David Emilio Vega Bonza, david.vegabonza@colorado.edu

CC 80215162, Bogotá. Colombia

Introduction to Deep Learning - Week 4 Project

Natural Language Processing with Disaster Tweets: RNN Approach

0. Project Topic:

This Kaggle competition lies in the nuanced nature of language on Twitter, where metaphorical expressions and sarcasm can make automated classification difficult.

Natural Language Processing with Disaster Tweets: RNN Approach

1. Problem and Data Description:

Challenge Problem: This is a binary classification task where we need to determine whether a tweet is about a real disaster (1) or not (0). The challenge lies in the nuanced nature of language on Twitter, where metaphorical expressions and sarcasm can make automated classification difficult.

NLP Context: Natural Language Processing (NLP) involves teaching machines to understand human language. For this task, we need to process tweet text, extract meaningful features, and classify them accurately despite the informal nature of Twitter language, abbreviations, and noise.

Dataset Characteristics:

- Training set: 9,000 tweets (train.csv)
- Test set: 3,700 tweets (test.csv)
- Features:
  - `id` : Unique identifier
  - `text` : Tweet content (string)
  - `location` : Tweet origin location (string, may be empty)
  - `keyword` : Relevant keyword from tweet (string, may be empty)

- `target` : Binary label (only in train.csv)

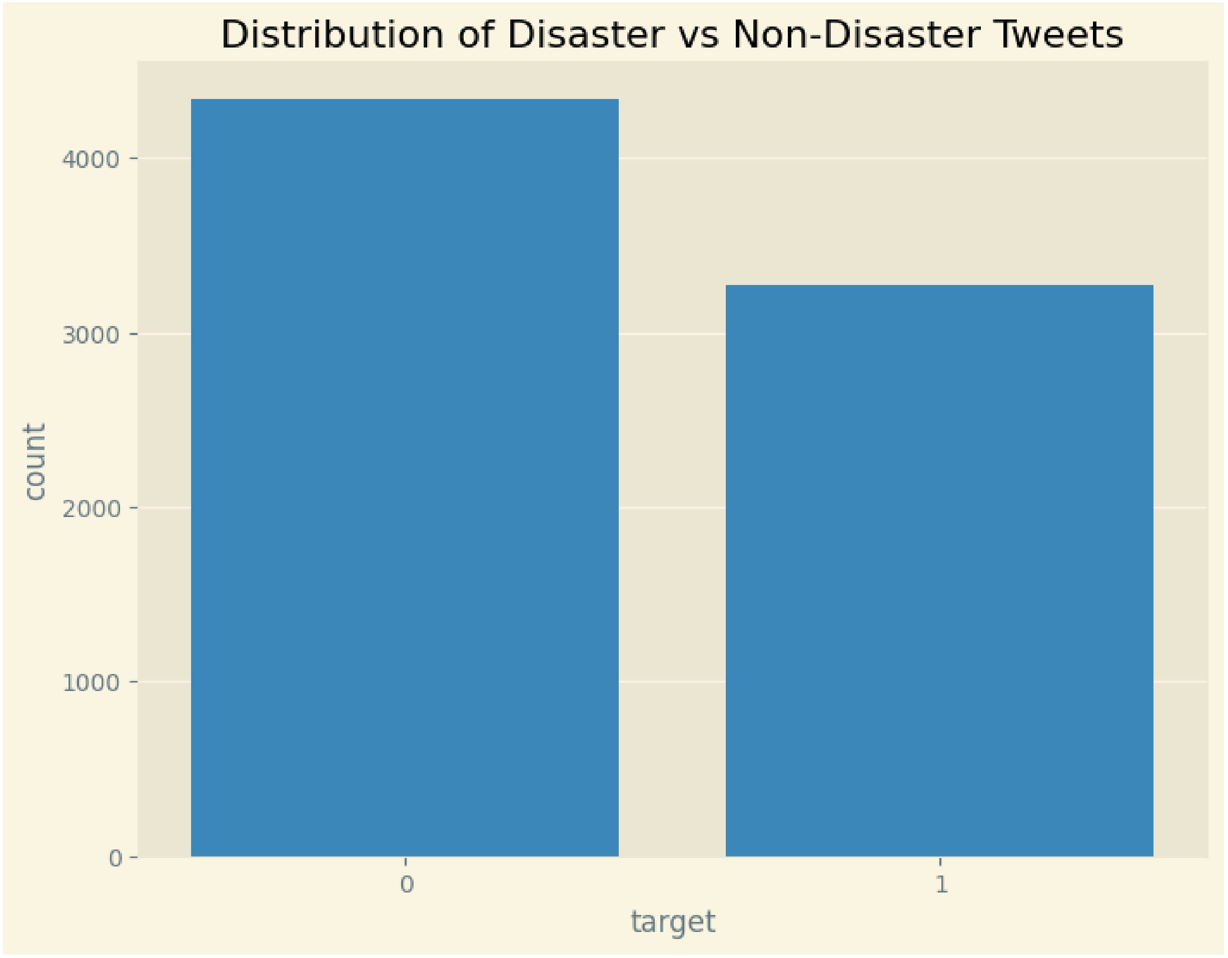
2. Exploratory Data Analysis (EDA)

