Classifying Disaster Tweets

Background

Social media has become an important communication channel in times of emergency. Twitter is one of the popular social media platforms. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Agencies like Disaster Relief Organizations and Media Outlets are interested in programmatically monitoring Twitter to learn about disasters in real-time.

Problem Identification

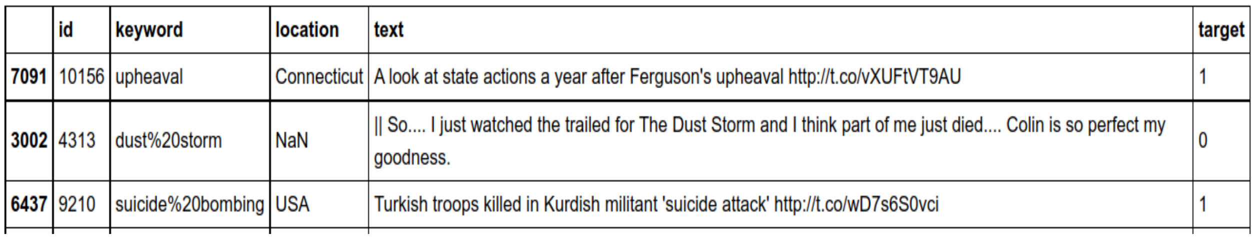
Tweets are streaming rapidly at scale. It is not feasible to have humans monitoring this flood of tweets and filter the ones that announce disaster. Thus, there is a need for automation using a Natural Language Processing (NLP) model. However, classifying tweets as disaster tweets and nondisaster tweets is a challenging NLP task. It's not always clear whether a person's words are actually announcing a disaster.

About the Data

The data set contains Train and Test sets and is taken from Kaggle. Source: https://www.kaggle.com/c/nlp-getting-started/data

A. Train Data Set

The train set has 7613 instances, 4 features, and a binary target variable (1 indicates disaster tweet and 0 for nondisaster tweets).



	id	keyword	location	text	target
7091	10156	upheaval	Connecticut	A look at state actions a year after Ferguson's upheaval http://t.co/vXUFtVT9AU	1
3002	4313	dust%20storm	NaN	|| So.... I just watched the trailed for The Dust Storm and I think part of me just died.... Colin is so perfect my goodness.	0
6437	9210	suicide%20bombing	USA	Turkish troops killed in Kurdish militant 'suicide attack' http://t.co/wD7s6S0vci	1

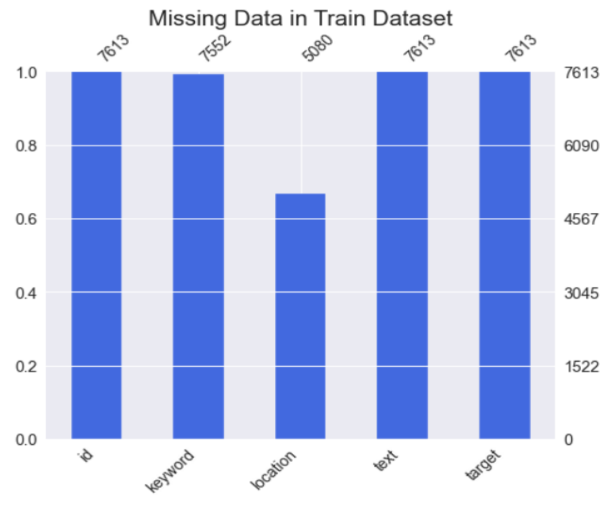
As shown in the table above, the features in the train data set are:

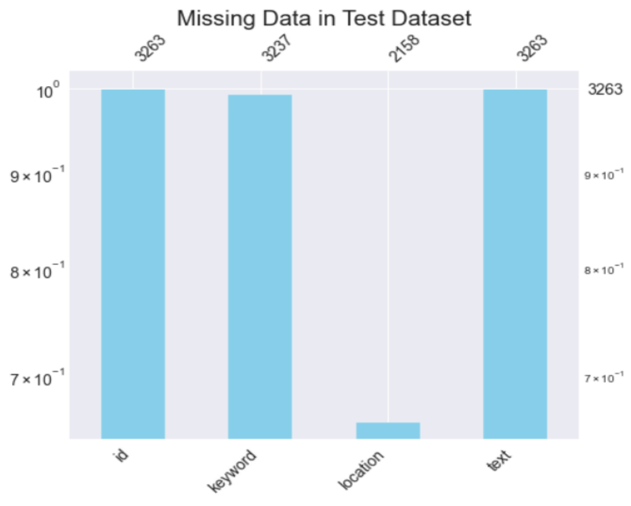
1. Id: a tweet identifier (number).
2. keyword: a keyword represents each tweet (text).
3. location: the location of the tweet (text).
4. text: the tweet text (text).

5. target: a binary target variable (1 indicates disaster tweet and 0 for nondisaster tweets).

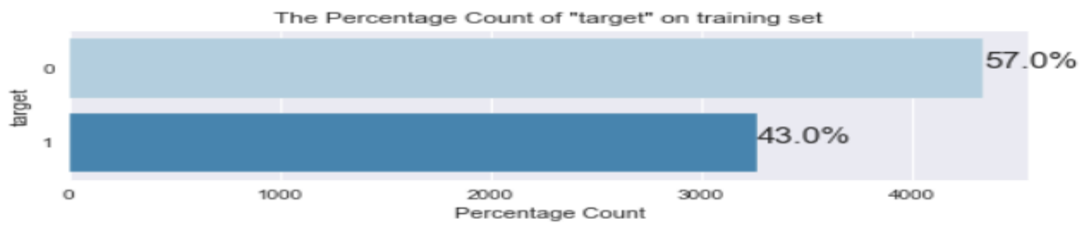
B. Test Data Set

The Test Data Set has 3265 samples, the same features as the train data set, but there is no target variable. The target variable needs to be predicted as required by the Kaggle competition.
As shown below, there is missing data in both sets in location and keyword features.





The original distribution of the target variable is shown below.



The Analysis Method

The problem is a binary classification problem. There is a need for a model that classifies whether the tweet announces disaster or not.
The business objective is to detect the disaster tweets (tweets that announce disaster) in real-time. The classifier will give 1 for disaster tweets and 0 for nondisaster ones. The data is chosen to represent both classes with reduced impedance. However, real-life tweets are mostly nondisaster, disasters do not happen often, real-life tweets are unbalanced. Thus, the focus should be more on

the positive class, disaster tweets, which are the minority class in reality. The agency would accommodate false positives for the sake of not missing True positive tweets.

The project went through all of the six steps of the data science project shown below.

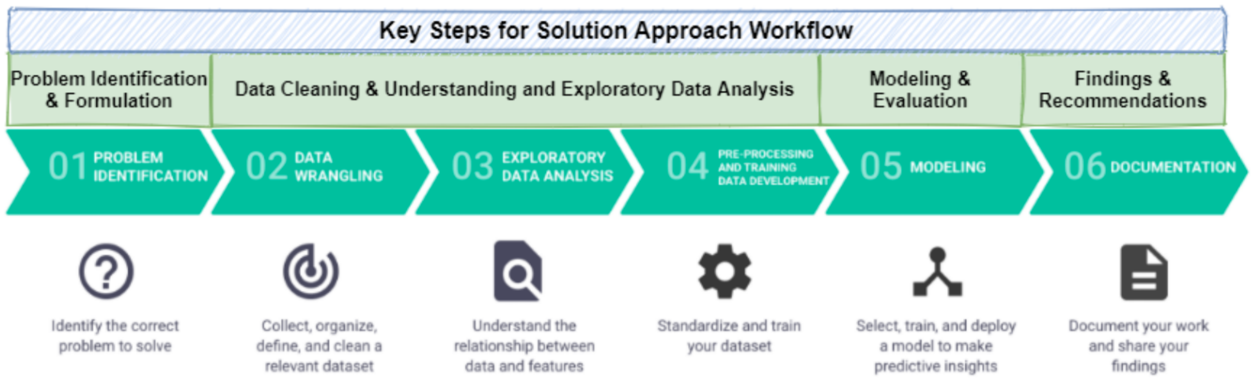

Key Steps for Solution Approach Workflow
Problem Identification & Formulation
Data Cleaning & Understanding and Exploratory Data Analysis
Modeling & Evaluation
Findings & Recommendations
01 PROBLEM IDENTIFICATION
02 DATA WRANGLING
03 EXPLORATORY DATA ANALYSIS
04 PRE-PROCESSING AND TRAINING DATA DEVELOPMENT
05 MODELING
06 DOCUMENTATION
Identify the correct problem to solve
Collect, organize, define, and clean a relevant dataset
Understand the relationship between data and features
Standardize and train your dataset
Select, train, and deploy a model to make predictive insights
Document your work and share your findings

Image source: Springboard.com

The tired modeling approaches:

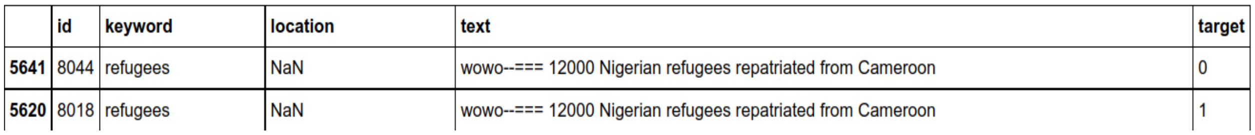
Two approaches are used to represent the text to be input to the machine learning models, TERT transformers (encoding - embeddings). Various models are tested.

a. Term Frequency-Inverse Document Frequency (TF-IDF)
    i. Multinomial Naive Bayes using GridSearch optimization
    ii. Logistic Regression using GridSearch optimization
    iii. XGboost with Bayesian Hyperparameter Optimization
    iv. Deep Learning:
        1. Simple Neural Network
        2. Recurrent Neural Network (LSTM)
        3. Convolutional Neural Network (CNN)
b. Transfer Learning / Deep Learning
    i. Bidirectional Encoder Representations from Transformers (BERT):
        1. Distilled BERT uncased
        2. BERT base uncased

Data Wrangling

Below are the major issues encountered in the data:

1. Missing data in location and keywords.
2. There are 110 text duplicates and 18 have labeling inconsistency. As shown in the table below, the same duplicate tweet was once flagged as disaster and the other time as nondisaster.



