CAP 6640 Group 5

Binary Text Classification with Disaster Tweets

Nikhil Sreedhar, Tsogjavkhlan Odbayar, Eduardo Bourget, Elmaddin Azizli

https://github.com/nikhilmsreedhar/CAP-6640-Projects

Problem overview

Problem statement

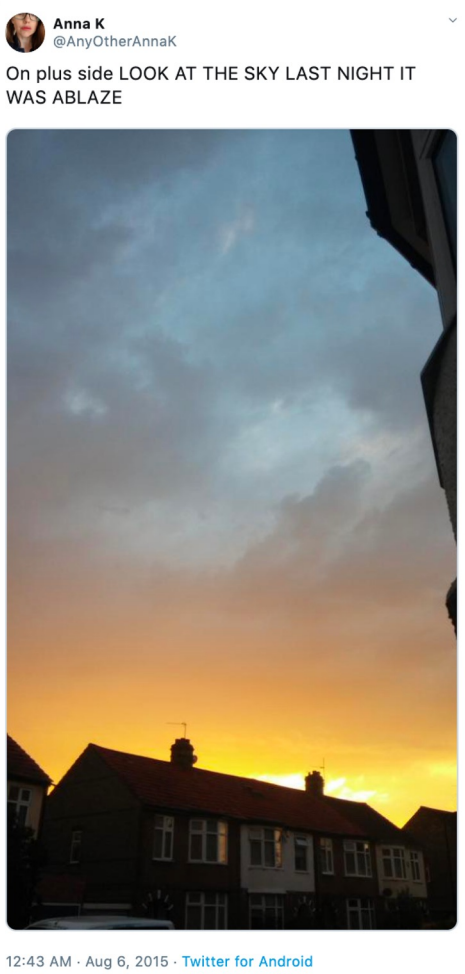
To construct a machine learning model that is capable of accurately performing binary classification on a dataset of disaster/non-disaster tweets to predict which tweets were related to real disasters and which tweets were not related to disasters. This entails data preprocessing, model selection, model training, and model prediction to perform binary classification on the testing dataset to predict whether a tweet was a disaster related tweet or a non-disaster related tweet.

Problem relevance and importance

This problem is relevant because twitter is a communication platform that is used in times of disaster or emergency. In today's world, most people have smartphones, and by extension, may also have twitter installed. This establishes twitter as a relevant communicative tool in our modern society. Therefore, disaster and emergency situations can be effectively communicated through this medium. This has propelled many agencies to observe twitter to capture any disaster related tweets and communications. However, to a machine, it is not always obvious whether a tweet is related to a disaster or not and doing so manually would be consume lots of time. Therefore, building a model capable of accurately predicting disaster tweets offers a systematic approach to observe these disaster tweets for organizations and agencies searching for a solution to this problem.

Example: The author explicitly uses the word "ABLAZE" but means it metaphorically. This is clear to a human right away, especially with the visual aid. But it's less clear to a machine.

Through this project, we evaluated results of different models for this binary sequence classification task.



Data overview

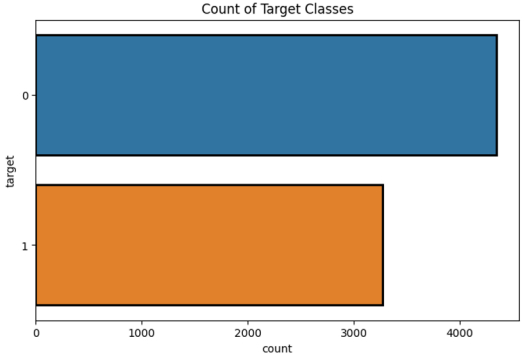
The training dataset contains 7503 values:

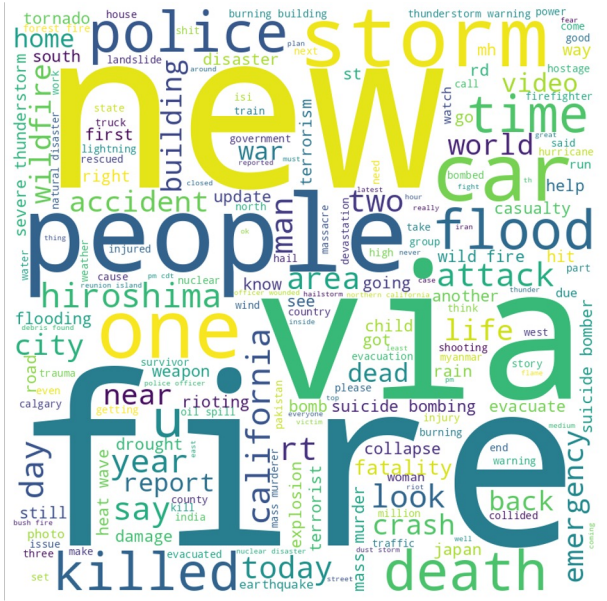
4342 being non-disastrous, 3271 being disastrous

