PROJECT – 7

1. Introduction

We will be developing a machine learning model capable of accurately classifying tweets as related to real disasters or not.

2. Overview

This project involves the development of a text classification application using an LSTM-based neural network model (though we tried all different method but this one is most suitable). The application is built using Streamlit / dash , allowing users to input text and receive predictions on whether the input describes a disaster or not.

3. Model Architecture The LSTM model consists of:

- An embedding layer with an input vocabulary of 10,000 words and an output dimension of 128.
- A bidirectional LSTM layer with 128 units, followed by a dropout layer (30%).
- A second LSTM layer with 64 units and another dropout layer (30%).
- A third LSTM layer with 32 units and an additional dropout layer (30%).
- A dense layer with 64 neurons and ReLU activation.
- A final output layer with a sigmoid activation function for binary classification.

4. Application Implementation

- The model is saved as "lstm_model.h5" and loaded into the Streamlit /dash application.
- A tokenizer (placeholder in the current implementation) is required for text preprocessing.
- The application accepts user input, tokenizes and pads the text, and feeds it to the model for prediction.
- The output is displayed as "Disaster" if the prediction probability is greater than or equal to 0.5; otherwise, it is "Not a Disaster."

Performing basic task

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
!pip install scipy
```

```
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (1.13.1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in c:\programdata\anaconda3\lib\site-packages (from scipy) (1.26.4)
```

[3]:
```
df = pd.read_csv("twitter_disaster.csv")
```

[4]:
```
df.head
```

[4]:
```
<bound method NDFrame.head of         id keyword location  \
0         1     NaN      NaN
1         4     NaN      NaN
2         5     NaN      NaN
3         6     NaN      NaN
4         7     NaN      NaN
...     ...     ...      ...
7608  10869     NaN      NaN
7609  10870     NaN      NaN
7610  10871     NaN      NaN
7611  10872     NaN      NaN
7612  10873     NaN      NaN

                                                  text  target
0     Our Deeds are the Reason of this #earthquake M...       1
1                Forest fire near La Ronge Sask. Canada       1
2     All residents asked to 'shelter in place' are ...       1
3     13,000 people receive #wildfires evacuation or...       1
4     Just got sent this photo from Ruby #Alaska as ...       1
...                                                 ...     ...
7608  Two giant cranes holding a bridge collapse int...       1
7609  @aria_ahrary @TheTawniest The out of control w...       1
7610  M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...       1
7611  Police investigating after an e-bike collided ...       1
7612  The Latest: More Homes Razed by Northern Calif...       1

[7613 rows x 5 columns]>
```

[5]:
```
df.shape
```

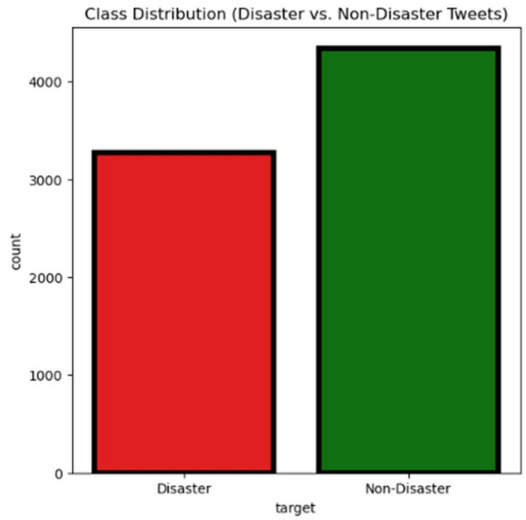
[5]:
```
(7613, 5)
```

[6]:
```
df.isnull().sum()
```

[6]:
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

Plotting bar graph between disaster and non disaster tweets

```
[7]: plt.figure(figsize=(6,6))
sns.countplot(x=df['target'].astype(str), hue=df['target'].astype(str), palette={"0": "green", "1": "red"}, edgecolor="black", linewidth=4, legend=False
plt.xticks(ticks=[0, 1], labels=["Disaster", "Non-Disaster"])
plt.title("Class Distribution (Disaster vs. Non-Disaster Tweets)")
plt.show()
```



Tokenizing and lemmatizing

```
import pandas as pd
import re
import nltk
from collections import Counter
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.util import ngrams
from wordcloud import WordCloud
from sklearn.feature_extraction.text import CountVectorizer

# Download required NLTK resources
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Initialize stopwords and lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