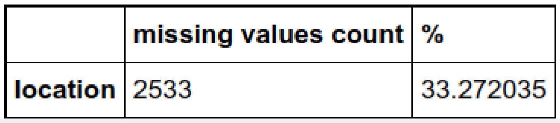
	id	keyword	location	text	target
5641	8044	refugees	NaN	wowo--=== 12000 Nigerian refugees repatriated from Cameroon	0
5620	8018	refugees	NaN	wowo--=== 12000 Nigerian refugees repatriated from Cameroon	1

3. URLs are removed from the tweets.
4. Other cleanings are involved with pre-processing since various models require different cleaning / preprocessing steps.

Exploratory Data Analysis

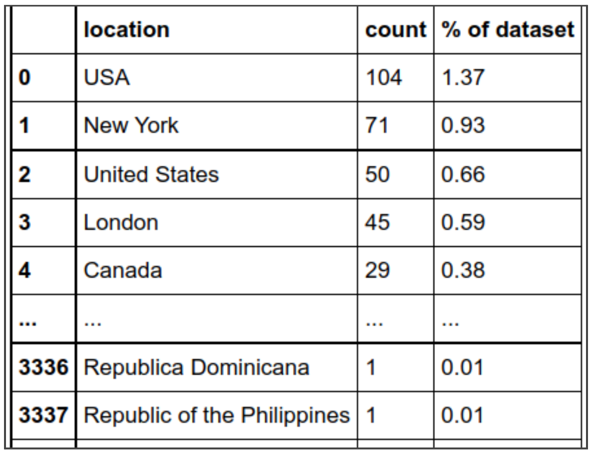
First, Each variable is inspected individually. Second, the correlation between the variables and the target variable is examined using Pearson correlation and the Predictive Power Score.

1. **Inspecting the "location" feature**
   - 33% of the location feature is missing.
   - There are 3341 unique locations for the tweets.



	missing values count	%
location	2533	33.272035

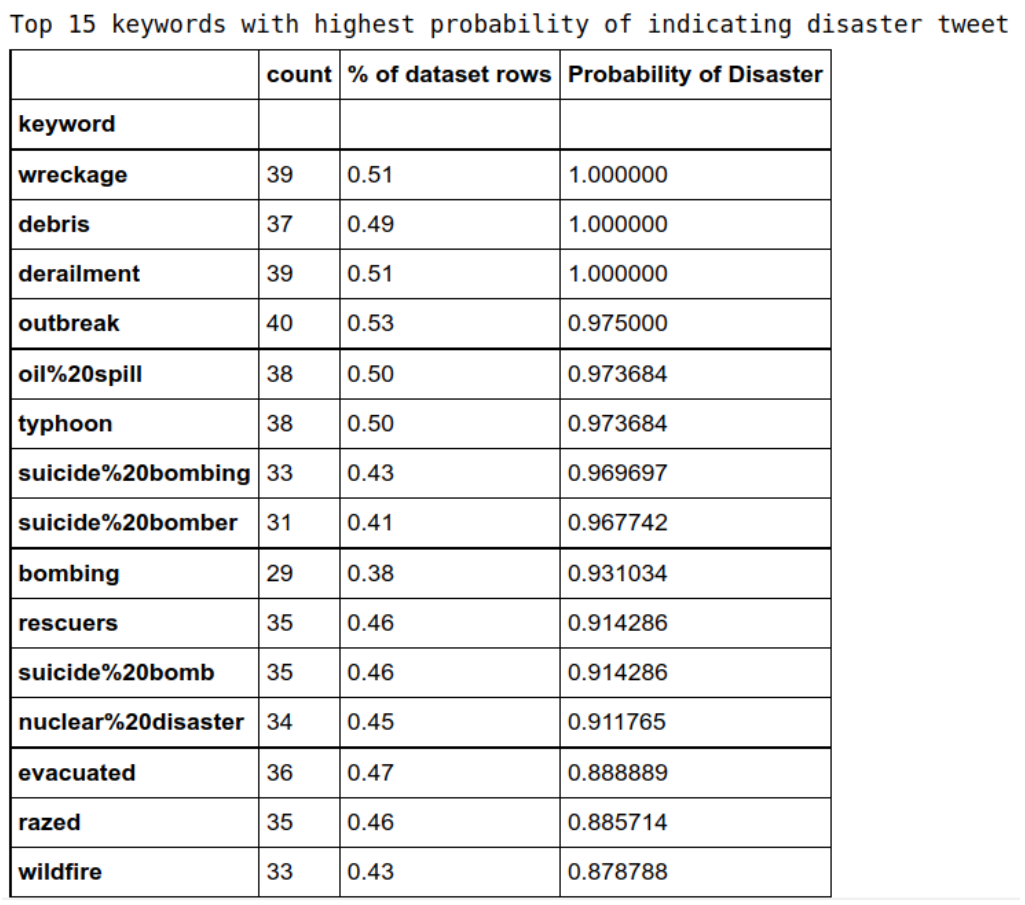
   - The naming of location in this data set is not consistent. We see some tweets' location as a country and others as a city.
   - As shown in the table below, the USA is the most frequent location for the tweets, followed by New York, then the united states representing (1.37%, 0.93%, and 0.66%) of the tweets respectively.



	location	count	% of dataset
0	USA	104	1.37
1	New York	71	0.93
2	United States	50	0.66
3	London	45	0.59
4	Canada	29	0.38
...	...	...	...
3336	Republica Dominicana	1	0.01
3337	Republic of the Philippines	1	0.01

- With this large number of missing values, it is not feasible to clean the location values. The location in this data set is not good to be the predictive feature and will not be considered in the modeling**.**

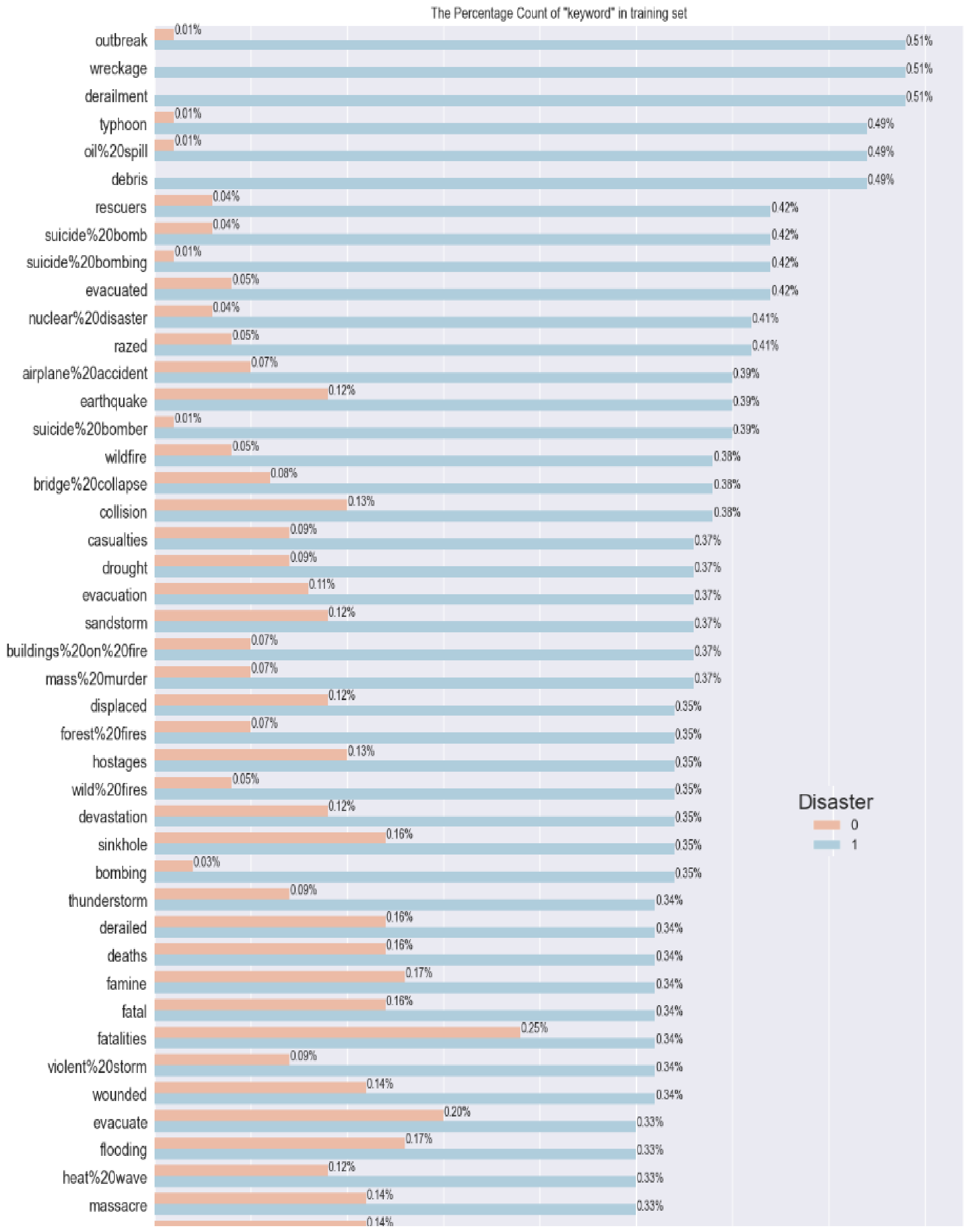
2. **Inspecting the "keyword" feature**
   - There are 61 (0.8%) missing keywords
   - As shown in the graph and table below, keywords derailment, wreckage, and debris only appear in the disaster tweets with disaster probability = 1. Also, outbreak, typhoon, oil%20omb, suicide%20bombing, and rescuers have more than 90%
   - keywords aftershock, body%20bags, ruin, blazing, body%20bag, electrocute, screaming, traumatised, and blew%20up mostly appear in the nondisaster tweets, near-zero probability of disaster tweets.



```
Top 15 keywords with highest probability of indicating disaster tweet
```

	count	**% of dataset rows**	**Probability of Disaster**
keyword			
wreckage	39	0.51	1.000000
debris	37	0.49	1.000000
derailment	39	0.51	1.000000
outbreak	40	0.53	0.975000
oil%20spill	38	0.50	0.973684
typhoon	38	0.50	0.973684
suicide%20bombing	33	0.43	0.969697
suicide%20bomber	31	0.41	0.967742
bombing	29	0.38	0.931034
rescuers	35	0.46	0.914286
suicide%20bomb	35	0.46	0.914286
nuclear%20disaster	34	0.45	0.911765
evacuated	36	0.47	0.888889
razed	35	0.46	0.885714
wildfire	33	0.43	0.878788