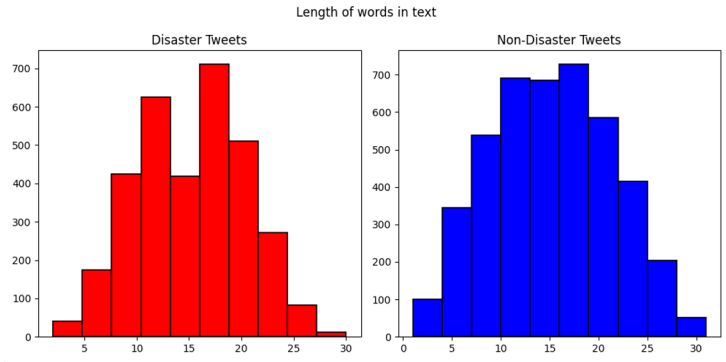
Each tweet ranges from 1-31 words.
Each word ranging from 2-20 characters long.

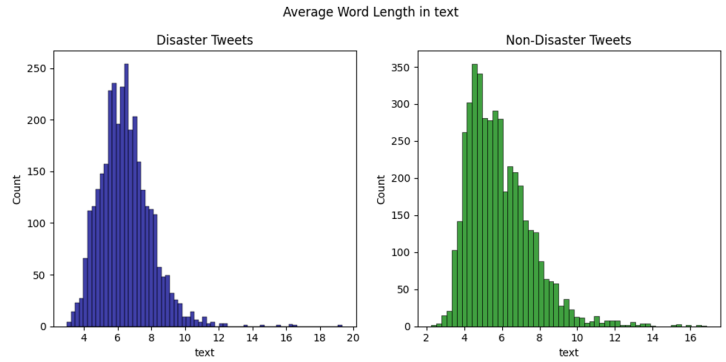
The classes are represented as 0 or 1. 0 being a non-disaster tweet and 1 being a disaster tweet

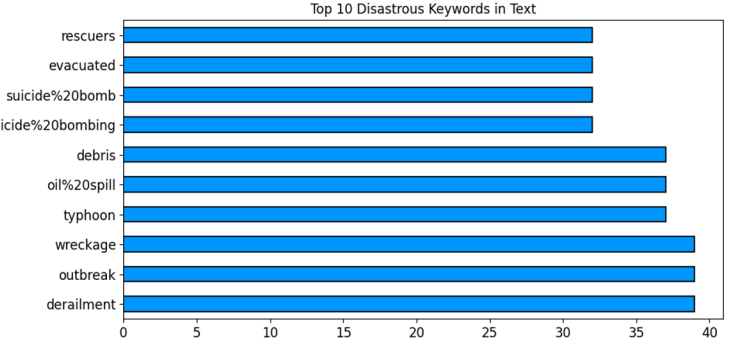
The dataset size is 1.43 MB. The test dataset does not come with class labels. The dataset was obtained from the NLP with Disaster tweets Kaggle competition specified in the following link: https://www.kaggle.com/competitions/nlp-getting-started/data











Data preprocessing and Feature Engineering

- Word level cleaning and preprocessing:
  - Lowercasing
  - De-abbreviating (ASAP => as soon as possible)
  - De-contracting (aren't => are not)
  - Lemmatizing
  - Tokenization
- Sentence level cleaning and preprocessing:
  - Removing mentions, urls, emojis, punctuations, digits
  - Removing stopwords
- Dataset level preprocessing:
  - Dropping duplicate data samples with ambiguous labels
  - Dropping/Filling null values
- Dealing with class imbalances
  - Performed data augmentation using random oversampling approach on training dataset of the bag of words model experiments.
- Feature Engineering
  - The training dataset contained a 'keyword' feature. This specific feature engineering approach aimed to combine the 'keyword' feature with the 'text' feature in the training, validation, and testing datasets yielding a combined text feature. This was done on the Bag of Words and SVM models to improve performance.
  - This aims to convey more explicit information about the tweets content by emphasizing the representation and frequency information regarding the keyword, leading to an improvement in the Bag of words and SVM models' ability to handle ambiguity and an improvement in their classification metrics.

