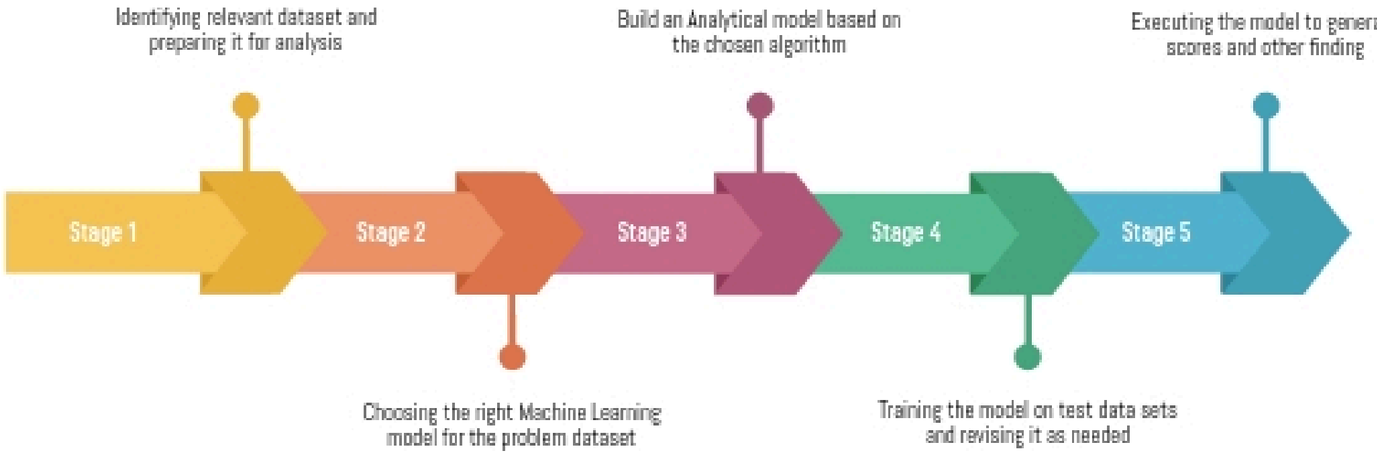
MGMT 59000: Machine Learning

Disaster Tweets Classification

Danish Muhammad Azhar
Tripta Bhattacherjee

Primary Objective
(Leveraging Natural Language Processing)

Social media provides real-time crisis updates, but not all disaster-related tweets are real. This project uses NLP and machine learning to classify 10,000 hand-labeled tweets as actual disasters or not.



Stage #1: Data Preprocessing

Cleaning Data for Analysis

Contraction Replacement

Purpose: Expands shortened forms to improve consistency and accuracy in text analysis.
Code: Used dictionary “contraction_dict”
Example: "He ain’t going to survive" becomes "He is not going to survive"

Punctuation Removal

Purpose: Removes noise into text classification tasks.
Code: Uses “string.punctuation” and a list comprehension.
Example: "The fire, as I saw, was spreading!" becomes "The fire as I saw was spreading"

Lowercasing

Purpose: Ensures that the same word, regardless of capitalization, is treated as the same token.
Code: Converts all text to lowercase using .lower()
Example: “Fire” becomes “fire”

Hashtag & URL removal

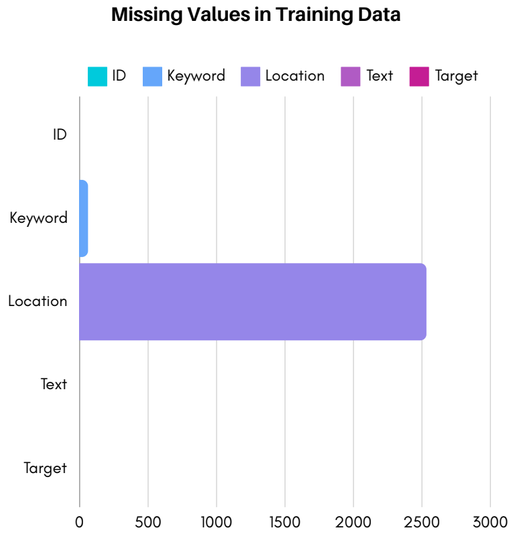
Purpose: Not useful for text classification, thus retained.
Code: Uses “re.sub”
Example: “#Thunder” and “https/www.fire.com” becomes “Thunder” and “URL”

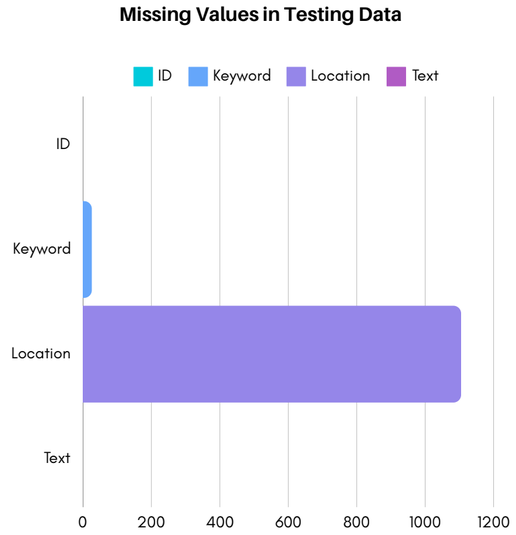
Mention Removal & Whitespace Reduction

Purpose: Reduces noise and disruption and focuses on content of the tweet
Code: Uses “re.sub”
Example: “@NYPD” becomes NYPD, and “Emergency call” becomes “Emergency call”

Stopword and Other Punctuation Removal

Purpose: Removes common English stopwords (e.g., "the", "a", "is") so focus is on keywords
Code: Uses “nltk.corpus.stopwords”
Example: “There is a forest fire near LA” becomes “forest fire near LA”.





