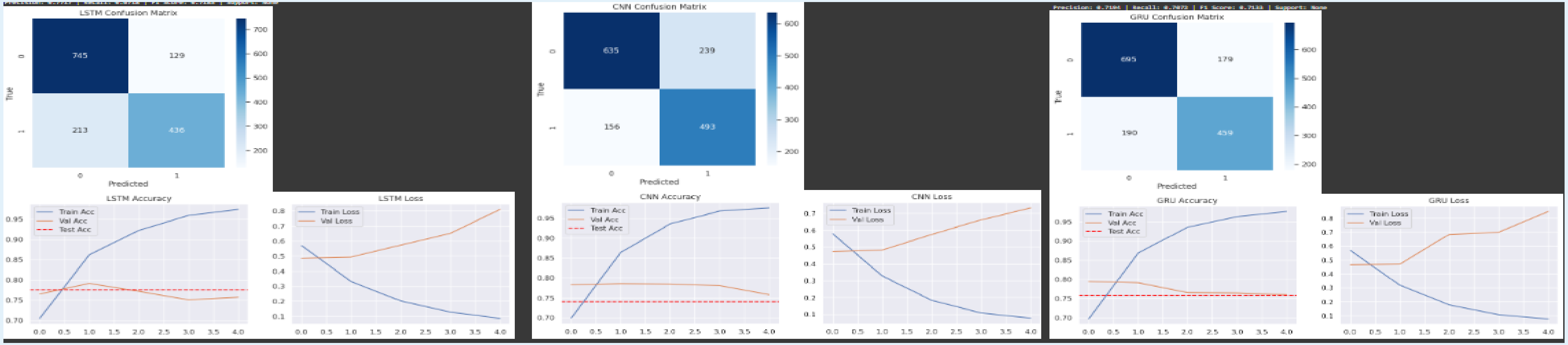





```
Neural Network Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.80      0.80       874
           1       0.73      0.72      0.72       649

    accuracy                           0.77      1523
   macro avg       0.76      0.76      0.76      1523
weighted avg       0.77      0.77      0.77      1523

Train Accuracy (Neural Network): 0.981
Test Accuracy  (Neural Network): 0.766
```

- ❖ The image displays evaluation metrics for a neural network classifier distinguishing between "Disaster" and "Non-Disaster" tweet.
- ❖ It includes a confusion matrix, ROC curve (AUC: 0.76), and Precision-Recall curve (AUC: 0.78).
- ❖ The classification report shows balanced precision, recall, and F1-scores for both classes, with overall test accuracy of 76.6%.
- ❖ Class 0 ("Non-Disaster") slightly out performs Class 1 in all metrics.
- ❖ The train accuracy is notably higher at 98.1%, suggesting potential overfitting.

Performance Comparison: LSTM vs GRU Models



- ❖ The image compares LSTM and GRU models using confusion matrices, accuracy plots, and loss plots.
- ❖ Both models show training and validation accuracy trends across epochs.
- ❖ Loss plots reveal how each model's error decreases during training.
- ❖ Confusion matrices highlight prediction strengths and misclassifications.
- ❖ This visual comparison helps assess which model performs better for classification tasks.

Summary Table of Model Performance

Model	Train Acc	Test Acc	Precision (0)	Recall (0)	F1-Score (0)	Precision (1)	Recall (1)	F1-Score (1)	Macro F1	Weighted F1
Logistic Regression	0.852	0.796	0.78	0.89	0.83	0.82	0.67	0.74	0.78	0.79
Random Forest	0.986	0.779	0.77	0.88	0.82	0.80	0.64	0.71	0.77	0.77
Neural Network	0.981	0.766	0.79	0.80	0.80	0.73	0.72	0.72	0.76	0.77
LSTM	0.937	0.763	0.79	0.80	0.80	0.73	0.71	0.72	0.76	0.76
CNN	0.945	0.784	0.79	0.86	0.82	0.78	0.68	0.73	0.77	0.78
GRU	0.937	0.753	0.80	0.76	0.78	0.70	0.74	0.72	0.75	0.75

Model Performance by Handling Over fitting

Model	Train Accuracy	Test Accuracy	Overfitting Gap	F1 Score (Class 1)
Logistic Regression	0.800	0.840	-0.040	0.75
Random Forest	1.000	0.910	0.090	0.84
MLP Classifier (Sklearn)	1.000	0.975	0.025	0.96
Dense NN (Keras)	0.976	0.925	0.051	0.88

- **Logistic Regression: Balanced performance with minimal overfitting and moderate F1 score (0.75).**
- **Random Forest: High accuracy but shows mild overfitting; F1 score improved to 0.8**
- **MLP Classifier: Best overall performance with highest F1 score (0.96) and minimal overfitting.**
- **Dense NN: Strong performance with good generalization, but slightly lower test accuracy than MLP.**

Web Application Screenshot Using Streamlit

Thank You