The figure below shows the percentage count distribution of the keyword in the train data set. Top disaster tweet keywords are shown



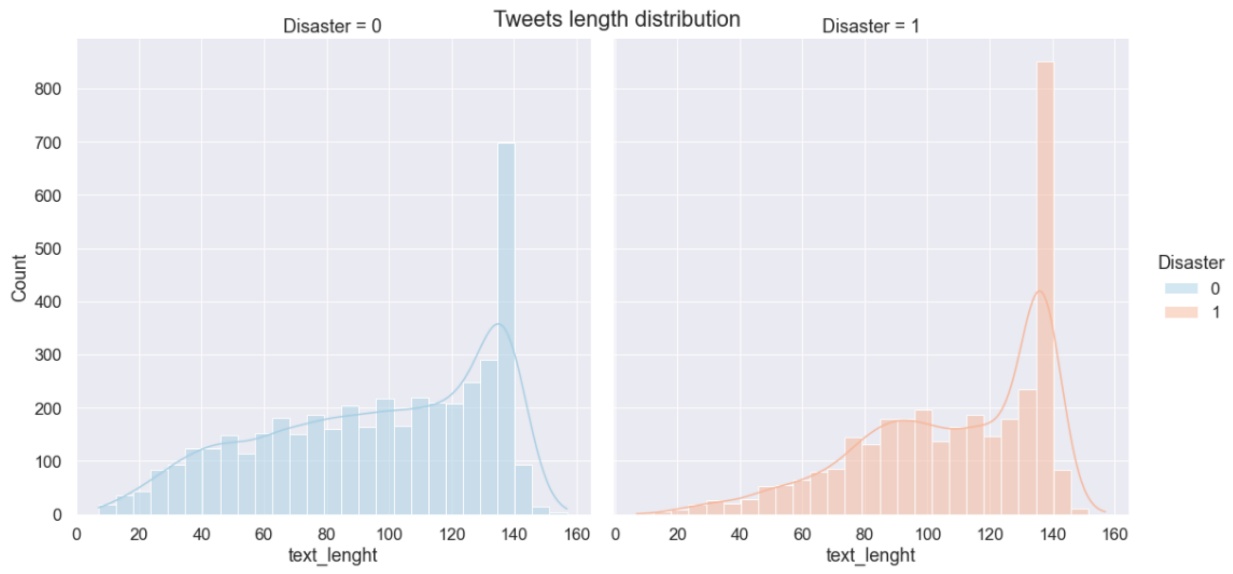
- many other keywords that are existed in both classes. The keyword without the context can not be a good predictor of the disaster tweets.

3. **Inspecting the "text" feature**

The tweet's text is a critical part of this data set. In this section of EDA, the metadata of the tweet text (such as length of the tweets, word count, etc..) and some linguistic features (such as 'noun count', 'pronoun count', auxiliary verb count, and 'verb count'). The latter is hoped to capture some semantic orientation of the tweets.

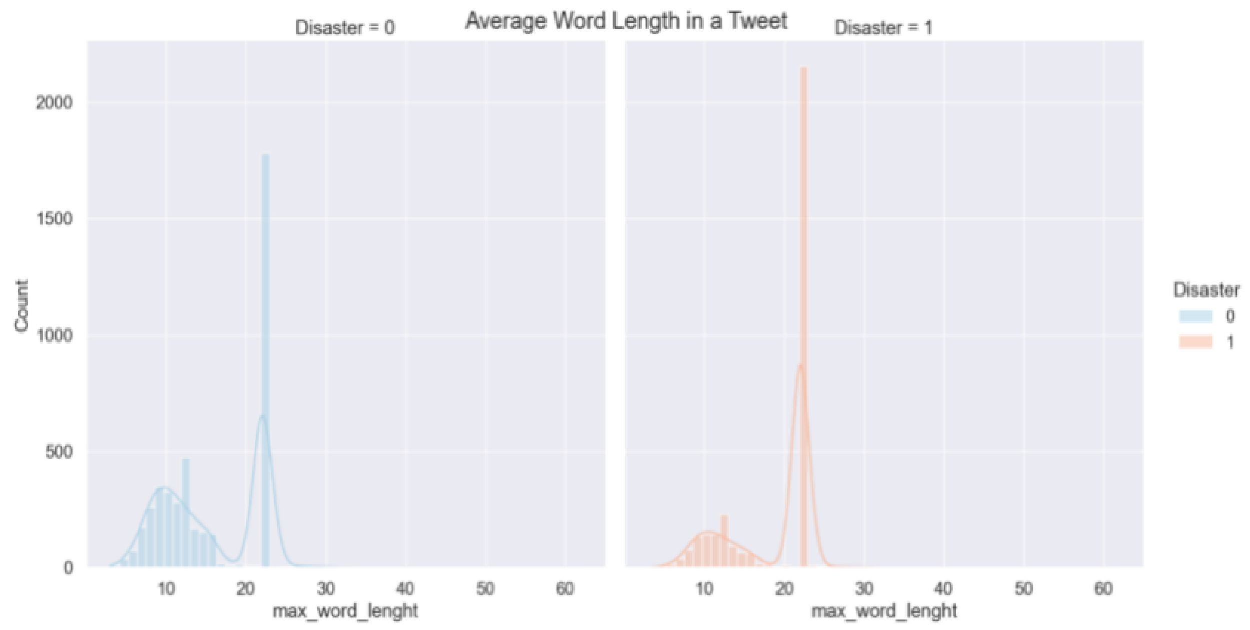
a. Tweets Metadata Analysis

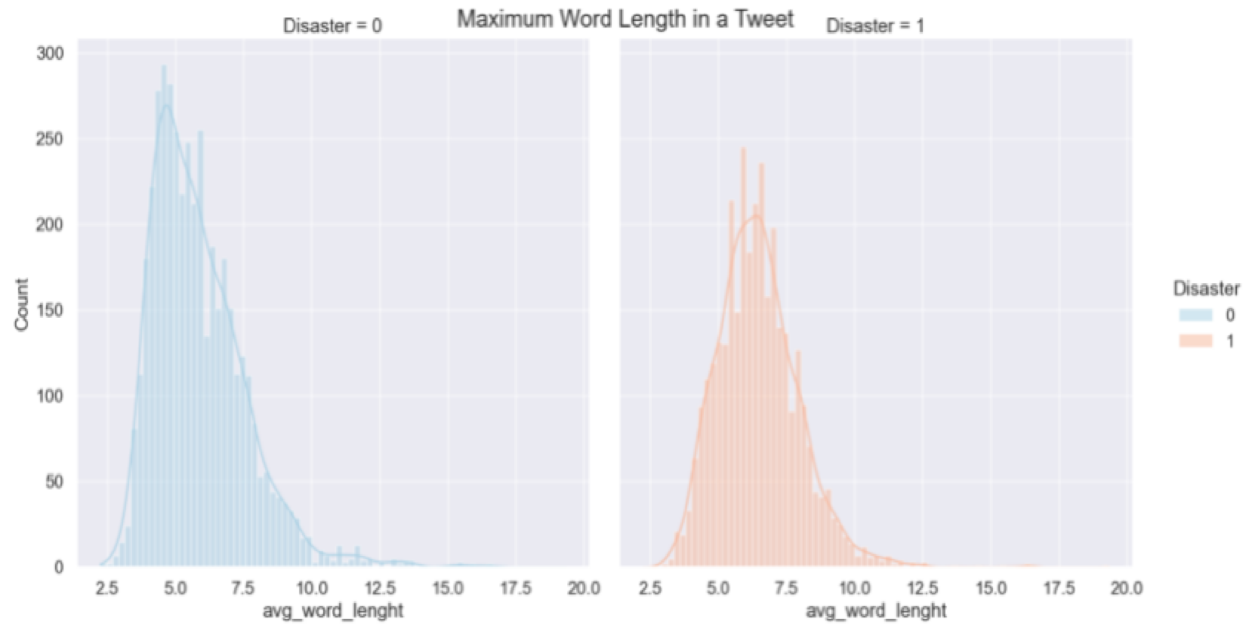
Tweets length (number of characters) distribution for Disaster and None Disaster classes



Both classes have a similar distribution of the tweets' length. Thus, by itself, the tweet length does not seem a good predictor of disaster tweets. The correlation will be examined later.

Average and Maximum word length in a tweet

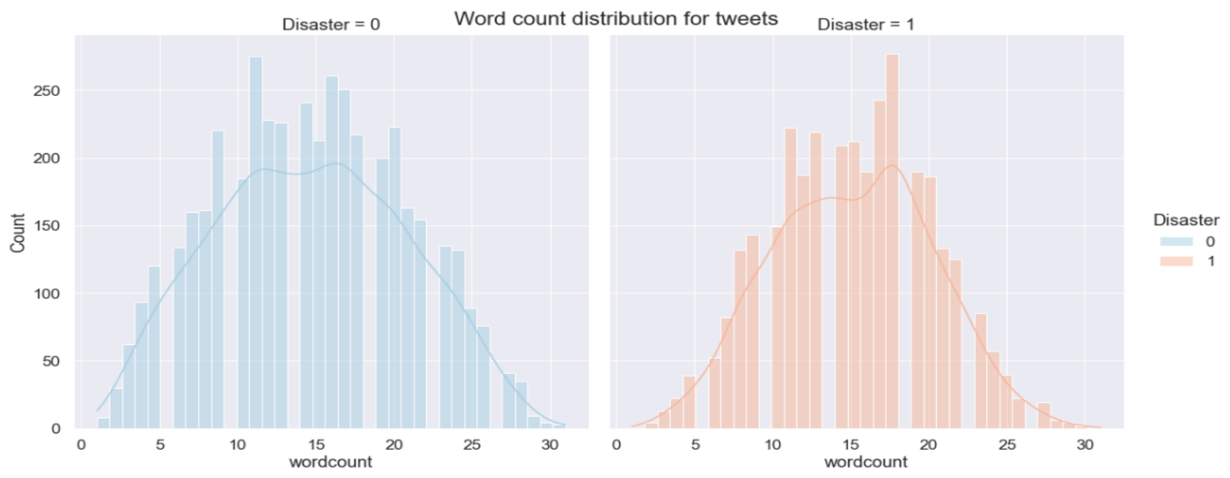




Both, average words length and maximum word length for the tweets have similar distribution for both classes.