SVM - Support Vector Machine

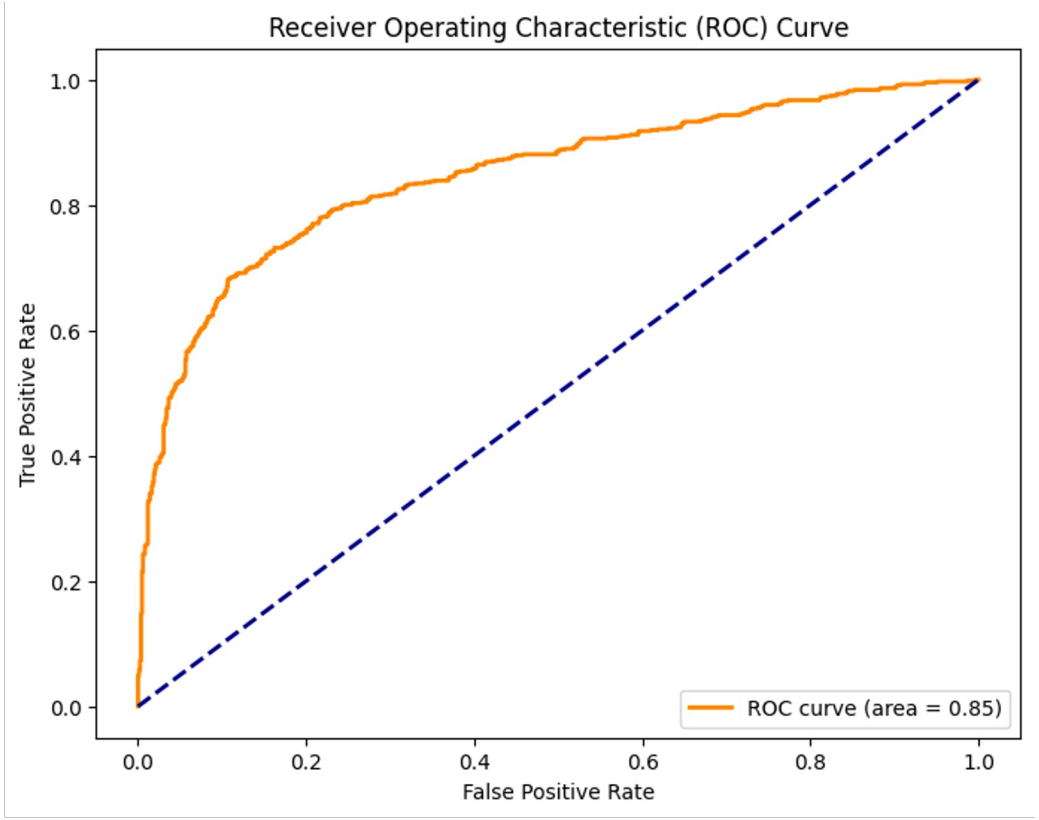
In our specific TF-IDF model, we chose to implement a Support Vector Machine (SVM) for its effectiveness in handling high-dimensional data and its capability to find optimal decision boundaries in classification tasks. The SVM used a linear kernel, meaning it aimed to find the best hyperplane to separate the classes in the feature space.

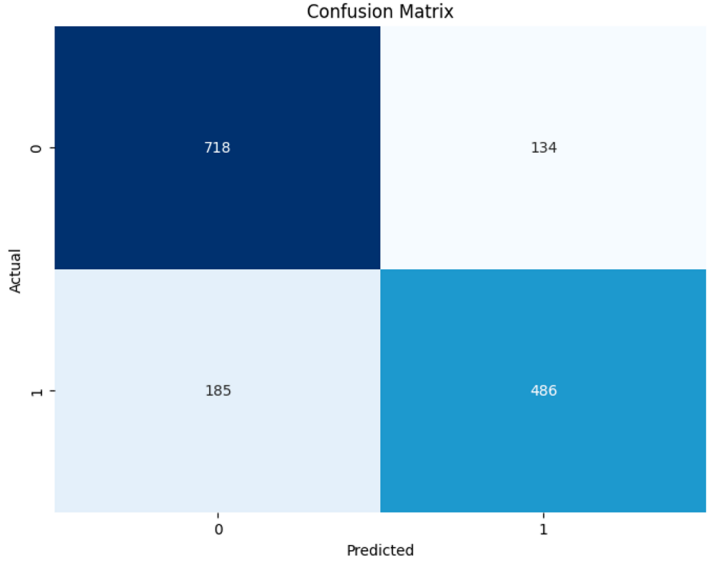
The SVM model was configured with the following hyperparameters:

- C=1 (regularization parameter)
- kernel='linear' (linear kernel for a linear decision boundary)
- decision_function_shape='ovo' (one-vs-one strategy for multi-class classification)

Our TF-IDF model was constructed as a pipeline, integrating a TF-IDF vectorizer and an SVM classifier. The TF-IDF vectorizer transformed the raw text data into a numerical format by considering the importance of each word in the context of the entire dataset. This transformed data was then fed into the SVM classifier, which learned to distinguish between different classes based on the vectorized representation of the text.

SVM - Support Vector Machine results





	precision	recall	f1-score	support
0	0.80	0.84	0.82	852
1	0.78	0.72	0.75	671
accuracy			0.79	1523
macro avg	0.79	0.78	0.79	1523
weighted avg	0.79	0.79	0.79	1523

BOW - Bag of Words

The bag of words model is a natural language processing and machine learning model that processes text and notes the frequency of certain tokens (words) in the text. This is especially useful in tasks like classification. Therefore, we decided to utilize this model.

Our specific Bag of words model consisted of a **Logistic Regression** model configured with the following hyperparameters:

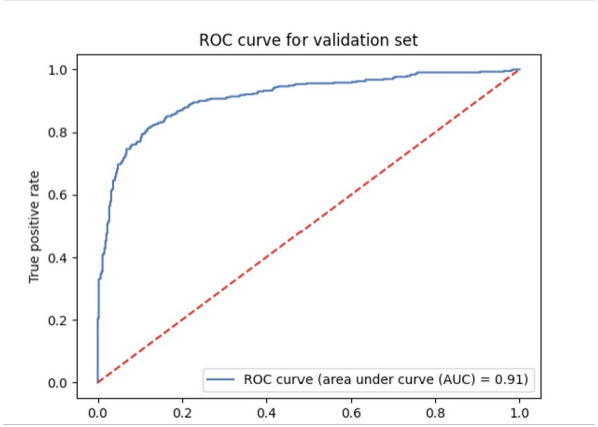
- solver='liblinear' (Increased performance slightly. Sklearn docs for logistic regression state this solver is beneficial for smaller datasets such as ours)
- class_weight='balanced' (Increased performance minimally. Want to balance the classes to be as unbiased as possible since there was a class imbalance)
- dual=True (No change in performance. To be used with l2 penalty (default penalty) and liblinear solver)
- C=0.85 (Increased performance moderately. inverse of regularization strength. Regularization benefits models to avoid memorizing training data and generalize better. Therefore we chose to use this)

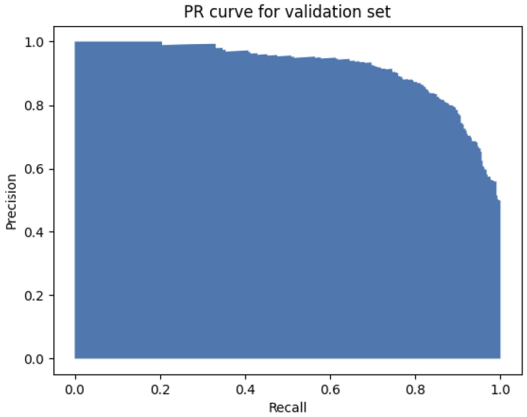
Our bag of words model is constructed as a pipeline consisting of a count-vectorizer to generate the frequency counts of the tokens in a text and a logistic regression model to feed the vectorized text as input to the logistic regression model to produce an output of which class the tweet belongs to.