# Function to clean, tokenize, and lemmatize tweets
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'http\S+|www\S+', '', text)  # Remove URLs
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters & numbers

    tokens = word_tokenize(text)  # Tokenization
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]  # Lemmatization & stopword removal

    return ' '.join(tokens)  # Return cleaned text



# Apply preprocessing
df['clean_text'] = df['text'].apply(preprocess_text)

# Filter disaster tweets (target = 1) - assuming we have a target column

disaster_tweets = df[df['target'] == 1]['clean_text']

print(df[['text', 'clean_text']])
```

Output of text and clean text

```
                                                   text  \
0     Our Deeds are the Reason of this #earthquake M...
1                Forest fire near La Ronge Sask. Canada
2     All residents asked to 'shelter in place' are ...
3     13,000 people receive #wildfires evacuation or...
4     Just got sent this photo from Ruby #Alaska as ...
...                                                 ...
7608  Two giant cranes holding a bridge collapse int...
7609  @aria_ahrary @TheTawniest The out of control w...
7610  M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...
7611  Police investigating after an e-bike collided ...
7612  The Latest: More Homes Razed by Northern Calif...

                                             clean_text
0           deed reason earthquake may allah forgive u
1                forest fire near la ronge sask canada
2     resident asked shelter place notified officer ...
3     people receive wildfire evacuation order calif...
4     got sent photo ruby alaska smoke wildfire pour...
...                                                 ...
7608  two giant crane holding bridge collapse nearby...
7609  ariaahrary thetawniest control wild fire calif...
7610                            utckm volcano hawaii
7611  police investigating ebike collided car little...
7612  latest home razed northern california wildfire...

[7613 rows x 2 columns]
```

```
df['clean_text']
```

```
0             deed reason earthquake may allah forgive u
1                  forest fire near la ronge sask canada
2       resident asked shelter place notified officer ...
3       people receive wildfire evacuation order calif...
4       got sent photo ruby alaska smoke wildfire pour...
                              ...
7608    two giant crane holding bridge collapse nearby...
7609    ariaahrary thetawniest control wild fire calif...
7610                              utckm volcano hawaii
7611    police investigating ebike collided car little...
7612    latest home razed northern california wildfire...
Name: clean_text, Length: 7613, dtype: object
```

Saving tokenizer

```
import pickle
import os
# Load or create tokenizer
TOKENIZER_PATH = "tokenizer.pkl"
if os.path.exists(TOKENIZER_PATH):
    with open(TOKENIZER_PATH, "rb") as f:
        tokenizer = pickle.load(f)
else:
    tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")
    tokenizer.fit_on_texts(df['clean_text'])
    with open(TOKENIZER_PATH, "wb") as f:
        pickle.dump(tokenizer, f)
```

Most frequent used words