Word count distribution for tweets

The word count for tweets has also the same similar distribution for both classes.



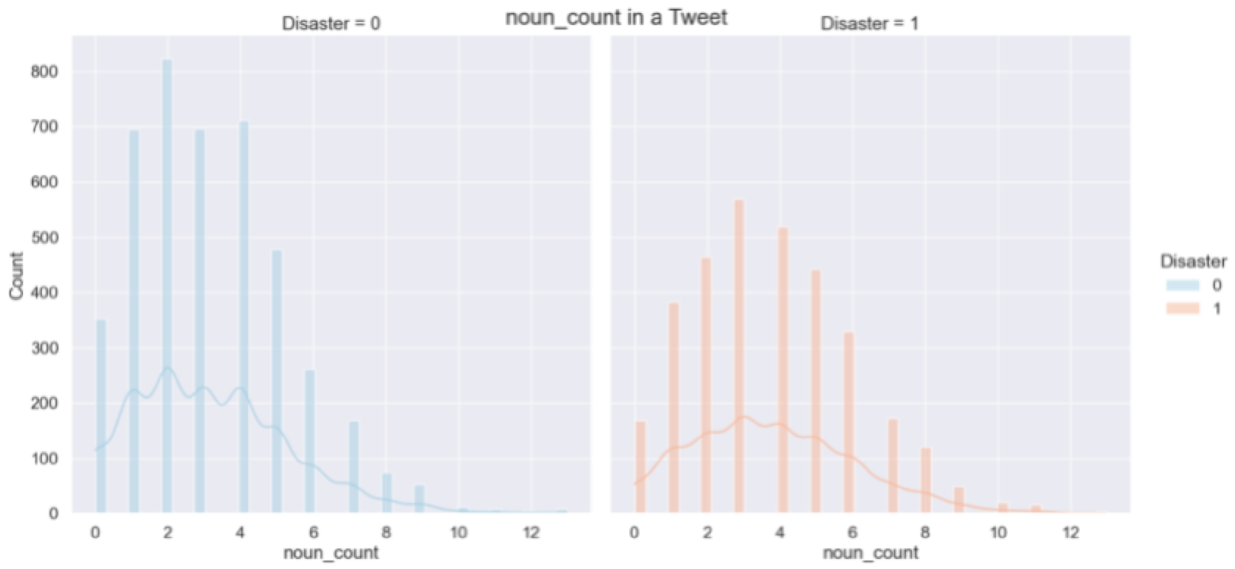
Insights:

The Metadata features of both classes have a similar distribution. None of these Metadata features seems to be a good predictor of the disaster tweets. The predictability of these imputed features will be further checked by the predictive power of these features.

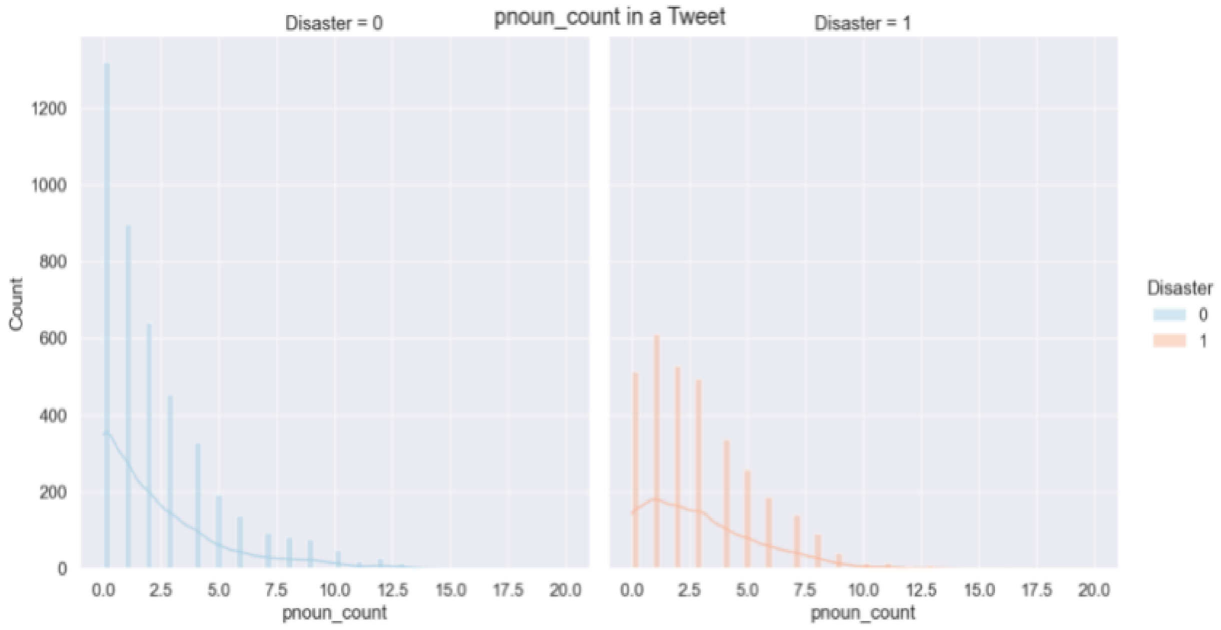
b. Part of Speech Analysis

This section explores the count of nouns, pronouns, auxiliary verbs, and verbs.

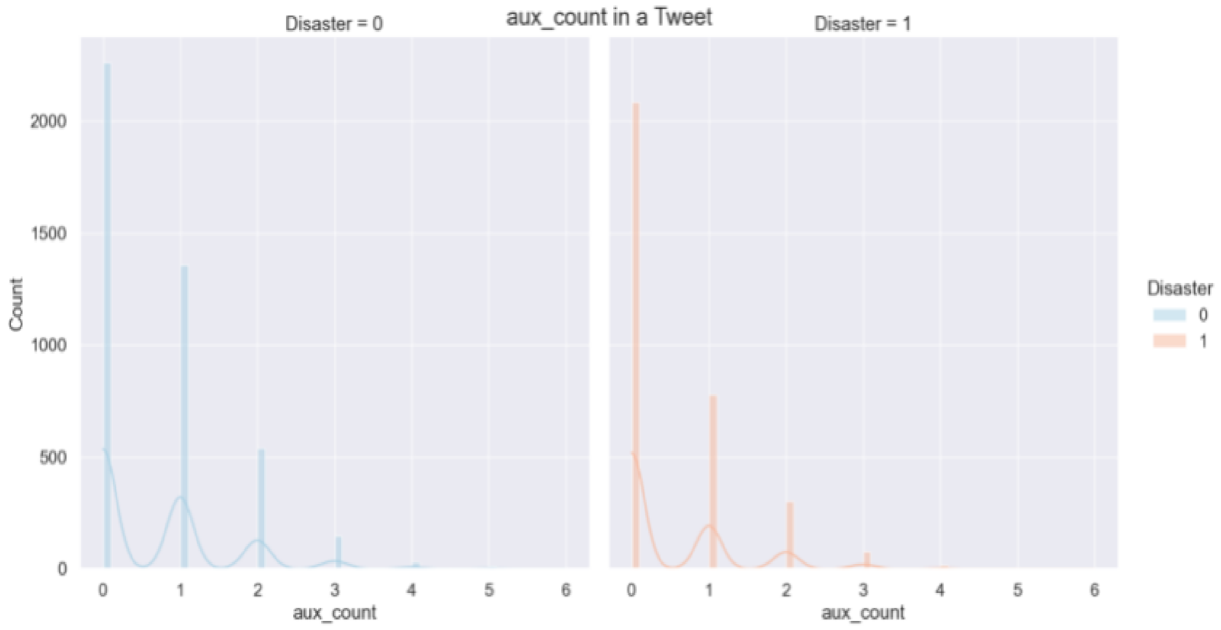
```
Average noun_count for disaster tweet is 3.792892156862745
Average noun_count for none disaster tweet is 3.191998160496666
```



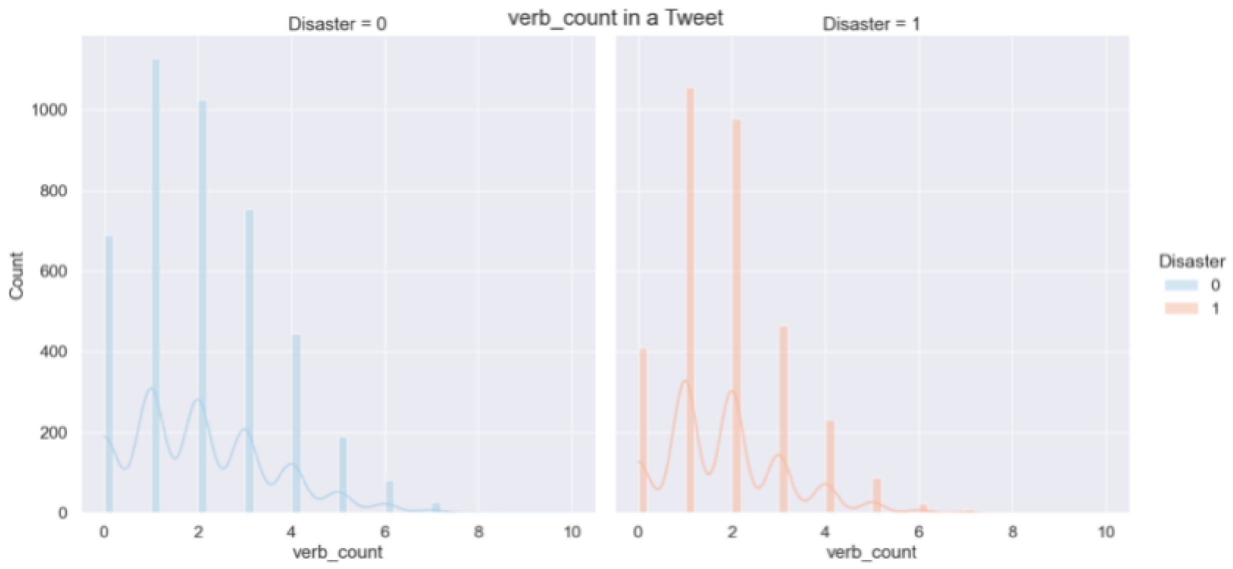
```
Average pnoun_count for disaster tweet is 2.9607843137254903
Average pnoun_count for none disaster tweet is 2.300988733042079
```



```
Average aux_count for disaster tweet is 0.5186887254901961
Average aux_count for none disaster tweet is 0.7052195907105082
```



```
Average verb_count for disaster tweet is 1.8385416666666667
Average verb_count for none disaster tweet is 2.0531156587721315
```