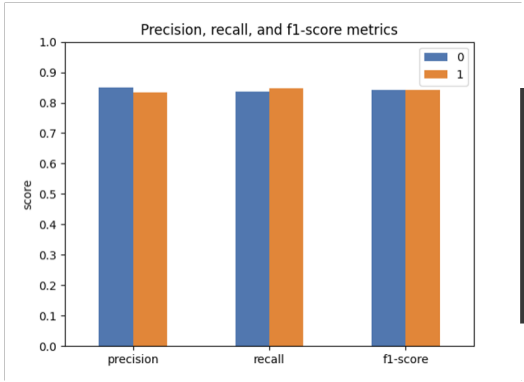
One key drawback of the bag of words model is its inability to capture context. This may inhibit from achieving higher f1-scores. However, after performing data augmentation by way of randomly oversampling the minority class, it appears that the bag of words classifies both disaster tweets and non disaster tweets equally as accurately for both classes as shown in the classification results on the next slide.

BOW - Bag of Words results

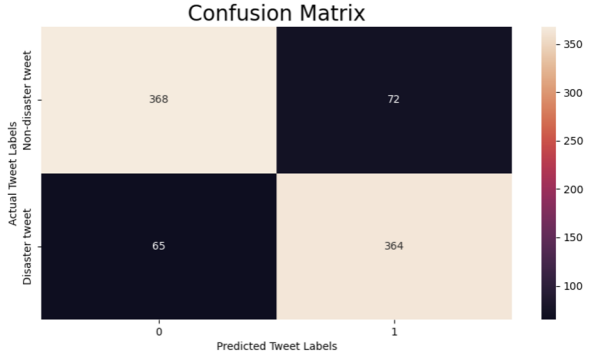
The results and appropriate metrics for the bag of words model is shown below. The max f1-score on the bag of words model was 0.841 on the validation set and 0.798 on the test set according to the Kaggle competition submission score.







	precision	recall	f1-score	support
0	0.85	0.84	0.84	440
1	0.83	0.85	0.84	429
accuracy			0.84	869
macro avg	0.84	0.84	0.84	869
weighted avg	0.84	0.84	0.84	869



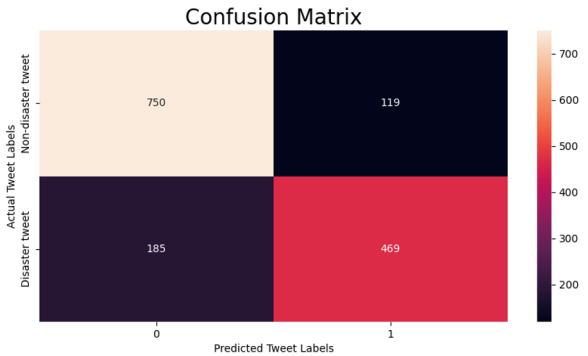
LSTM variations

- LSTMs are fit for sequence classification tasks because they can utilize sequential structure of our data in this task by learning long-term dependencies.
- For this model, we are using keras tokenizer to transform our text tweets into sequences of embedding followed by padding the tokenized sequence with zeros for word embeddings.
- Methods we tried out with this model are:
  - Added dropouts and recurrent_dropouts (major f1-score increase = was overfitting before)
  - Added Dense layers after LSTM (minor f1-score increase)
  - Added CNN with MaxPooling1D layers (minor f1-score increase)
  - Used BiLSTM rather than LSTM (no-change)
  - Stacked LSTM layers with first layer feeding sequences into the second (no-change)
  - Tuning hyperparameters (minor f1-score increase)

  Major difference ~ 0.1, Minor difference ~ 0.01

LSTM variation results

Here are the results for network configuration shown below. F1-score of 0.79 was achieved on kaggle submission.



embedding_input	input:	[(None, 100)]
InputLayer	output:	[(None, 100)]

↓

embedding	input:	(None, 100)
Embedding	output:	(None, 100, 100)

↓

spatial_dropout1d	input:	(None, 100, 100)
SpatialDropout1D	output:	(None, 100, 100)

↓

conv1d	input:	(None, 100, 100)
Conv1D	output:	(None, 100, 64)

↓

max_pooling1d	input:	(None, 100, 64)
MaxPooling1D	output:	(None, 50, 64)

↓

bidirectional(lstm)	input:	(None, 50, 64)
Bidirectional(LSTM)	output:	(None, 128)

↓

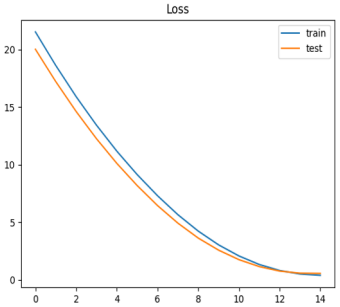
dense	input:	(None, 128)
Dense	output:	(None, 32)