Stage #2: Exploratory Data Analysis (EDA)
Keyword Analysis

Disaster Tweets

The most frequently used **keywords in Disaster tweets** are:

- Fatalities
- Armaggedon
- Deluge
- Harm
- Damage
- Suicide
- Typhoon
- Fire
- Bombing
- Wreckage

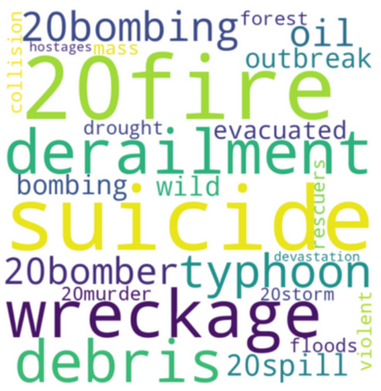
Non-Disaster Tweets

The most frequently used **keywords in Non-Disaster tweets** are:

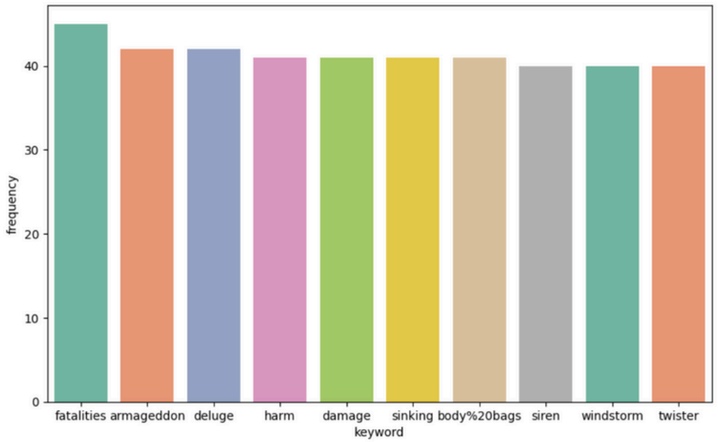
- Like
- Amp
- Get
- New
- One
- Body
- Time

Insight: Disaster tweets tend to contain more specific **event-related vocabulary**, while non-disaster tweets use more **general terms.**