```
# Display top 10 most common words
print("Most Common Words in Disaster Tweets:")
print(word_freq.most_common(10))
```

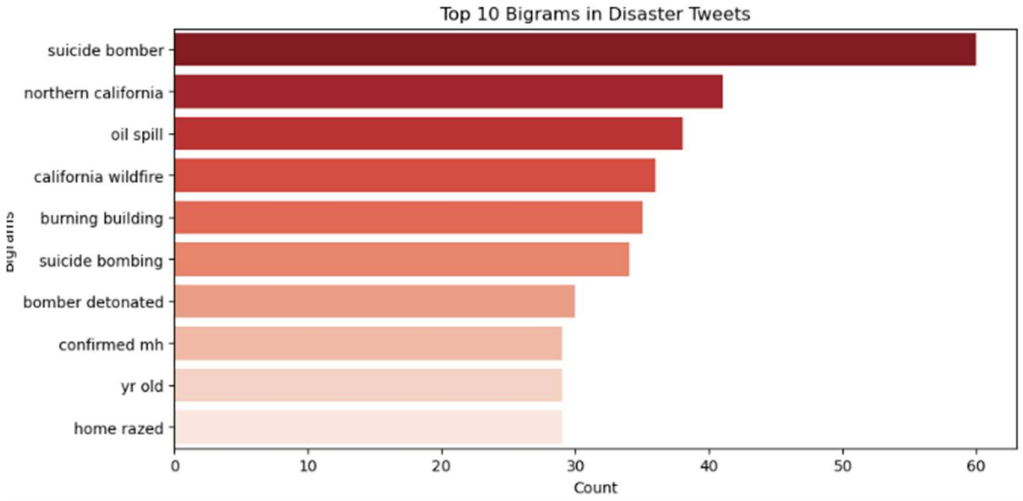
```
Most Common Words in Disaster Tweets:
[('fire', 264), ('news', 140), ('via', 121), ('disaster', 118), ('california', 115), ('suicide', 110), ('u', 107), ('police', 107), ('amp', 107), ('people', 106)]
```

Word cloud

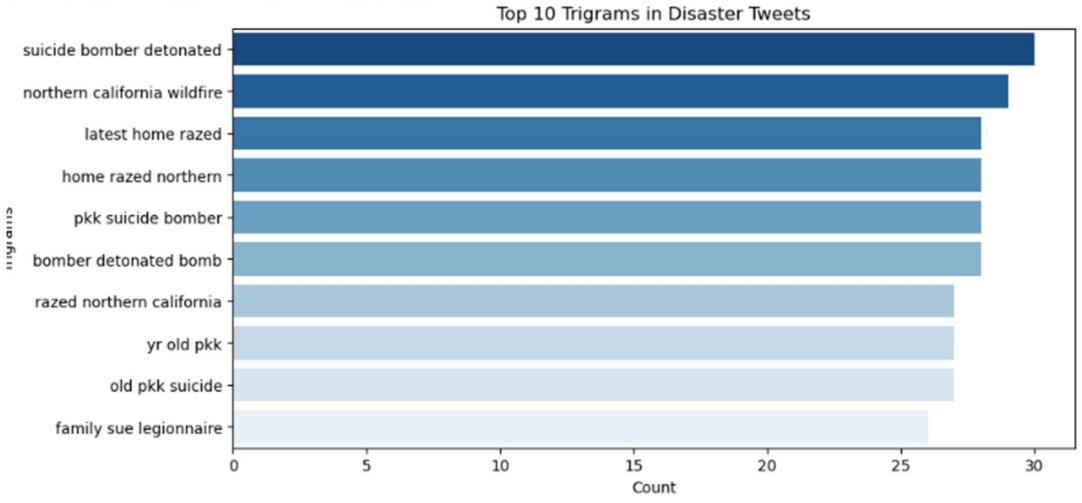
```
# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='black', colormap='Reds').generate(' '.join(disaster_tweets))

# Display word cloud
plt.figure(figsize=(10,5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Word Cloud of Disaster Tweets", fontsize=14)
plt.show()
```

Bigram and Trigram

```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert bigrams & trigrams to DataFrame
bigram_df = pd.DataFrame(bigrams, columns=['Phrase', 'Count'])
trigram_df = pd.DataFrame(trigrams, columns=['Phrase', 'Count'])

# Convert tuple phrases to strings
bigram_df['Phrase'] = bigram_df['Phrase'].apply(lambda x: ' '.join(x))
trigram_df['Phrase'] = trigram_df['Phrase'].apply(lambda x: ' '.join(x))

# Plot Bigrams
plt.figure(figsize=(10,5))
sns.barplot(x='Count', y='Phrase', data=bigram_df, palette="Reds_r")
plt.title("Top 10 Bigrams in Disaster Tweets")
plt.xlabel("Count")
plt.ylabel("Bigrams")
plt.show()

# Plot Trigrams
plt.figure(figsize=(10,5))
sns.barplot(x='Count', y='Phrase', data=trigram_df, palette="Blues_r")
plt.title("Top 10 Trigrams in Disaster Tweets")
plt.xlabel("Count")
plt.ylabel("Trigrams")
plt.show()
```




```
:\Users\ppppr\AppData\Local\Temp\ipykernel_2584\656031064.py:22: FutureWarning:

assing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for
ame effect.

  sns.barplot(x='Count', y='Phrase', data=trigram_df, palette="Blues_r")
```



Counter vectorizing and TfidVectorizing

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# Initialize vectorizers
count_vectorizer = CountVectorizer(max_features=5000)
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Apply vectorization
X_word_freq = count_vectorizer.fit_transform(df['clean_text'])
X_tfidf = tfidf_vectorizer.fit_transform(df['clean_text'])

print("Shape of Word Frequency Features:", X_word_freq.shape)
print("Shape of TF-IDF Features:", X_tfidf.shape)
```

```
Shape of Word Frequency Features: (7613, 5000)
Shape of TF-IDF Features: (7613, 5000)
```