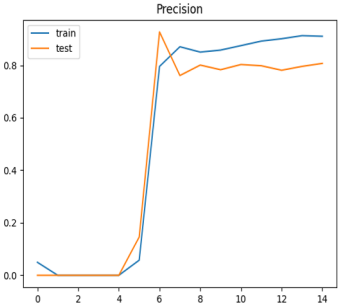
↓

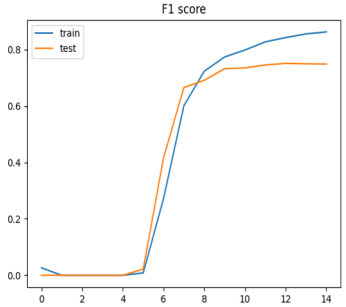
dropout	input:	(None, 32)
Dropout	output:	(None, 32)

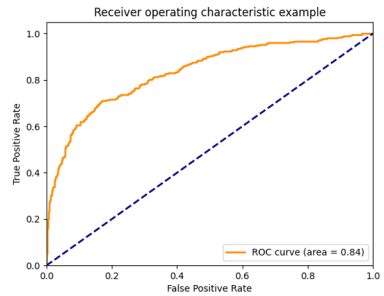
↓

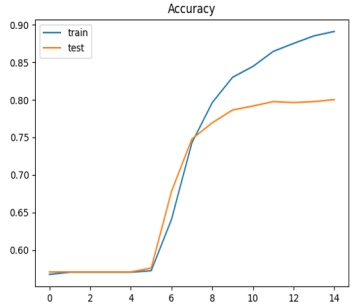
dense_1	input:	(None, 32)
Dense	output:	(None, 1)

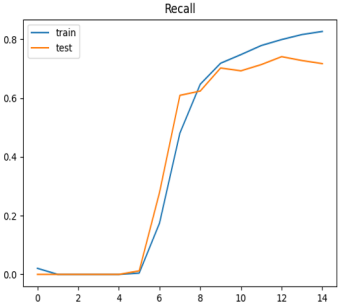


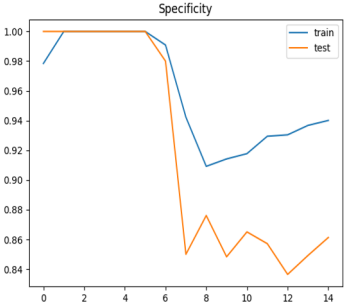


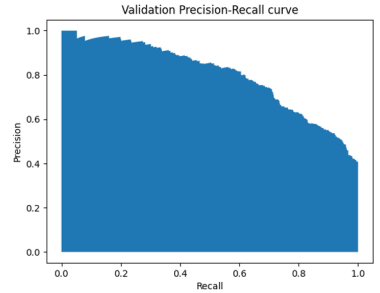












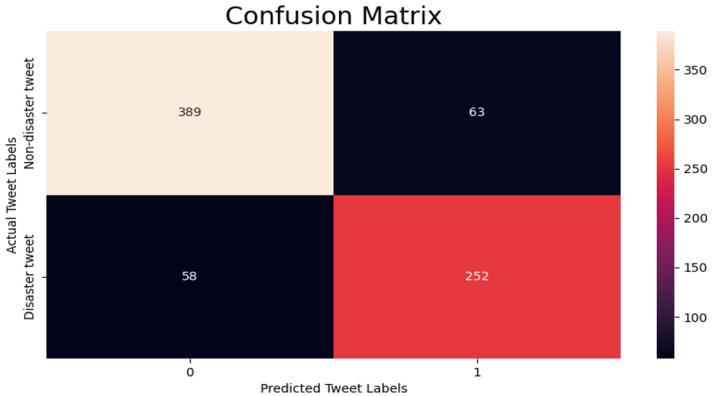
BERT model (DistilBERT)