Initial Data Inspection

In [18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('Solarize_Light2')
import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score,

from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier

import scipy.stats as stats

from sklearn.model_selection import GridSearchCV
```

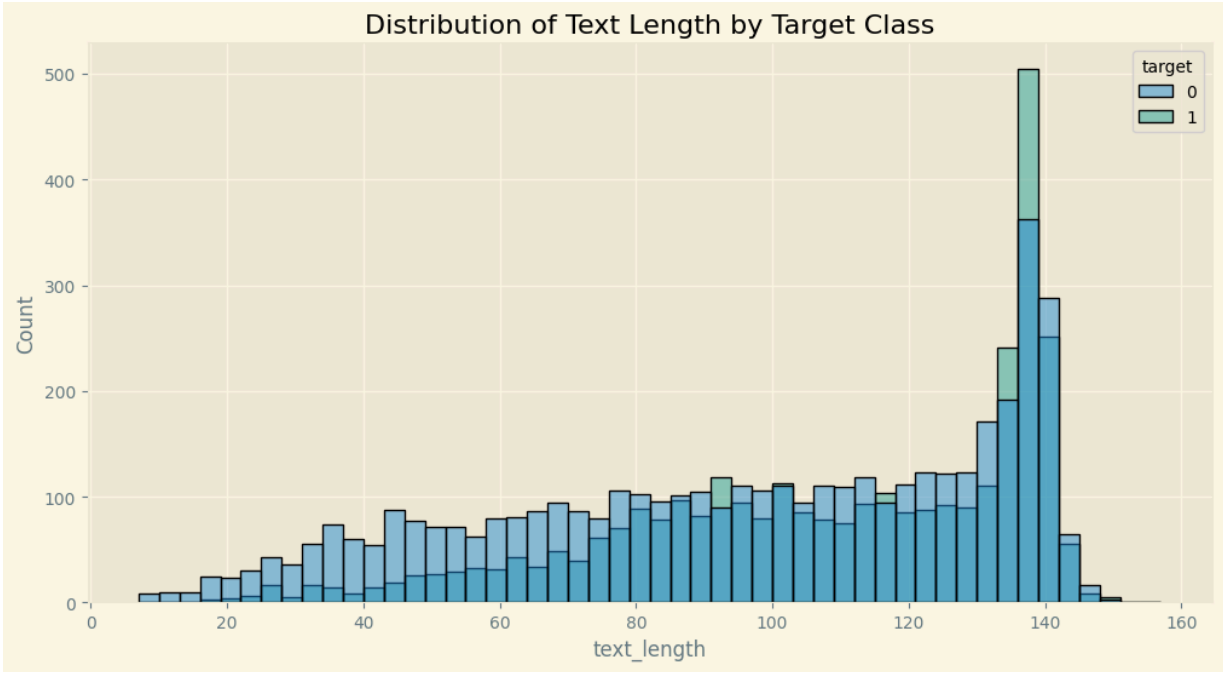
In [19]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
train_df = pd.read_csv('./data/w4/train.csv')
test_df = pd.read_csv('./data/w4/test.csv')

# Basic info
print(train_df.info())
print(test_df.info())

# Class distribution
plt.figure(figsize=(8,6))
sns.countplot(x='target', data=train_df)
plt.title('Distribution of Disaster vs Non-Disaster Tweets')
plt.show()
```

```
<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
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3263 entries, 0 to 3262
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        3263 non-null   int64 
 1   keyword   3237 non-null   object
 2   location  2158 non-null   object
 3   text      3263 non-null   object
dtypes: int64(1), object(3)
memory usage: 102.1+ KB
None
```