Extracting extra features

```
import re

# Function to extract additional features
def extract_features(text):
    tweet_length = len(text)  # Character count
    word_count = len(text.split())  # Word count
    hashtag_count = len(re.findall(r"#\w+", text))  # Count hashtags
    mention_count = len(re.findall(r"@\w+", text))  # Count mentions
    return tweet_length, word_count, hashtag_count, mention_count

# Apply feature extraction
df[['tweet_length', 'word_count', 'hashtag_count', 'mention_count']] = df['text'].apply(lambda x: pd.Series(extract_features(x)))

print(df[['text', 'tweet_length', 'word_count', 'hashtag_count', 'mention_count']].head())


df1[['tweet_length', 'word_count', 'hashtag_count', 'mention_count']] = df1['text'].apply(lambda x: pd.Series(extract_features(x)))

print(df1[['text', 'tweet_length', 'word_count', 'hashtag_count', 'mention_count']].head())


df2[['tweet_length', 'word_count', 'hashtag_count', 'mention_count']] = df2['text'].apply(lambda x: pd.Series(extract_features(x)))

print(df2[['text', 'tweet_length', 'word_count', 'hashtag_count', 'mention_count']].head())
```

```
   word_count  hashtag_count  mention_count
0          13              1              0
1           7              0              0
2          22              0              0
3           8              1              0
4          16              2              0
                                                text  tweet_length  \
0  Our Deeds are the Reason of this #earthquake M...            69
1              Forest fire near La Ronge Sask. Canada            38
2  All residents asked to 'shelter in place' are ...           133
3  13,000 people receive #wildfires evacuation or...            65
4  Just got sent this photo from Ruby #Alaska as ...            88

   word_count  hashtag_count  mention_count
0          13              1              0
1           7              0              0
2          22              0              0
3           8              1              0
4          16              2              0
```

Splitting data

```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, random_state=42)


print("Final Feature Shape:", X_train.shape)  # Includes embeddings + extra features

X_test.shape
```

Now trying different models

```
classifier = MultinomialNB()
classifier.fit(X_train1, y_train1)
```

```
MultinomialNB
MultinomialNB()
```

```
y_pred1 = classifier.predict(X_test1)
```

```
print("Accuracy:", accuracy_score(y_test1, y_pred1))
print("Precision:",precision_score(y_test1, y_pred1))
print("recall:",recall_score(y_test1, y_pred1))
print("f1score:",f1_score(y_test1, y_pred1))
```

```
Accuracy: 0.5489166119500984
Precision: 0.4736111111111111
recall: 0.5254237288135594
f1score: 0.4981738495252009
```

Linear regression

```
log_model = LogisticRegression()
log_model.fit(X_train, y_train)
y_pred_log = log_model.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_log))
```

```
Logistic Regression Accuracy: 0.5705843729481287
```

Random forest

```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.6841759684832567
```

Graientboosting

```
gb_model1 = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_model1.fit(X_train1, y_train1)
y_pred_gb1 = gb_model1.predict(X_test1)
print("Gradient Boosting Accuracy:", accuracy_score(y_test1, y_pred_gb1))
```

```
Gradient Boosting Accuracy: 0.690741956664478
```