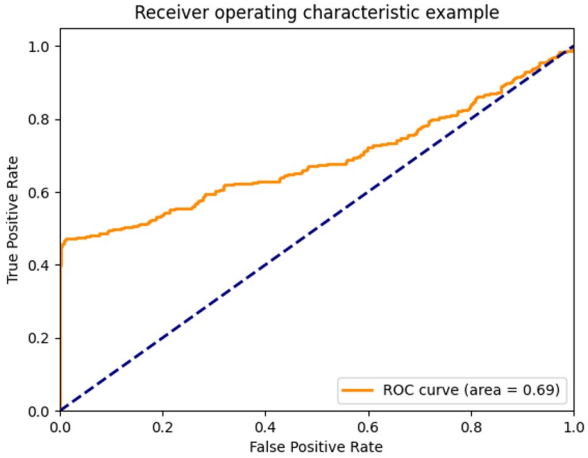
- For this model we have used DistilBERT: A simplified version of the BERT (Bidirectional Encoder Representations from Transformers) model.
- We decided to use this specific model for our project, because is designed to be smaller, faster, and lighter than BERT, making it more efficient for deployment without significantly compromising performance.
- With fewer parameters and faster processing, it maintains the majority of BERT's language understanding capabilities.
    - DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased') was used for tokenization
    - DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased') was fine tuned.

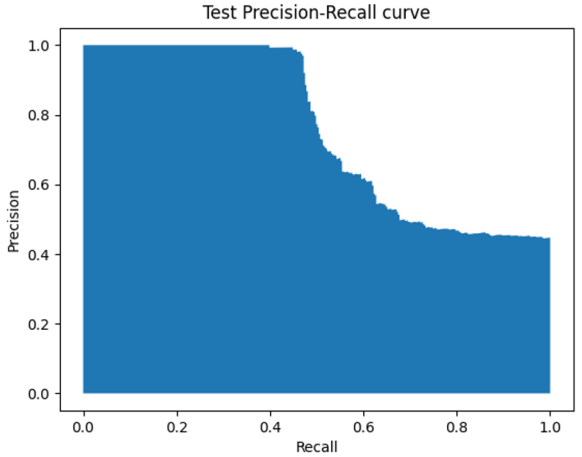
DistilBERT Results

Kaggle submission F1-score of 0.8201

	precision	recall	f1-score	support
0	0.87	0.86	0.87	452
1	0.80	0.81	0.81	310
accuracy			0.84	762
macro avg	0.84	0.84	0.84	762
weighted avg	0.84	0.84	0.84	762







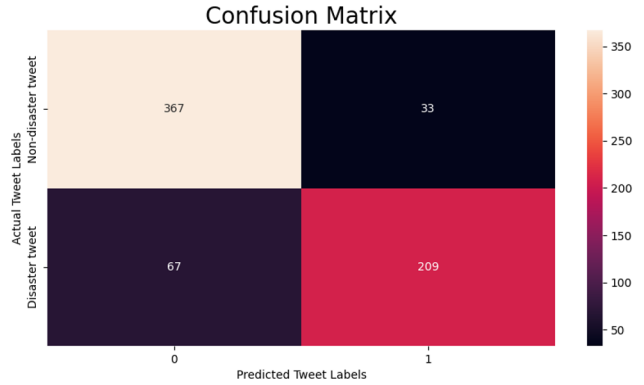
RoBERTa - Robustly optimized BERT

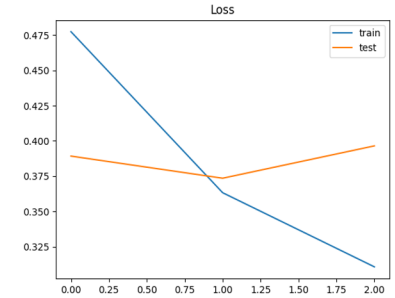
RoBERTa is trained on large amount of data and is an optimized version of BERT with better performance.

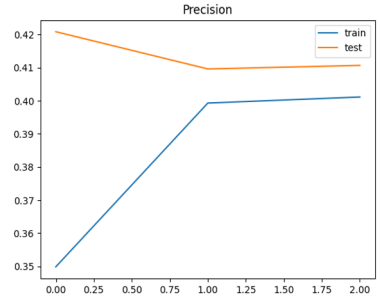
- We use AutoTokenizer class from transformers library to tokenize our texts
- TFAutoModelForSequenceClassification.from_pretrained(‘roberta-base’) model was fine-tuned with our labeled dataset.