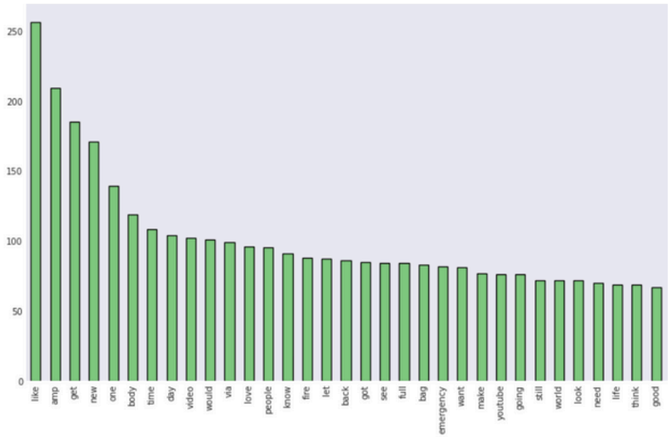
Word Cloud of Disastrous Keywords



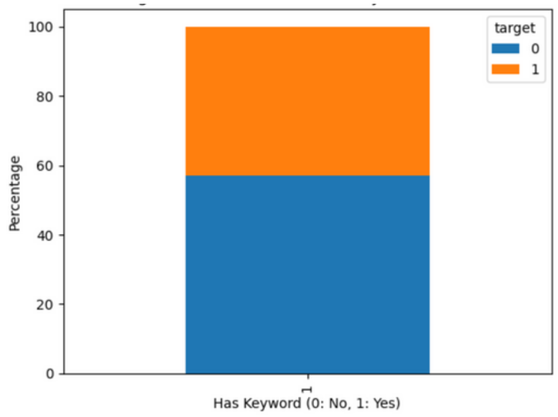
Most Frequently Used Disaster Tweets



Most Frequently Used in Non-Disaster Tweets



Target Distribution based on Keyword Presence



Stage #2: Exploratory Data Analysis (EDA)
Location Analysis

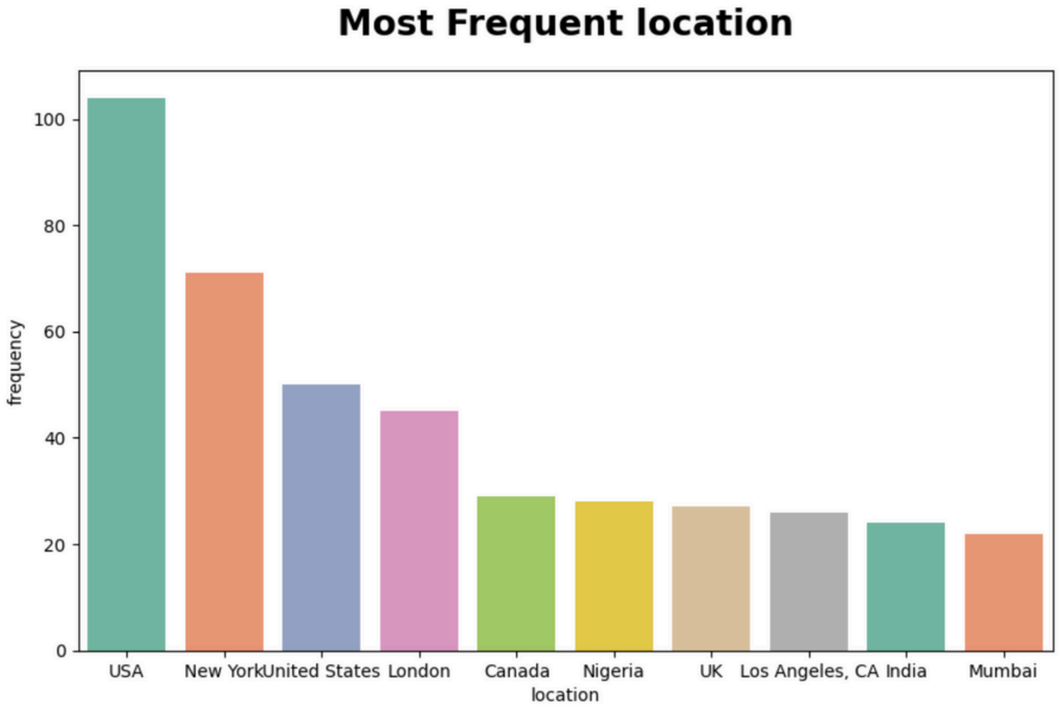
Top Locations

United States
- The United States is the **most frequently mentioned** location in the dataset, with over 100 mentions.
- New York is the **second most common location**, reflecting its status as a major metropolitan hub and a frequent topic of discussion in global events.

London
- London ranks third, highlighting its prominence as a global city and its **frequent appearance** in news and social media discussions.

Canada and Nigeria
- These countries also appear frequently, indicating their relevance in the dataset's context, possibly due to **specific events or disasters**.

Other Locations
- UK, Los Angeles (CA), India, and Mumbai are among the top locations, showcasing a mix of **global cities and regions**.



Stage #2: Exploratory Data Analysis (EDA)
Text Distribution Analysis

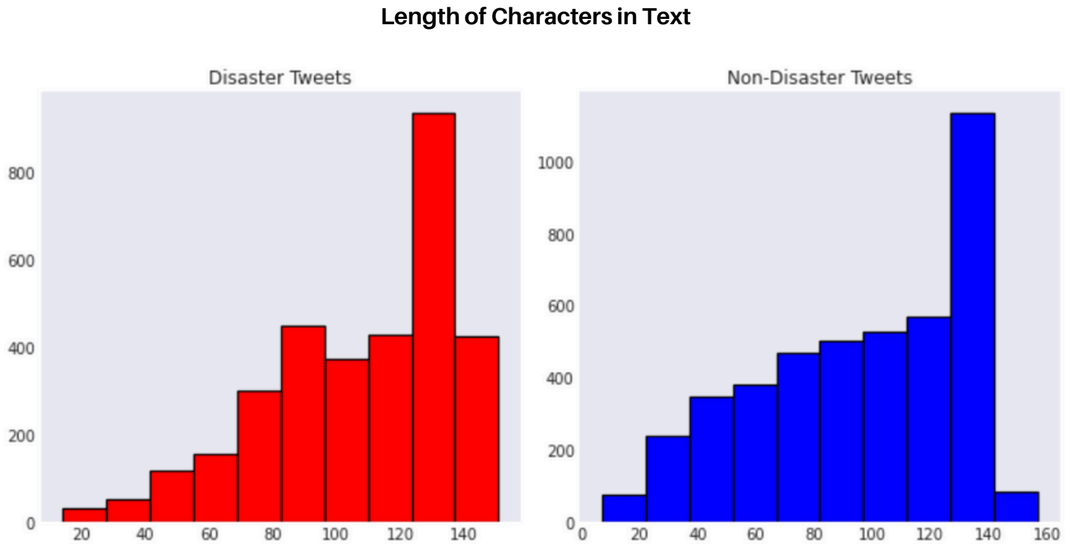
Distribution Overview
- The graph shows the **distribution of tweet lengths**, both in terms of words and characters.
- There are **two distinct distributions**: one for disaster-related tweets and one for non-disaster tweets.