Insights:

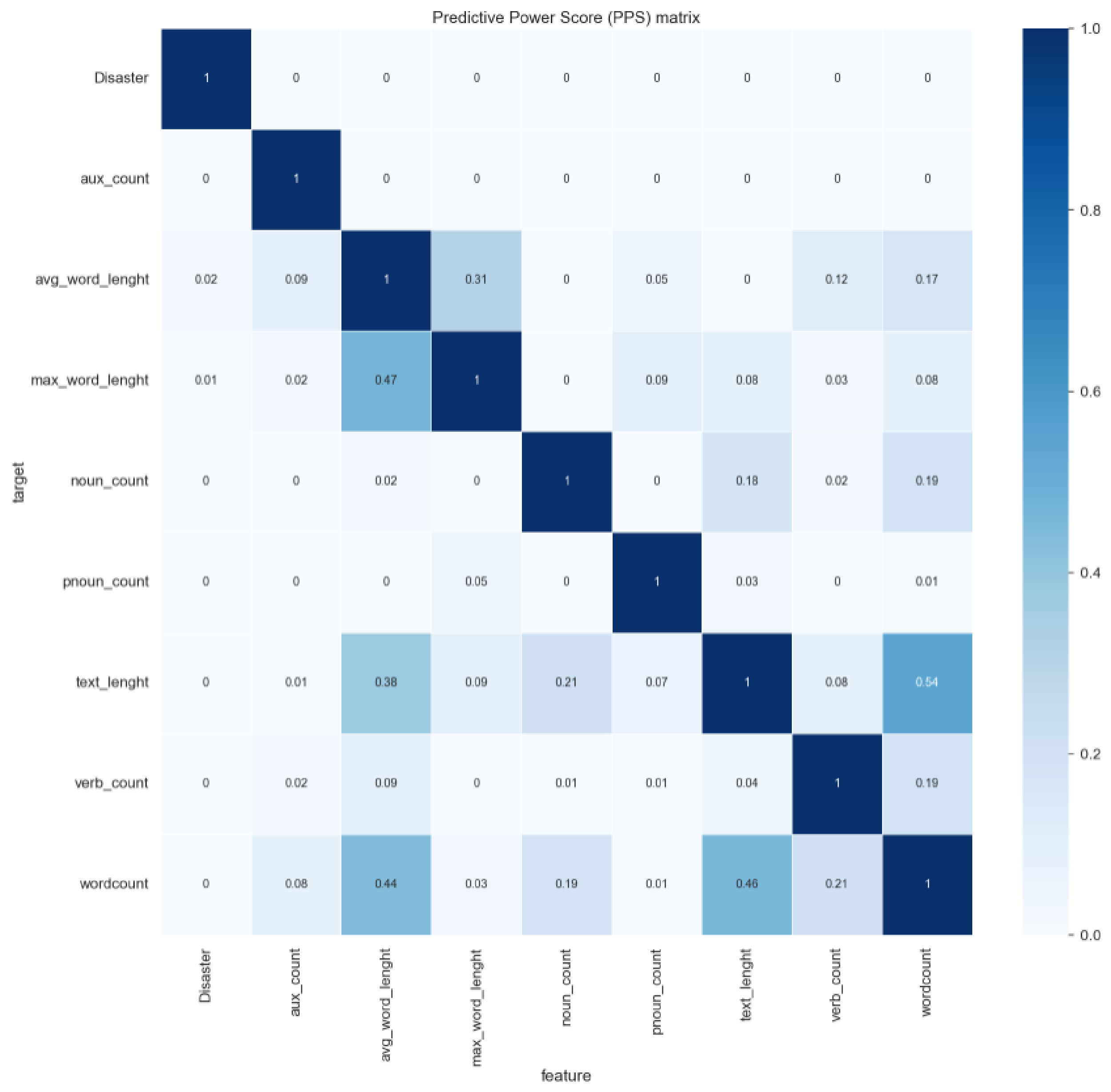
The Linguistic Features, except the proper nouns, seem to have similar distribution for both classes. The predictive power of these features will be examined next.

Predictive Power Analysis

Another approach to finding the correlation through calculating the Predictive Power Score (PPS). The correlation methods such as Pearson and Cramer’s V assume the symmetry of the relationship, which is not always the case, they only work in a particular kind of data, either numeric or categorical,

The PPS is an asymmetric, data-type-agnostic score that can detect linear or non-linear relationships between two columns. The score ranges from 0 (no predictive power) to 1 (perfect predictive power). It can be used as an alternative to the correlation (matrix).

The table below shows the predictive power of all variables with the response variable using PPS.

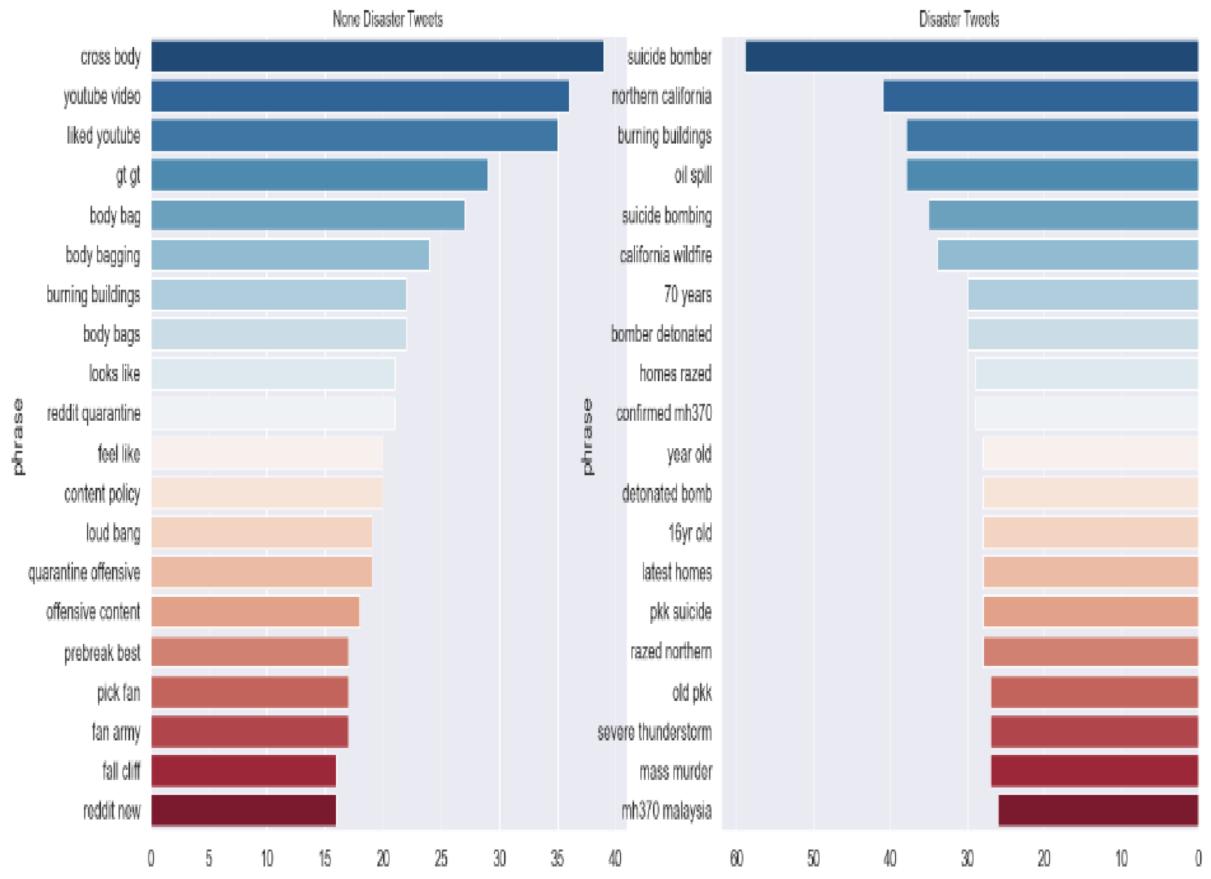


Insights:

As we noticed from the distribution analysis of the metadata features, the PPS confirms that the computed meta-features have no predictive power of the target variable (PPS = 0 for all variables), "disaster" in this case.

Pearson correlation showed some weak correlation of these metadata variables with the maximum word length being the highest (r = 0.25). However, Pearson correlation can give misleading results for a binary classification problem (i.e. examining binary target variable), at least, it is not a good option. Chick this for more details. Based on PPS values, these variables should not be used in modeling. The modeling will rely on the tweet text the most.

Common words and phrases in both classes



Data Pre-processing and Modeling

The first approach is modeling using Term Frequency-Inverse Document Frequency (TF-IDF).

TF-IDF is a popular measure that evaluates how relevant a word is to a tweet in a collection of tweets. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use TF-IDF.