Lstm model

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
```

```
model = Sequential([
    Embedding(input_dim=10000, output_dim=128, input_length=100),
    Bidirectional(LSTM(128, return_sequences=True)),
    Dropout(0.3),
    LSTM(64, return_sequences=True),
    Dropout(0.3),
    LSTM(32),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Epoch 28/30
191/191 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9745 - loss: 0.0671 - val_accuracy: 0.7413 - val_loss: 1.1105
Epoch 29/30
191/191 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9752 - loss: 0.0687 - val_accuracy: 0.7255 - val_loss: 1.1323
Epoch 30/30
191/191 ━━━━━━━━━━━━━━━━━━━━ 6s 30ms/step - accuracy: 0.9779 - loss: 0.0584 - val_accuracy: 0.7472 - val_loss: 1.0984
<keras.src.callbacks.history.History at 0x1dc1aa19220>
```

LSTM model is most accurate

Creating Dash interference LSTM model with h.5

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import pickle


model.save("lstm_model.h5")

# Load the model for prediction
loaded_model = load_model("lstm_model.h5")

# Dummy tokenizer (replace with actual tokenizer used during training)
tokenizer = {"word_index": {"example": 1}}  # Placeholder

# Dash App
app = dash.Dash(__name__)
app.layout = html.Div([
    html.H1("LSTM Model Prediction"),
    dcc.Input(id='text_input', type='text', placeholder='Enter text...'),
    html.Button('Predict', id='predict_button', n_clicks=0),
    html.Div(id='prediction_output')
])

@app.callback(
    Output('prediction_output', 'children'),
    Input('predict_button', 'n_clicks'),
    Input('text_input', 'value')
)
def predict(n_clicks, text):
    if n_clicks > 0 and text:
        # Preprocess input text (dummy processing, replace with actual tokenizer)
        sequence = np.array([[tokenizer['word_index'].get(word, 0) for word in text.split()]])
        sequence = tf.keras.preprocessing.sequence.pad_sequences(sequence, maxlen=100)
        prediction = loaded_model.predict(sequence)[0][0]
        result = "Disaster" if prediction >= 0.5 else "Not a Disaster"
        return f'Prediction: {result}'
    return ''

if __name__ == '__main__':
    app.run_server(debug=True)
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```python
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import model_from_json
import pickle


# Save the model using Pickle format (separate weights and architecture)
def save_model(model, filename="model.pkl"):
    # Save the model architecture (JSON format)
    model_json = model.to_json()


    # Save architecture using Pickle
    with open(filename, "wb") as f:
        pickle.dump(model_json, f)


# Load the model from Pickle
def load_model_from_pickle(filename="model.pkl"):
    # Load model architecture (JSON format)
    with open(filename, "rb") as f:
        model_json = pickle.load(f)

    # Recreate the model from the architecture
    model = model_from_json(model_json)


    return model



save_model(model)

# Load the model for prediction (in your app)
loaded_model = load_model_from_pickle()

# Dummy tokenizer (replace with actual tokenizer used during training)
tokenizer = {"word_index": {"example": 1}}  # Placeholder

# Dash App
app = dash.Dash(__name__)
app.layout = html.Div([
    html.H1("LSTM Model Prediction"),
    dcc.Input(id='text_input', type='text', placeholder='Enter text...'),
    html.Button('Predict', id='predict_button', n_clicks=0),
    html.Div(id='prediction_output')
])

@app.callback(
    Output('prediction_output', 'children'),
    Input('predict_button', 'n_clicks'),
    Input('text_input', 'value')
)
def predict(n_clicks, text):
    if n_clicks > 0 and text:
        # Preprocess input text (dummy processing, replace with actual tokenizer)
        sequence = np.array([[tokenizer['word_index'].get(word, 0) for word in text.split()]])
        sequence = tf.keras.preprocessing.sequence.pad_sequences(sequence, maxlen=100)
        prediction = loaded_model.predict(sequence)[0][0]
        result = "Disaster" if prediction >= 0.5 else "Not a Disaster"
        return f'Prediction: {result}'
    return ''

if __name__ == '__main__':
    app.run_server(debug=True)
```

LSTM model as pkl file

Using streamlit

```python
import streamlit as st
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
import pickle

# Load the model for prediction
loaded_model = load_model("lstm_model.h5")

# Dummy tokenizer (replace with actual tokenizer used during training)
tokenizer = {"word_index": {"example": 1}}  # Placeholder

# Streamlit App
st.title("LSTM Model Prediction")
text_input = st.text_input("Enter text:")
if st.button("Predict"):
    if text_input:
        # Preprocess input text (dummy processing, replace with actual tokenizer)
        sequence = np.array([[tokenizer['word_index'].get(word, 0) for word in text_input.split()]])
        sequence = tf.keras.preprocessing.sequence.pad_sequences(sequence, maxlen=100)
        prediction = loaded_model.predict(sequence)[0][0]
        result = "Disaster" if prediction >= 0.5 else "Not a Disaster"
        st.write(f'Prediction: {result}')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
2025-02-02 19:07:36.252 WARNING streamlit.runtime.scriptrunner_utils.script_run_context: Thread 'MainThread': missing ScriptRunContext! This warning can
be ignored when running in bare mode.
2025-02-02 19:07:36.361
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run C:\ProgramData\anaconda3\Lib\site-packages\ipykernel_launcher.py [ARGUMENTS]
2025-02-02 19:07:36.363 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.363 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.364 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.364 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.365 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.365 Session state does not function when running a script without `streamlit run`
2025-02-02 19:07:36.366 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.366 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.367 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.367 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.367 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.368 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
2025-02-02 19:07:36.369 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
```

Key Features

- User-friendly interface built with Streamlit.
- Real-time text classification.
- Simple model loading and prediction execution.
- Scalable for integration with other NLP applications.

Conclusion

This project demonstrates an effective approach to classifying text as disaster-related or not using an LSTM-based model. Further improvements in data preprocessing and UI enhancements can make the application more robust and user-friendly.