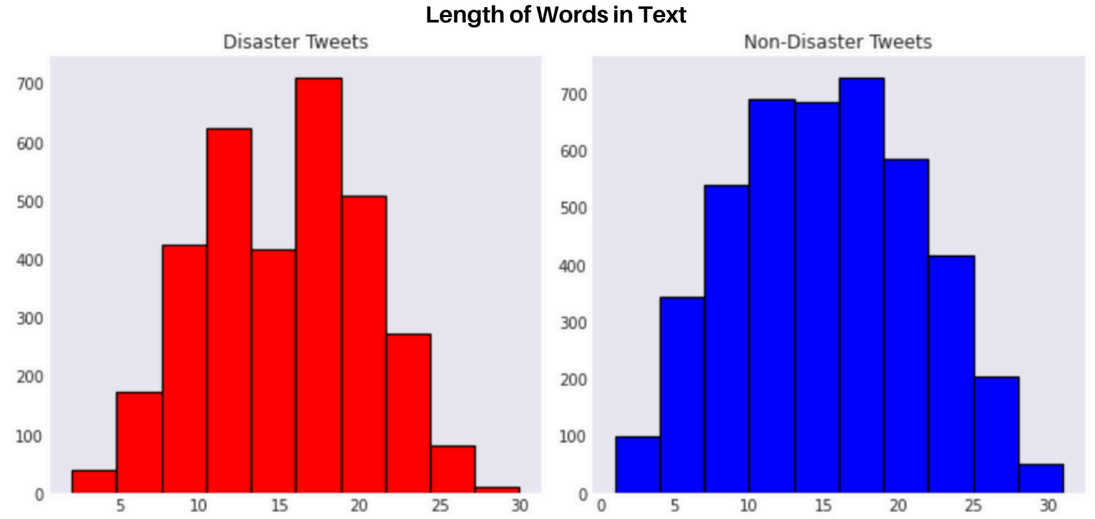
Word Count Distribution
- Disaster tweets tend to have a **slightly higher average word count** compared to non-disaster tweets.
- The peak of the disaster tweet distribution is **shifted slightly to the right** of the non-disaster tweet distribution.

Character Count Distribution
- Disaster tweets generally have **more characters** than non-disaster tweets.
- The character count distribution for disaster tweets is **wider**, indicating more variability in length.

Length Ranges
- Most tweets (both disaster and non-disaster) fall within a range of about **10-30 words**.
- Character counts typically range from about **50-200 characters** for both categories.

Outliers
- There are some outliers in both categories, with some tweets having **significantly higher** word and character counts.
- These outliers are more pronounced in the **disaster tweet category**.





STAGE #3-5: Model Architecture

Selecting the appropriate NLP

Feature	GRU	LSTM	BERT family
Type	RNN (Recurrent Neural Network)	RNN (Recurrent Neural Network)	Transformer-based model
Gates	2 gates (Update, Reset)	3 gates (Input, Forget, Output)	No gates (uses self-attention)
Memory	Short-term memory	Long-term memory	Contextual understanding
Training	Trained from scratch	Trained from scratch	Pretrained on large datasets
Input Processing	Word by word	Word by word sequentially	Entire tweet
Strong Points	Simple and fast	More accurate on longer sequences	Understanding context through self-attention Strong performance on informal and fragmented Text

Model Architecture for RNN

Parameter Selection and Model Tuning

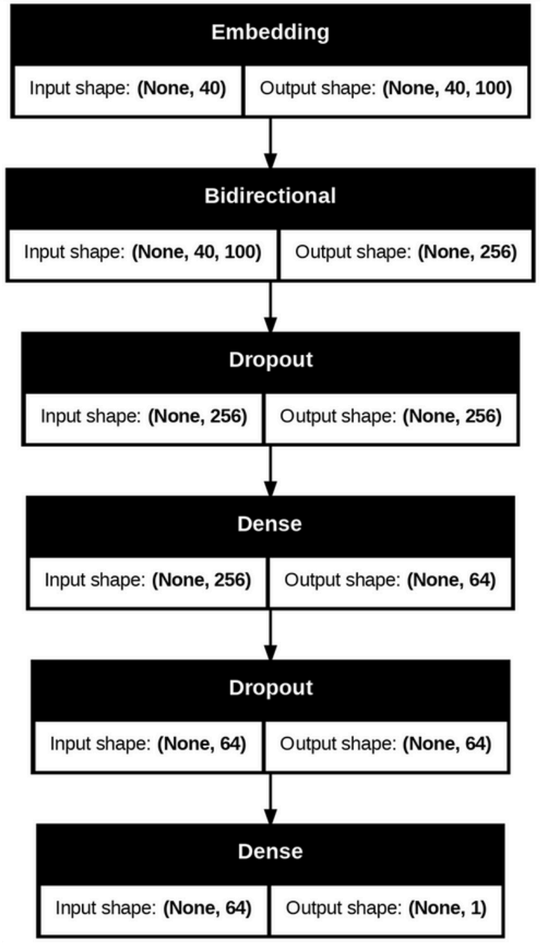
Parameters :

```
# Hyperparameters
vocab_size = 20000  # maximum number of words to keep in the vocabulary
max_len = 40        # maximum tweet length in tokens (adjust if needed)

tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)  # learns the top words from training data

# Convert text to integer sequences
train_sequences = tokenizer.texts_to_sequences(X_train)
val_sequences   = tokenizer.texts_to_sequences(X_val)

# Pad sequences to fixed length
train_padded = pad_sequences(train_sequences, maxlen=max_len, padding='post', truncating='post')
val_padded   = pad_sequences(val_sequences,   maxlen=max_len, padding='post', truncating='post')

print("Example original text:\n", X_train[0])
print("Tokenized + padded:", train_padded[0])
print("Shape of train_padded:", train_padded.shape)
```

```
Example original text:
 snowstorm south usa sassy city girl country hunk stranded in smoky mountain snowstorm aoms ibooklove bookboost
Tokenized + padded: [ 421  194   36 4267   83  489  876 4268 2659    4 4269 1131  421 4270
 4271 4272    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0]
Shape of train_padded: (6090, 40)
```

```
optimizer = Adam(learning_rate=1e-4)
model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# (B) Early stopping
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
```