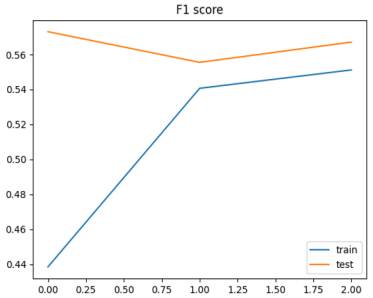
RoBERTa results

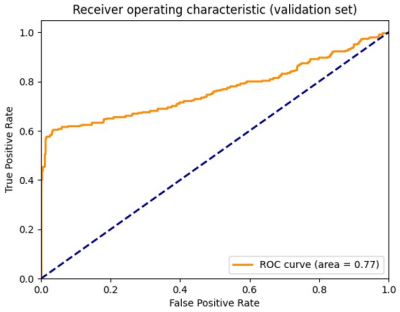
Our best test result for RoBERTa model was f1-score of 0.839 on test set.

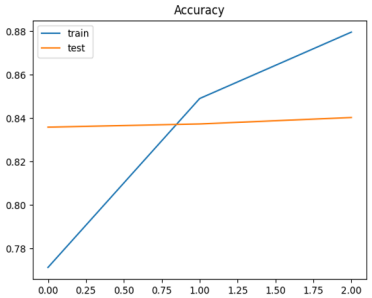


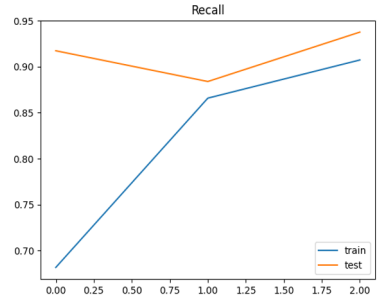


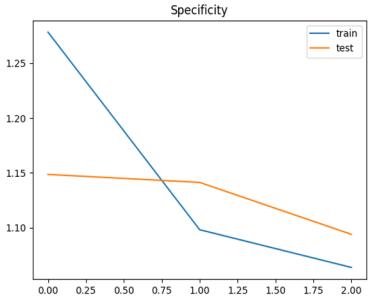


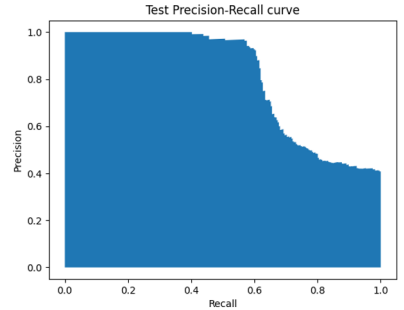












Results and Performance comparison

Generally, on the validation and testing datasets, it appears that the BERT model variants perform better than the LSTM, Bag of Words, and SVM models. Specifically, RoBERTa performed the best achieving an F1 score of 0.839 on the test set. The DistilBERT, LSTM, Bag of Words, and SVM models performed decently overall achieving an F1 score of 0.82, 0.79, 0.798, and 0.75, respectively.

Below are some inference results on example text prompts with RoBERTa model.

	Text	Label	Confidence	Status
0	A huge tornado just ripped off the neighbors roof!!!	Disaster	0.89597297	TP
1	Sun in Florida around August feels scorchingly hot, I'm burning alive	Non-Disaster	0.515149	TN
2	A truck containing 15 tonns of gas exploded in the middle of the city, costing lives of 3 firefighters	Disaster	0.9236359	TP
3	It has been raining heavily for the last 5 days	Disaster	0.6938958	FP
4	A 7.6 magnitude earthquake struck Japan on Jan. 1 around 2:10 a.m. EST (0710 GMT, or 4:10 p.m. local time in Japan)	Disaster	0.9261727	TP
5	Have a great weekends	Non-Disaster	0.81319886	TN

Conclusion

One key lesson our team learned as we approached the end of our project development phase was that dataset size and dataset quality are just as important as tuning a model to achieve higher performance. Model hyperparameter tuning can lead to some performance gains. However, effective preprocessing, text cleaning, feature engineering, and ensuring a dataset is large enough for a model to accurately train on is where a large bulk of performance gains can come from.

Overall, it appears that the BERT models, especially RoBERTa performed the best due to their ability to capture not only local but global context bidirectionally and capture long range dependencies of the text as well. The Bag of Words and SVM models lack this capability and that hindered their performance. Therefore, another key learning from these experiments is that a model's ability to understand context is crucial for it to perform well classifying tweets into disaster/non-disaster categories.

Our team enjoyed this project as it was a challenge to effectively preprocess the dataset specifically to each model and train multiple models to perform well on this classification task given the dataset's size and quality. Through many attempts at experimentation with each model, we learned volumes about proper data cleaning and preprocessing techniques respective to each model along with the strengths and weaknesses behind each model for the task of predicting disaster and non-disaster tweets.