Source: https://kops.uni-konstanz.de/handle/123456789/32348

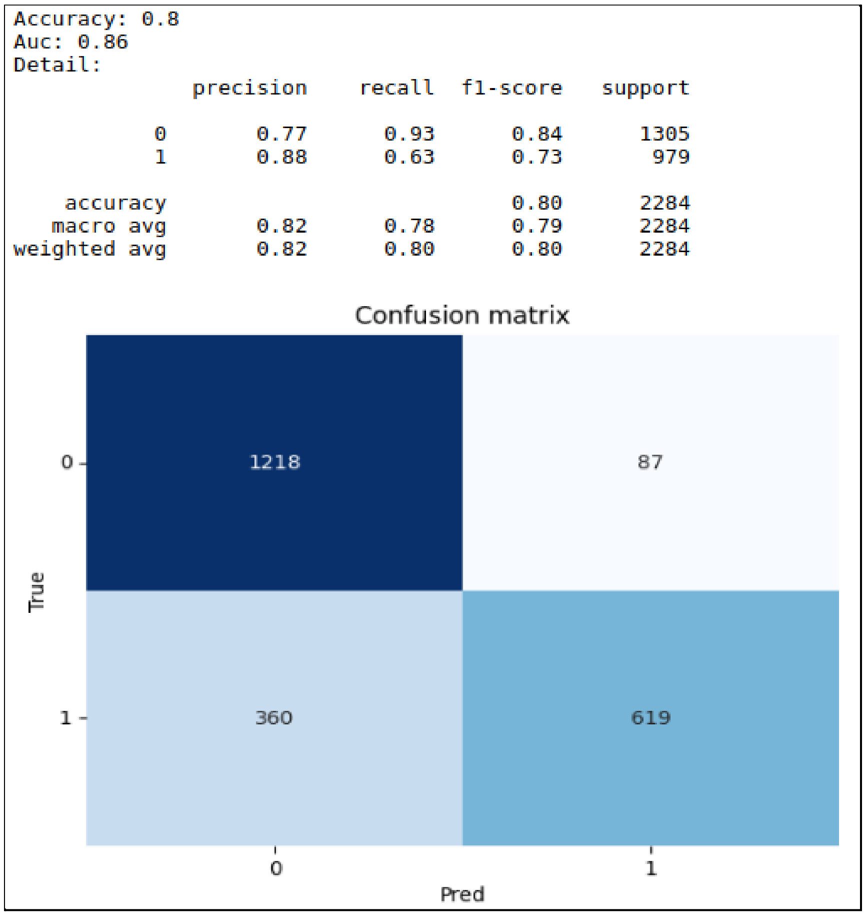
TF-IDF has experimented with various models.

Preprocessing:

The preprocessing step for this approach involved removing English stop words, URLs, digits, hashtags, and tags. Also, text lower casing and lemmatization are applied.

First: Multinomial naïve Bayes

This model's parameters were tuned and optimized by the Grid search algorithm. As shown in the graph below, the model achieved 80% accuracy and AUC ROC = 0.86.

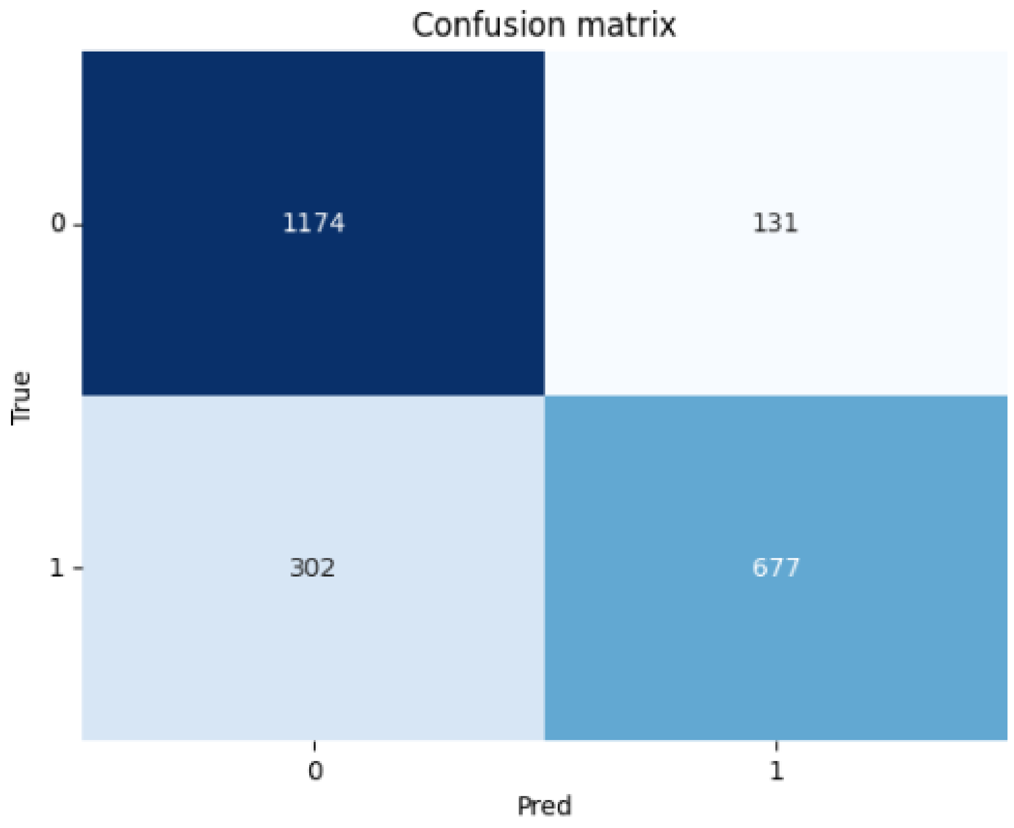


```
Accuracy: 0.8
Auc: 0.86
Detail:
              precision    recall  f1-score   support

           0       0.77      0.93      0.84      1305
           1       0.88      0.63      0.73       979

    accuracy                           0.80      2284
   macro avg       0.82      0.78      0.79      2284
weighted avg       0.82      0.80      0.80      2284
```

The recall for the disaster class is relatively low = 0.63, which indicates that 47% of disaster tweets in the test data set are misclassified.

Second: Logistic Regression

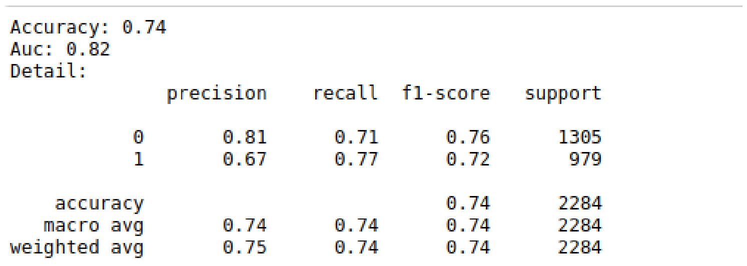
```
Accuracy: 0.81
Auc: 0.86
Detail:
              precision    recall  f1-score   support

           0       0.80      0.90      0.84      1305
           1       0.84      0.69      0.76       979

    accuracy                           0.81      2284
   macro avg       0.82      0.80      0.80      2284
weighted avg       0.81      0.81      0.81      2284
```



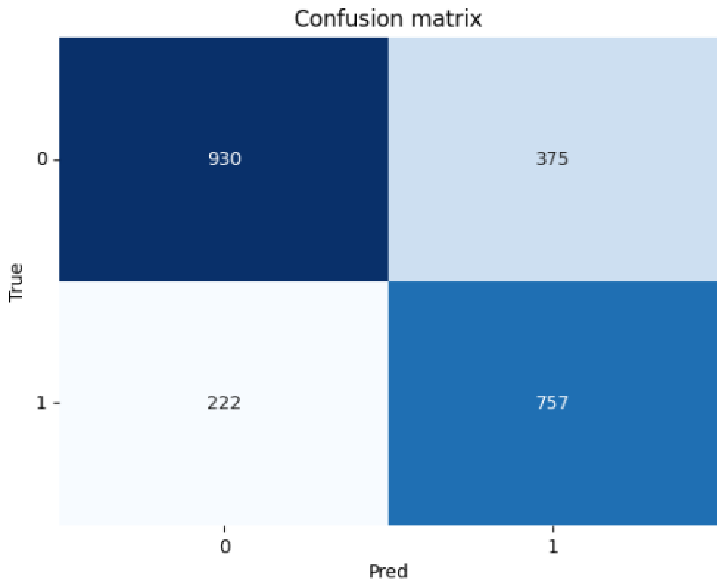
The logistic regression was also optimized using the Grid Search algorithm and gave a better performance. Accuracy = 81%, AUC ROC = 0.86. Alos provided a higher recall on the disaster class, recall = 0.69

Third: XGBoost

Bayesian hyperparameter optimization, using Tree-structured Parzen Estimator Approach (TPE), from the hyperopt package is used to optimize the hyperparameters of the XGBoot model.



```
Accuracy: 0.74
Auc: 0.82
Detail:
              precision    recall  f1-score   support

           0       0.81      0.71      0.76      1305
           1       0.67      0.77      0.72       979

    accuracy                           0.74      2284
   macro avg       0.74      0.74      0.74      2284
weighted avg       0.75      0.74      0.74      2284
```

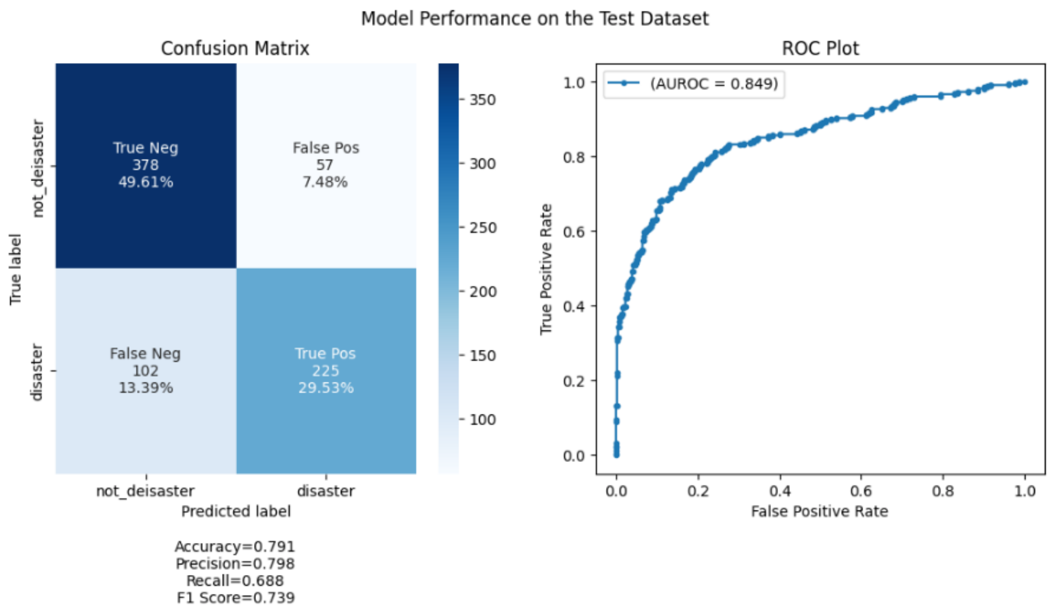


XGBoost provided less performance than the two previous models in terms of (accuracy = 0.74 and AUC = 0.82). However, the model provided more balance in terms of f1 and provided a higher recall = 0.72 on the disaster class.