Model Summary



Bidirectional Gated Recurrent Unit (GRU)

PROS:
- Fast processing

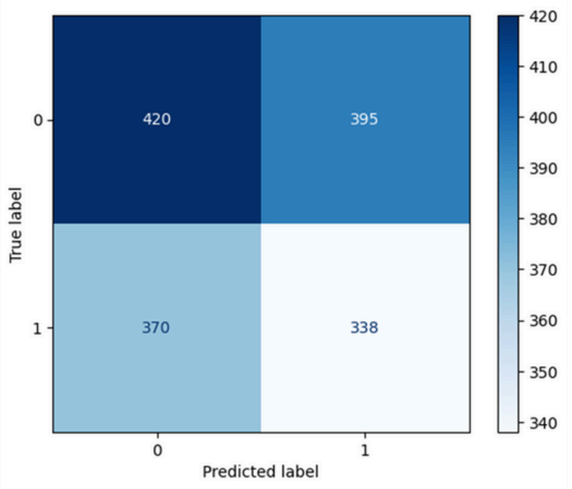
CONS:
- Very Low Accuracy with high FP and FN values
- Overfitting

Parameters

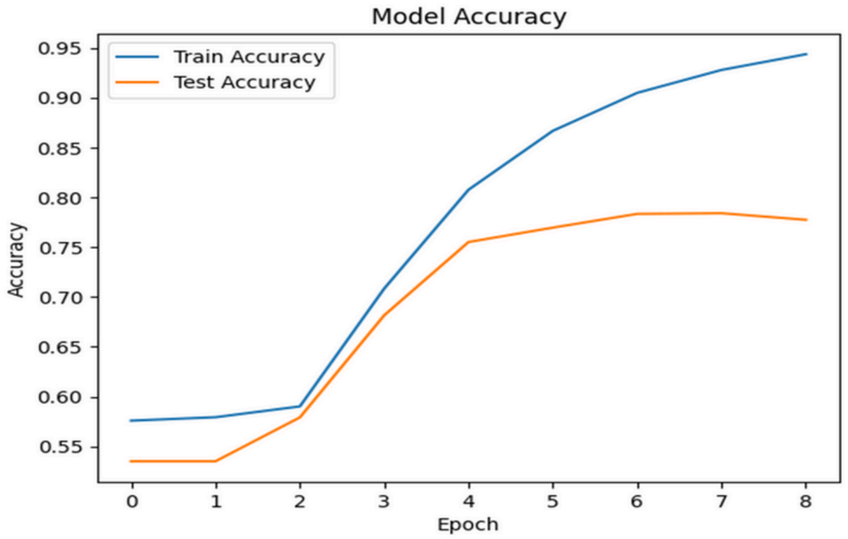
- Vocab Size = 20,000 (Max was 19701)
- Max Length: 40 (Max was 36)
- Embedding = 100
- Batch Size = 32
- Num Epochs = 10
- Early stopping: Patience = 2
- Dropout :0.3

Results

- ✓ True Positive Rate (Sensitivity): 0.4774
- ✓ False Positive Rate (1 - Specificity): 0.4847
- ✓ False Negative Rate (1 - Sensitivity): 0.5226
- ✓ True Negative Rate (Specificity): 0.5153



F1 Score: 0.4691



Overfitting: Train Accuracy > Test Accuracy

Bidirectional Long Short-Term Memory (LSTM)

PROS:
- Comparatively higher F1 score than GRU
- Lower number of FP

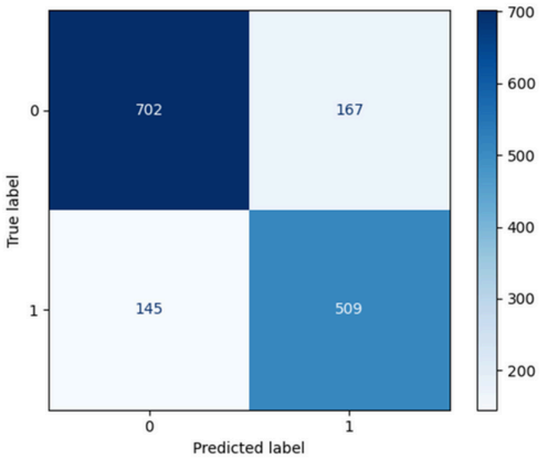
CONS:
- Accuracy is still low

Parameters

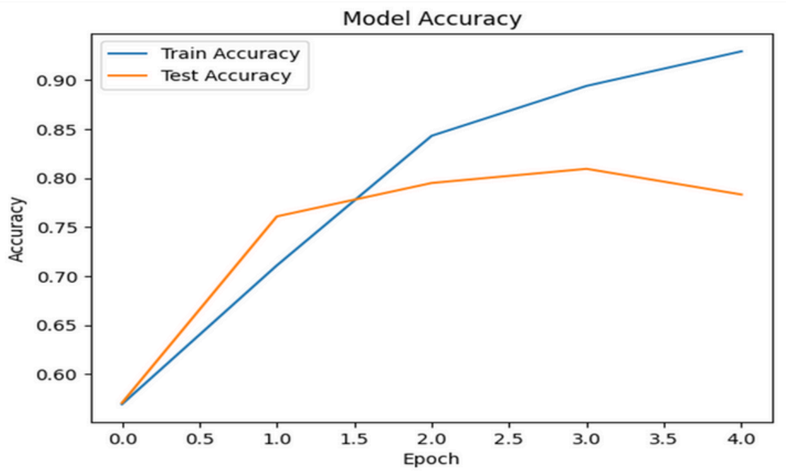
- Vocab Size = 20,000 (Max was 19701)
- Max_Length: 40 (Max was 36)
- Embedding = 100
- Batch Size = 32
- Num Epochs = 10
- Early stopping: Patience = 2
- Dropout :0.3