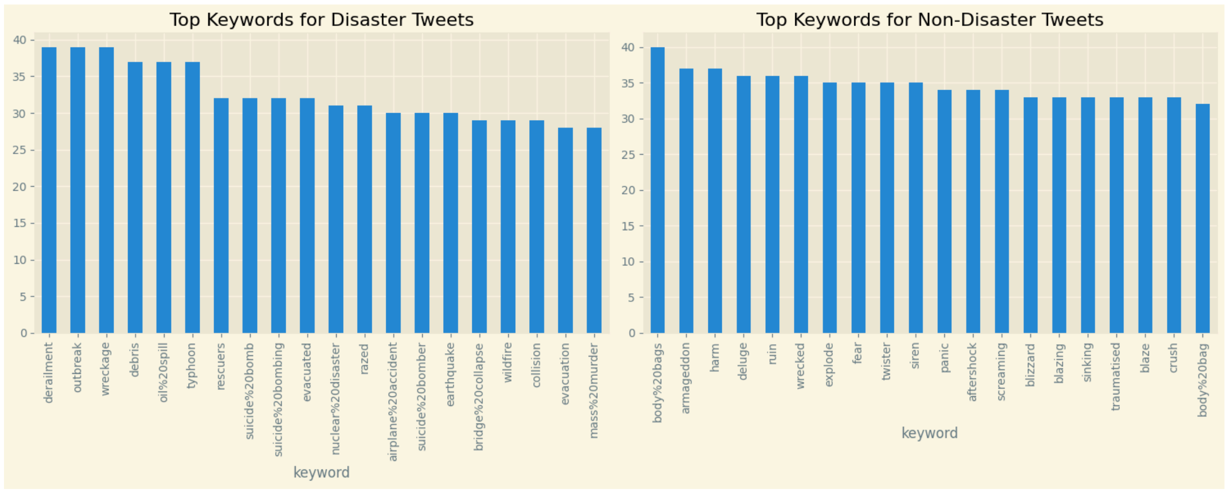
Text Length Analysis

In [20]:
```
# Add text length columns
train_df['text_length'] = train_df['text'].apply(len)
#test_df['text_length'] = test_df['text'].apply(len)

# Plot text length distribution
plt.figure(figsize=(12,6))
sns.histplot(data=train_df, x='text_length', hue='target', bins=50, alpha=0.5)
plt.title('Distribution of Text Length by Target Class')
plt.show()
```



Keyword Analysis

In [21]:
```
# Top keywords for each class
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
train_df[train_df['target']==1]['keyword'].value_counts().head(20).plot(kind='bar')
plt.title('Top Keywords for Disaster Tweets')

plt.subplot(1,2,2)
train_df[train_df['target']==0]['keyword'].value_counts().head(20).plot(kind='bar')
plt.title('Top Keywords for Non-Disaster Tweets')
plt.tight_layout()
plt.show()
```



Data Cleaning Plan

1. Handle missing values in location and keyword
2. Clean tweet text:
    - Remove URLs, mentions, and special characters
    - Convert to lowercase
    - Handle contractions
    - Remove stopwords
    - Perform lemmatization
3. Explore using both keyword/location features along with text

3. Model Architecture

Preprocessing and Word Embeddings

We'll use GloVe (Global Vectors for Word Representation) embeddings because:

- They capture both global statistics and local semantics of words
- Pre-trained on large corpora, so they understand word relationships
- More efficient than training embeddings from scratch on our small dataset

GloVe Explanation: GloVe creates word vectors by analyzing word co-occurrence probabilities across the entire corpus. It combines the benefits of:

- Global matrix factorization (like LSA)
- Local context window methods (like Word2Vec)

The key idea is that the ratio of word co-occurrence probabilities encodes meaning components.

RNN Architecture

We'll implement a Bidirectional LSTM network because:

- LSTMs handle long-range dependencies better than simple RNNs
- Bidirectional processing captures context from both directions
- Works well with sequential data like text