Forth: Recurrent Neural Network (LSTM) with Dropout

This is a simple Long short-term memory LSTM model with a simple architecture. Also, a dropout is applied to avoid overfitting. The model ended with 495,149 trainable parameters as shown in the figure below.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 23, 32)            472768
_________________________________________________________________
lstm (LSTM)                  (None, 60)                22320
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 61
=================================================================
Total params: 495,149
Trainable params: 495,149
Non-trainable params: 0
_________________________________________________________________
```



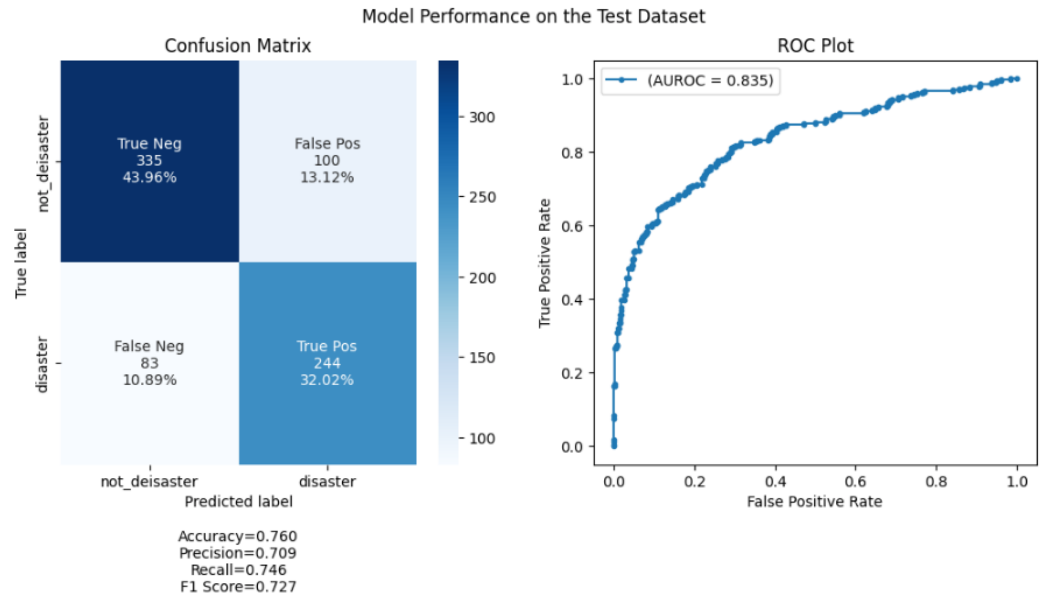
```
Details:
              precision    recall  f1-score   support

           0       0.79      0.87      0.83       435
           1       0.80      0.69      0.74       327

    accuracy                           0.79       762
   macro avg       0.79      0.78      0.78       762
weighted avg       0.79      0.79      0.79       762
```

The model provided more balanced performance. Acc = 79%, AUC = 0.85, recall = 0.688

Fifth: Convolutional Neural Network (CNN)

As shown in the architecture below, the model ended with 494,889 trainable parameters. The graph shows that this model provided a better recall of 0.75, the model does better in classifying the disaster tweets but with more folse positive

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
embedding_6 (Embedding)     (None, 23, 32)            472768
_________________________________________________________________
conv1d_6 (Conv1D)           (None, 23, 64)            8256
_________________________________________________________________
dropout_3 (Dropout)         (None, 23, 64)            0
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 11, 64)           0
_________________________________________________________________
conv1d_7 (Conv1D)           (None, 11, 32)            8224
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 5, 32)            0
_________________________________________________________________
conv1d_8 (Conv1D)           (None, 5, 8)              1032
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 2, 8)             0
_________________________________________________________________
flatten_5 (Flatten)         (None, 16)                0
_________________________________________________________________
dense_9 (Dense)             (None, 256)               4352
_________________________________________________________________
dropout_4 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_10 (Dense)            (None, 1)                 257
=================================================================
Total params: 494,889
Trainable params: 494,889
Non-trainable params: 0
```



```
Details:
              precision    recall  f1-score   support

           0       0.80      0.77      0.79       435
           1       0.71      0.75      0.73       327

    accuracy                           0.76       762
   macro avg       0.76      0.76      0.76       762
weighted avg       0.76      0.76      0.76       762
```

The second approach is using Transfer Learning / Deep Learning through BERT transformer

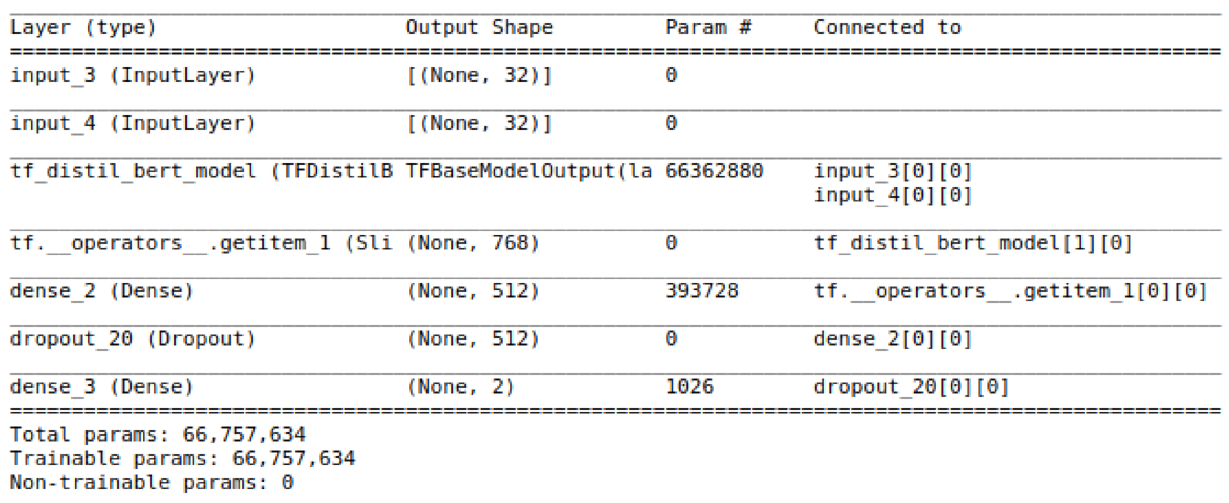
Bidirectional Encoder Representations from Transformers (BERT)

BERT is state-of-the-art in NLP. BERT was developed by Google as a transformer-based machine learning technique for natural language processing (NLP) pre-training. BERT large with 12 encoders and 12 decoders is used here. The experiment also involved used DistilBERT, which is, according to the HuggingFace, a small, fast, cheap, and light Transformer model trained by distilling BERT base. All the models were trained using 16 GigaByte Ram / 4 cores laptop.

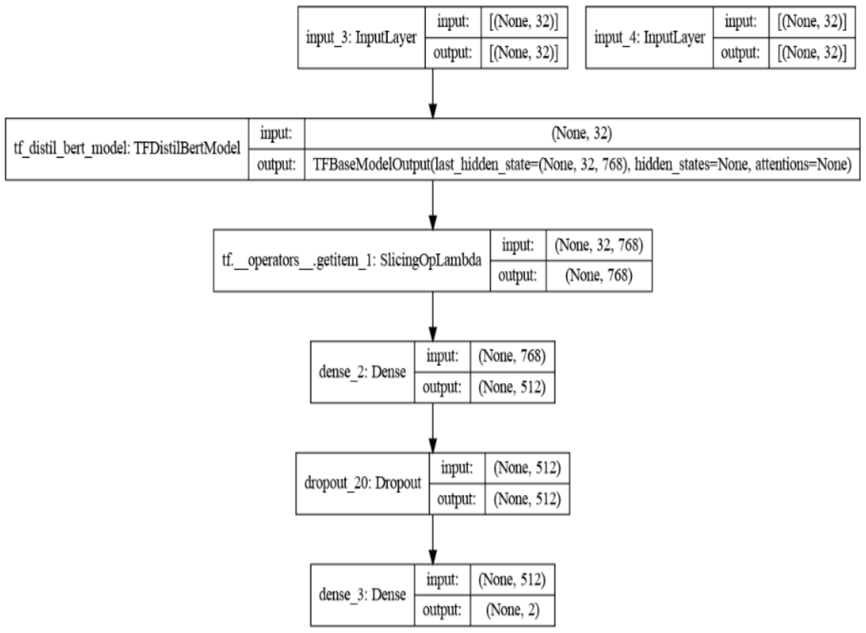
DistilBERT uncased

The pre-trained model is provided by the HuggingFace is used to cope with the limitation of the resources. Source: https://huggingface.co/transformers/model_doc/distilbert.html