Results

- ✓ True Positive Rate (Sensitivity): 0.7187
- ✓ False Positive Rate (1 - Specificity): 0.1289
- ✓ False Negative Rate (1 - Sensitivity): 0.2813
- ✓ True Negative Rate (Specificity): 0.8711



F1 Score 0.7605



Overfitting: Train Accuracy > Test Accuracy

The BERT Family

Model #1: distilBERT

PROS:
- More optimized and 60% faster than the regular BERT model, making it ideal for resource-constrained environments and real-time applications.
- Efficient especially for text classification tasks where speed is critical content and position embeddings, improving dependency modelling.

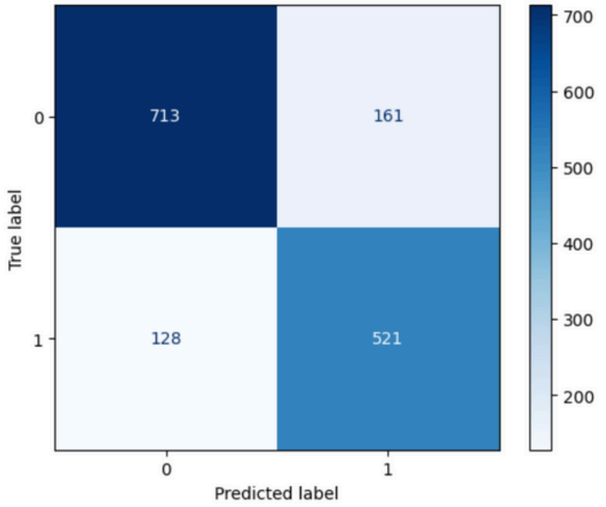
CONS:
- Slightly lower accuracy (retains ~97% of BERT's performance).

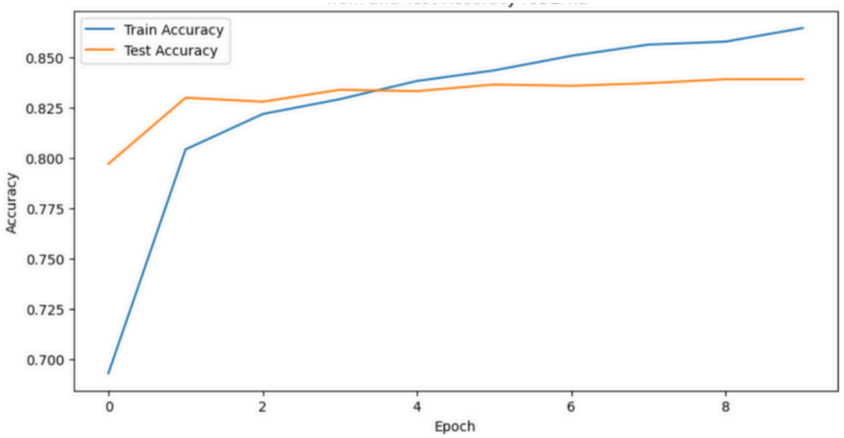
Parameters

- Fine Tuning = nltk library
- Sequence_Length: 160
- Batch Size = 32
- Num_Epochs = 10
- Optimizer = keras.optimizers.Adam(1e-6)
- Warmup steps = 0.1 x total steps

Results

- TPR (Sensitivity): 0.802773
- FPR (1-Specificity): 0.184210
- FNR (1-Sensitivity): 0.197227
- TNR (Specificity): 0.815789





F1 - Score: 0.8029

The BERT Family

Model #2: deBERTa

PROS:
- Introduces disentangled attention - separating content and position embeddings improving contextual understanding and dependency modeling.

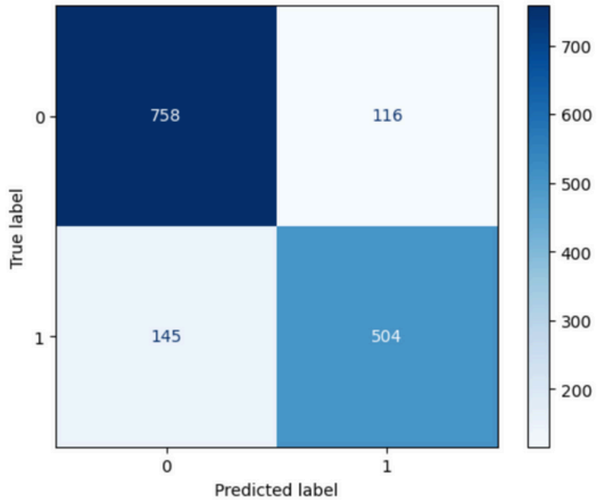
CONS:
- deBERTa has a higher computational cost compared to models like RoBERTa, requiring more GPU memory.
- Took FOREVER to execute :/