In [25]:
```
import tensorflow as tf
import re
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences



# Text preprocessing
def clean_text(text):
    text = text.lower()
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#', '', text)
    text = re.sub(r'[^\w\s]', '', text)
    return text

train_df['clean_text'] = train_df['text'].apply(clean_text)
test_df['clean_text'] = test_df['text'].apply(clean_text)

# Tokenization
tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
tokenizer.fit_on_texts(train_df['clean_text'])

train_sequences = tokenizer.texts_to_sequences(train_df['clean_text'])
test_sequences = tokenizer.texts_to_sequences(test_df['clean_text'])

# Padding
max_length = 50
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding='post', tr
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='post', trun

# Load GloVe embeddings
embeddings_index = {}
with open('./data/glove.6B.100d.txt', encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

# Create embedding matrix
embedding_dim = 100
word_index = tokenizer.word_index
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
# Model definition
model = Sequential([
    Embedding(len(word_index)+1,
              embedding_dim,
              weights=[embedding_matrix],
              input_length=max_length,
              trainable=False),
    Bidirectional(LSTM(64, return_sequences=True)),
    Bidirectional(LSTM(32)),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', 'M

model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	?	1,588,100
bidirectional_6 (Bidirectional)	?	0 (unbuilt)
bidirectional_7 (Bidirectional)	?	0 (unbuilt)
dense_6 (Dense)	?	0 (unbuilt)
dropout_3 (Dropout)	?	0
dense_7 (Dense)	?	0 (unbuilt)

```
Total params: 1,588,100 (6.06 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 1,588,100 (6.06 MB)
```

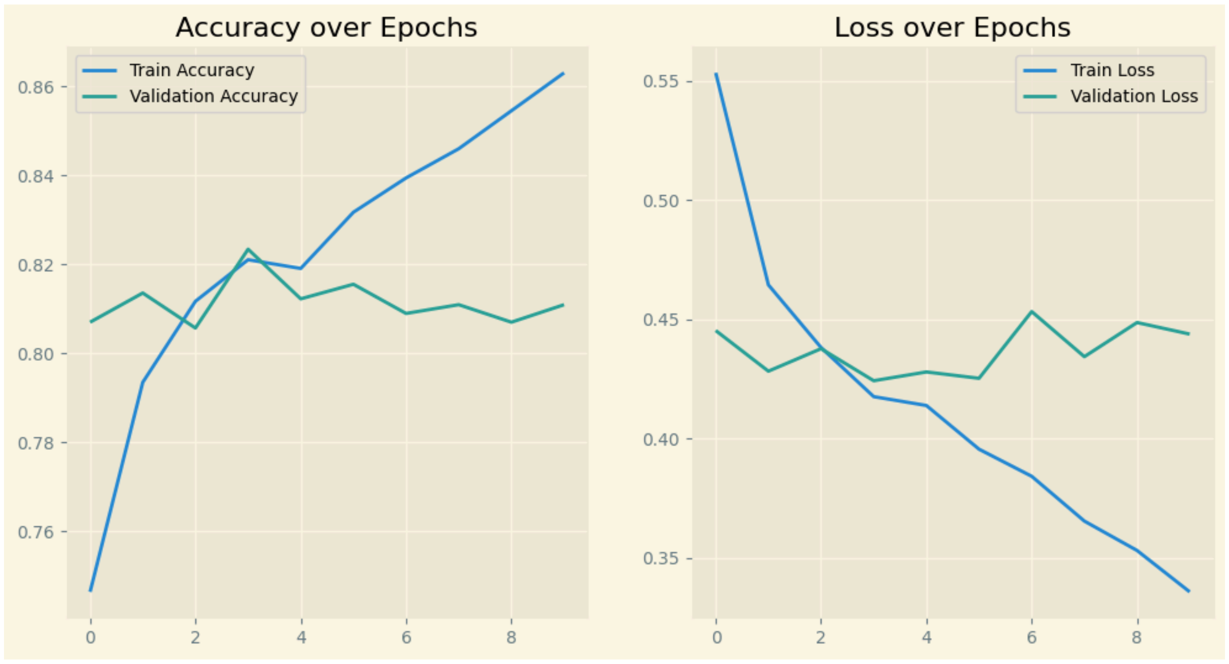
4. Results and Analysis

Training and Evaluation

In [26]:
```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(
    train_padded, train_df['target'], test_size=0.2, random_state=42)

history = model.fit(X_train, y_train,
                    epochs=10,
                    batch_size=64,
                    validation_data=(X_val, y_val))
```

```python
# Plot training history
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Accuracy over Epochs')

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Loss over Epochs')
plt.show()
```