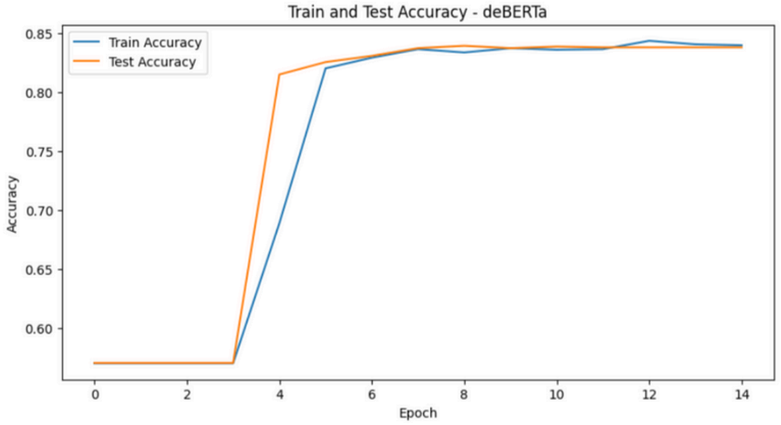
Parameters

- Max_Length: 150
- Batch Size = 8
- Num_Epochs = 14
- Initial Learning Rate = 2 x 10^-2
- Warmup steps = 0.1 x total steps
- Weight Decay Rate = 0.2

Results

- TPR (Sensitivity): 0.786579
- FPR (1-Specificity): 0.132723
- FNR (1-Sensitivity): 0.223421
- TNR (Specificity): 0.867277





F1 - Score: 0.8143

The BERT Family

Model #3: roBERTa

PROS:
- Outperforms BERT by using a larger dataset and advanced training techniques (e.g., dynamic masking).
- Excellent for text classification, summarization, and question answering.

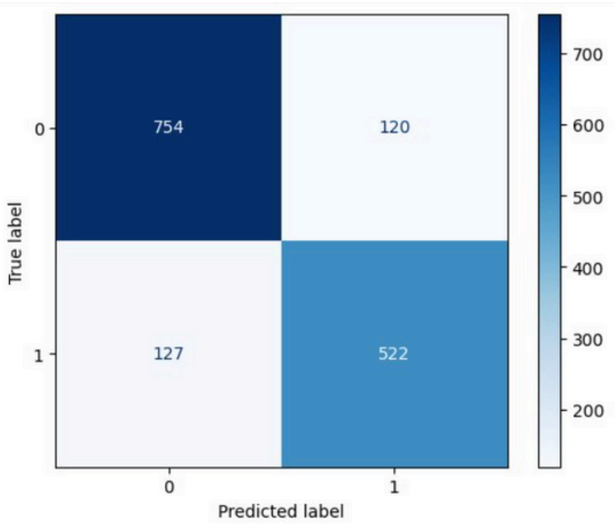
CONS:
- Larger model size and higher computational cost compared to BERT.
- No next sentence prediction (NSP) task, limiting its use in some contexts.

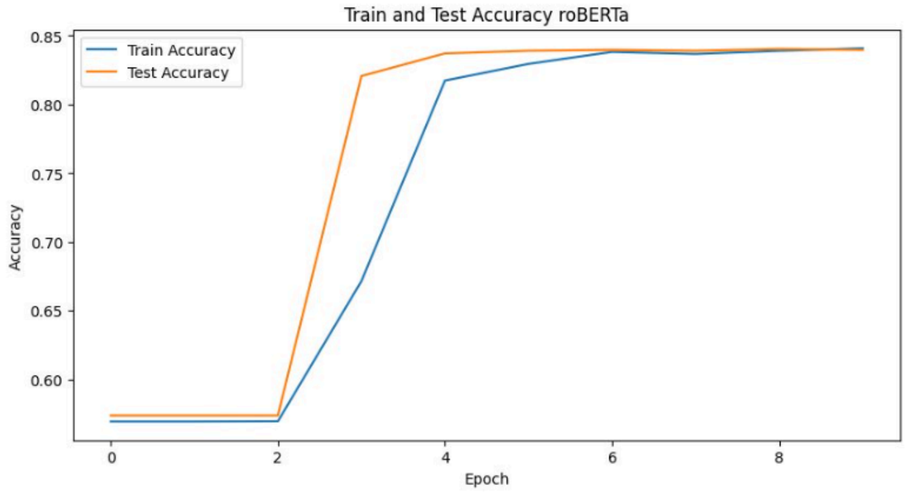
Parameters

- Max_Length: 150
- Batch Size = 36
- Num_Epochs = 10
- Initial Learning Rate = 5 x 10^-6
- Warmup steps = 0.1 x total steps
- Weight Decay Rate = 0.01

Results

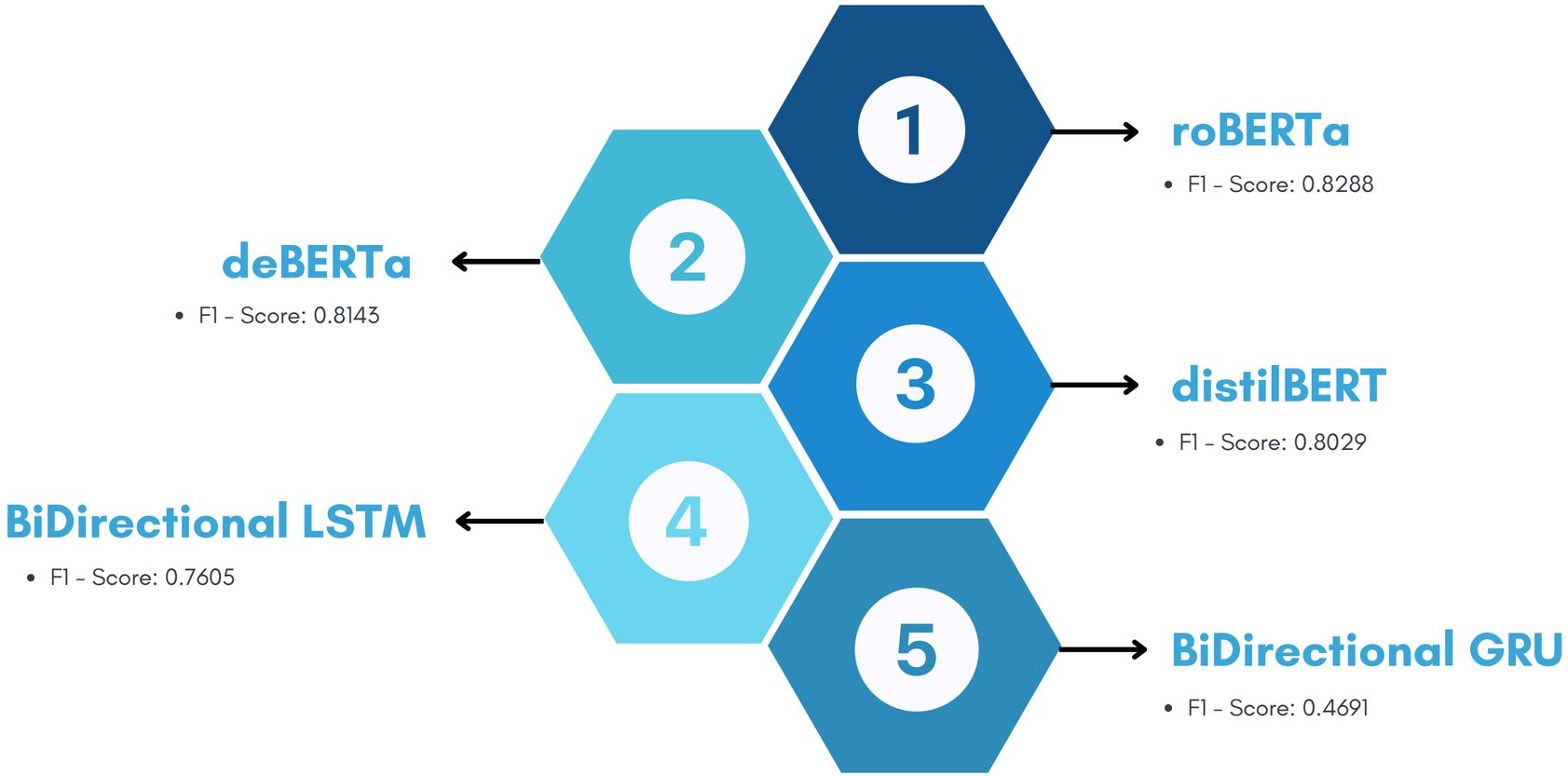
- TPR (Sensitivity): 0.804314
- FPR (1-Specificity): 0.137299
- FNR (1-Sensitivity): 0.195685
- TNR (Specificity): 0.862700



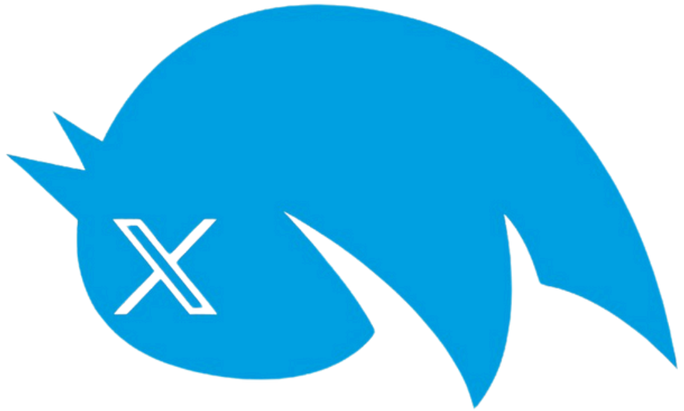


F1 - Score: 0.8288

Conclusion


1
2
3
4
5
roBERTa
F1 - Score: 0.8288
deBERTa
F1 - Score: 0.8143
distilBERT
F1 - Score: 0.8029
BiDirectional LSTM
F1 - Score: 0.7605
BiDirectional GRU
F1 - Score: 0.4691

THANK YOU



The BERT Family

Opting for a Transformer Approach

Leveraging Pretrained Models for Adaptability

- With exposure to extensive datasets, these models transfer knowledge effectively to new tasks.

Strong Performance on Informal and Fragmented Text

- They efficiently interpret slang, broken sentences, and unstructured language.

Understanding Context Through Self-Attention

- Transformers analyze words in relation to others in a sentence, ensuring a deeper grasp of meaning.

Parallel Processing for Speed and Scalability

- As RNNs read text step by step, Transformers process entire inputs at once for greater efficiency.

Advanced Tokenization for Complex Words

- Using Byte-Pair Encoding (BPE), they break down unfamiliar words, misspellings, and social media terms for better comprehension.