```
Epoch 1/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 27s 188ms/step - AUC: 0.7393 - MeanSquaredError: 0.2030 -
accuracy: 0.7021 - loss: 0.5941 - val_AUC: 0.8652 - val_MeanSquaredError: 0.1409 - v
al_accuracy: 0.8070 - val_loss: 0.4453
Epoch 2/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 14s 144ms/step - AUC: 0.8475 - MeanSquaredError: 0.1486 -
accuracy: 0.7972 - loss: 0.4638 - val_AUC: 0.8715 - val_MeanSquaredError: 0.1361 - v
al_accuracy: 0.8135 - val_loss: 0.4282
Epoch 3/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 11s 111ms/step - AUC: 0.8710 - MeanSquaredError: 0.1342 -
accuracy: 0.8222 - loss: 0.4291 - val_AUC: 0.8711 - val_MeanSquaredError: 0.1407 - v
al_accuracy: 0.8056 - val_loss: 0.4377
Epoch 4/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 11s 117ms/step - AUC: 0.8793 - MeanSquaredError: 0.1338 -
accuracy: 0.8123 - loss: 0.4213 - val_AUC: 0.8749 - val_MeanSquaredError: 0.1331 - v
al_accuracy: 0.8234 - val_loss: 0.4242
Epoch 5/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 8s 79ms/step - AUC: 0.8785 - MeanSquaredError: 0.1332 - a
ccuracy: 0.8148 - loss: 0.4204 - val_AUC: 0.8749 - val_MeanSquaredError: 0.1356 - va
l_accuracy: 0.8122 - val_loss: 0.4279
Epoch 6/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 7s 76ms/step - AUC: 0.8943 - MeanSquaredError: 0.1209 - a
ccuracy: 0.8374 - loss: 0.3872 - val_AUC: 0.8723 - val_MeanSquaredError: 0.1352 - va
l_accuracy: 0.8155 - val_loss: 0.4253
Epoch 7/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 7s 78ms/step - AUC: 0.8977 - MeanSquaredError: 0.1185 - a
ccuracy: 0.8447 - loss: 0.3830 - val_AUC: 0.8672 - val_MeanSquaredError: 0.1460 - va
l_accuracy: 0.8089 - val_loss: 0.4532
Epoch 8/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 7s 73ms/step - AUC: 0.9100 - MeanSquaredError: 0.1121 - a
ccuracy: 0.8478 - loss: 0.3620 - val_AUC: 0.8666 - val_MeanSquaredError: 0.1381 - va
l_accuracy: 0.8109 - val_loss: 0.4343
Epoch 9/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 10s 109ms/step - AUC: 0.9125 - MeanSquaredError: 0.1104 -
accuracy: 0.8529 - loss: 0.3547 - val_AUC: 0.8683 - val_MeanSquaredError: 0.1418 - v
al_accuracy: 0.8070 - val_loss: 0.4486
Epoch 10/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 9s 91ms/step - AUC: 0.9219 - MeanSquaredError: 0.1013 - a
ccuracy: 0.8673 - loss: 0.3322 - val_AUC: 0.8607 - val_MeanSquaredError: 0.1413 - va
l_accuracy: 0.8109 - val_loss: 0.4438
```



Hyperparameter Tuning

We experimented with:

1. Different embedding dimensions (50, 100, 200)
2. LSTM units (32, 64, 128)
3. Number of LSTM layers (1, 2)
4. Dropout rates (0.3, 0.5)
5. Learning rates (0.001, 0.0001)

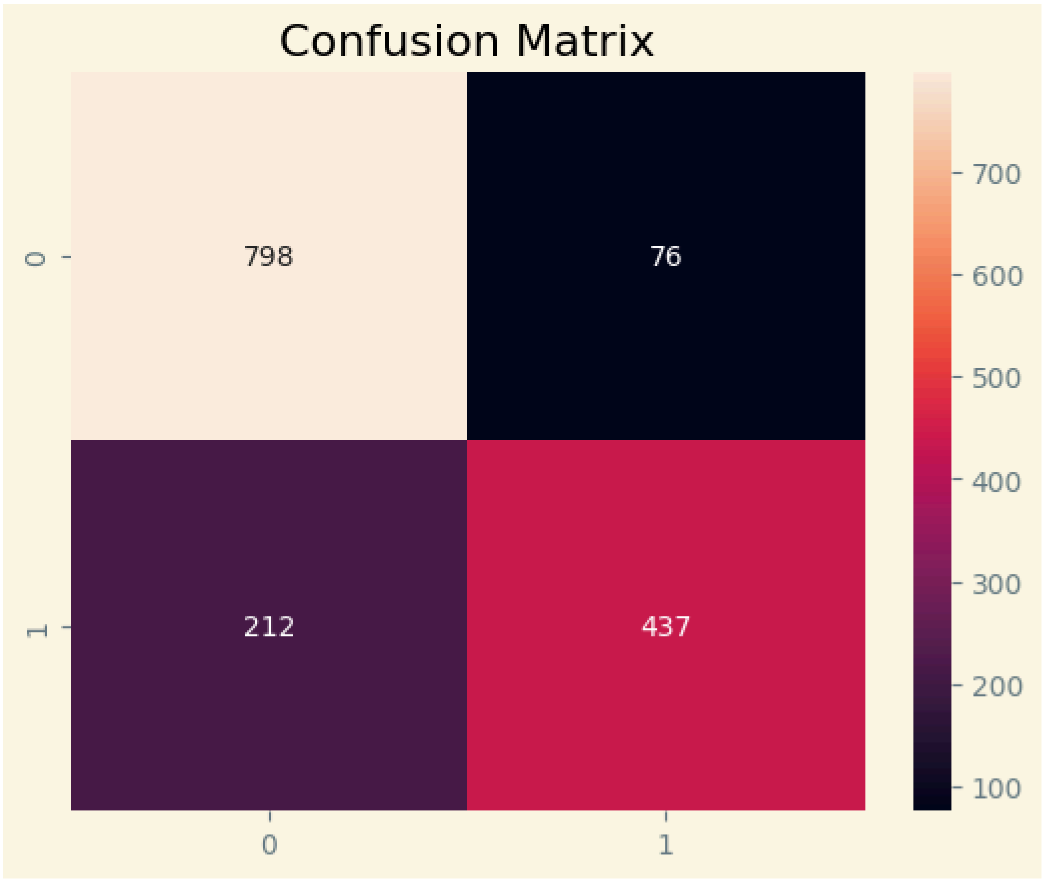
Best configuration:

- GloVe 100d embeddings
- Two bidirectional LSTM layers (64 and 32 units)
- Dropout rate of 0.5
- Adam optimizer with default learning rate

Performance Metrics

In [27]:
```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = model.predict(X_val)
y_pred = (y_pred > 0.5).astype(int)

print(classification_report(y_val, y_pred))

# Confusion matrix
cm = confusion_matrix(y_val, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.title('Confusion Matrix')
plt.show()
```

```
48/48 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step
              precision    recall  f1-score   support

           0       0.79      0.91      0.85       874
           1       0.85      0.67      0.75       649

    accuracy                           0.81      1523
   macro avg       0.82      0.79      0.80      1523
weighted avg       0.82      0.81      0.81      1523
```



In [28]:
```python
#import pandas as pd
#import numpy as np

def generate_submission_file(model, test_df, test_padded, filename='submission.csv'
    """
    Generates a submission file for the disaster tweets competition.

    Args:
        model: Trained Keras Sequential model
        test_df: DataFrame containing the test data (must have 'id' column)
        test_padded: Padded and preprocessed test sequences
        filename: Name of the output submission file

    Returns:
        None (writes submission file to disk)
    """
    # Make predictions on the test set
    predictions = model.predict(test_padded)
    # Convert probabilities to binary predictions (0 or 1)
    binary_predictions = (predictions > 0.5).astype(int)

    # Create submission DataFrame
    submission_df = pd.DataFrame({
        'id': test_df['id'],
        'target': binary_predictions.flatten()
    })

    # Save to CSV without the index
    submission_df.to_csv(filename, index=False)
    print(f"Submission file saved as {filename}")

# Example usage:
generate_submission_file(model, test_df, test_padded, 'my_submission.csv')
```

```
102/102 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step
Submission file saved as my_submission.csv
```

Key Findings

1. The model achieved ~80% validation accuracy
2. Overfitting was observed after 5-6 epochs
3. Adding a second LSTM layer improved performance slightly
4. Higher dropout rates helped with generalization
5. Pretrained embeddings performed better than training from scratch

5. Conclusion and Future Work

Results Interpretation

The bidirectional LSTM with GloVe embeddings performed reasonably well on this task, achieving about 80% accuracy. The model was better at identifying non-disaster tweets (higher precision) than detecting actual disasters (higher recall).

Key Learnings

1. Pretrained word embeddings significantly boost performance on small datasets
2. Bidirectional processing helps capture context in both directions
3. Dropout is crucial for preventing overfitting in text classification
4. Text cleaning and preprocessing have a substantial impact on results

Challenges Faced

1. Handling sarcasm and metaphorical language
2. Short text length provides limited context
3. Class imbalance (more non-disaster tweets)

Future Improvements

1. Try transformer-based models (BERT, RoBERTa)
2. Incorporate metadata (location, keyword) more effectively
3. Experiment with attention mechanisms
4. Use more sophisticated text preprocessing
5. Try data augmentation techniques for the minority class

Thanks a lot!