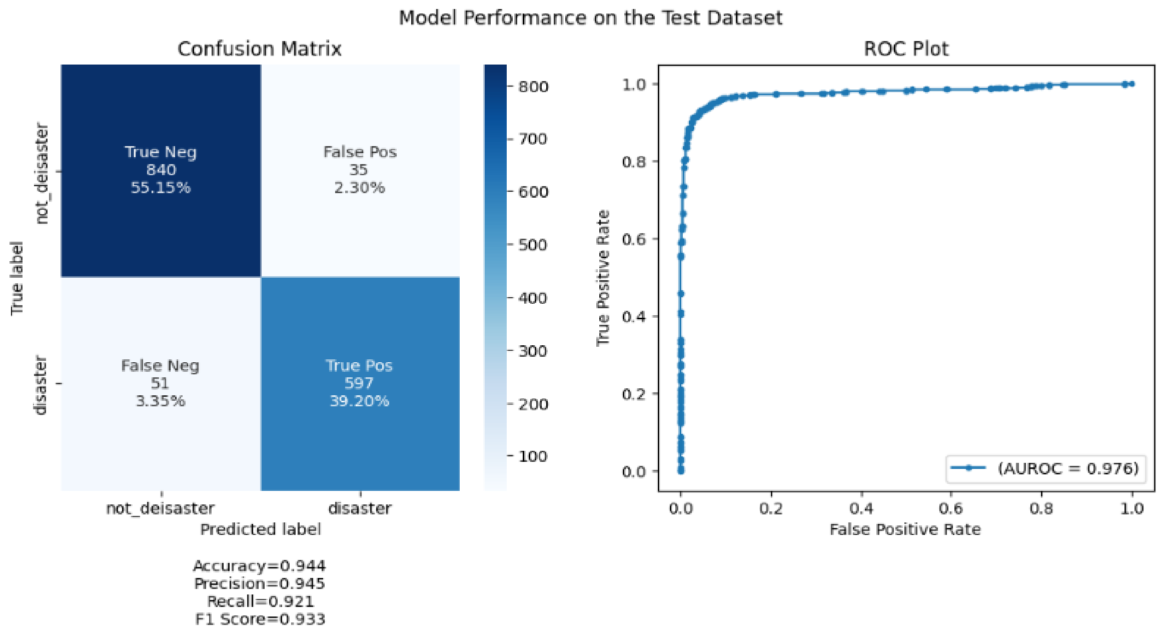
The model as shown below has 66, 757, 634 trainable parameters. The model was trained using 5 epochs and the training time was about 100 minutes. The model provided an impressive performance as shown below.



```
______________________________________________________________________________________________________
Layer (type)                     Output Shape         Param #     Connected to
======================================================================================================
input_3 (InputLayer)             [(None, 32)]         0
______________________________________________________________________________________________________
input_4 (InputLayer)             [(None, 32)]         0
______________________________________________________________________________________________________
tf_distil_bert_model (TFDistilB TFBaseModelOutput(la 66362880     input_3[0][0]
                                                                  input_4[0][0]
______________________________________________________________________________________________________
tf.__operators__.getitem_1 (Sli (None, 768)          0            tf_distil_bert_model[1][0]
______________________________________________________________________________________________________
dense_2 (Dense)                  (None, 512)          393728      tf.__operators__.getitem_1[0][0]
______________________________________________________________________________________________________
dropout_20 (Dropout)             (None, 512)          0           dense_2[0][0]
______________________________________________________________________________________________________
dense_3 (Dense)                  (None, 2)            1026        dropout_20[0][0]
======================================================================================================
Total params: 66,757,634
Trainable params: 66,757,634
Non-trainable params: 0
```


input_3: InputLayer | input: | [(None, 32)] | output: | [(None, 32)]
input_4: InputLayer | input: | [(None, 32)] | output: | [(None, 32)]
tf_distil_bert_model: TFDistilBertModel | input: | (None, 32) | output: | TFBaseModelOutput(last_hidden_state=(None, 32, 768), hidden_states=None, attentions=None)
tf.__operators__.getitem_1: SlicingOpLambda | input: | (None, 32, 768) | output: | (None, 768)
dense_2: Dense | input: | (None, 768) | output: | (None, 512)
dropout_20: Dropout | input: | (None, 512) | output: | (None, 512)
dense_3: Dense | input: | (None, 512) | output: | (None, 2)

DistilBERT performance

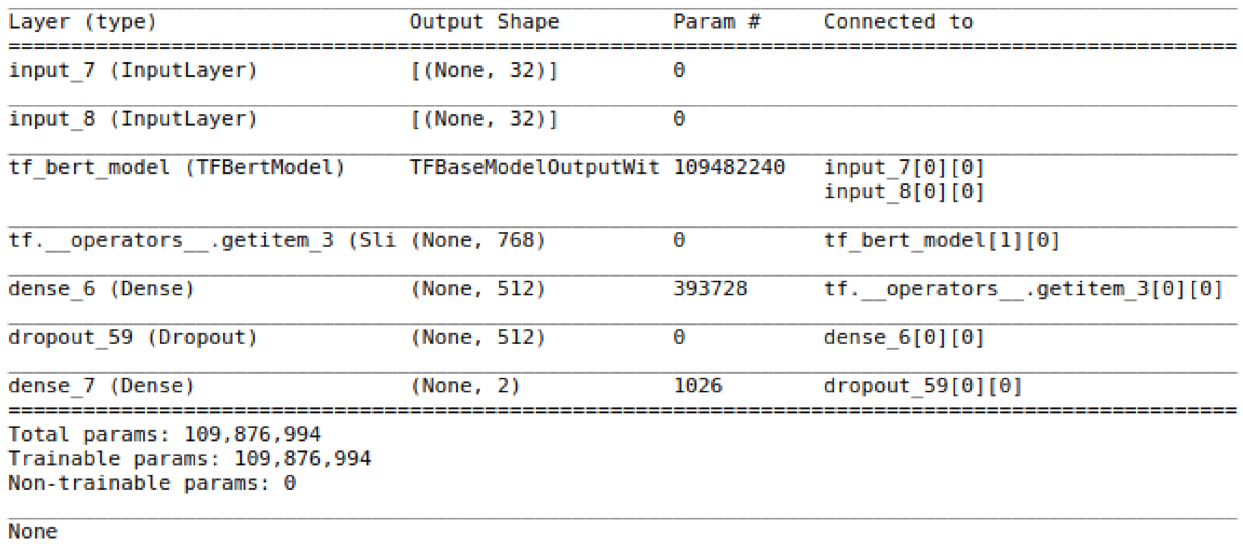


```
Details:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       875
           1       0.94      0.92      0.93       648

    accuracy                           0.94      1523
   macro avg       0.94      0.94      0.94      1523
weighted avg       0.94      0.94      0.94      1523
```

DistilBERT baes model provided an excellent performance accuracy = 94%, F1 = 0.93 on the disaster tweets class, recall = 0.92 and AUC = 0.97.

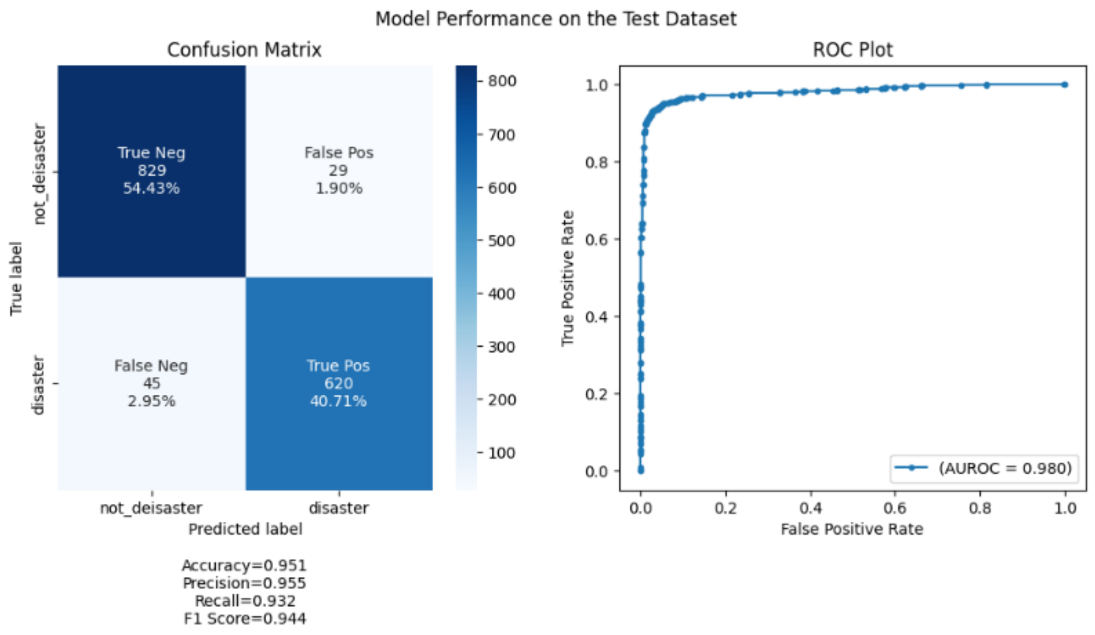
BERT base uncases

This experiment involved using the larger pre-trained model “bert_base_uncased.” This model has 109,876,994 trainable parameters. It took more than 9 hours to train this model for 30 epochs on 16 GByte ram with 4 cores CPU.



```
________________________________________________________________________________________________________________
Layer (type)                     Output Shape         Param #     Connected to
================================================================================================================
input_7 (InputLayer)             [(None, 32)]         0

________________________________________________________________________________________________________________
input_8 (InputLayer)             [(None, 32)]         0

________________________________________________________________________________________________________________
tf_bert_model (TFBertModel)      TFBaseModelOutputWit 109482240   input_7[0][0]
                                                                  input_8[0][0]
________________________________________________________________________________________________________________
tf.__operators__.getitem_3 (Sli (None, 768)           0           tf_bert_model[1][0]
________________________________________________________________________________________________________________
dense_6 (Dense)                  (None, 512)          393728      tf.__operators__.getitem_3[0][0]
________________________________________________________________________________________________________________
dropout_59 (Dropout)             (None, 512)          0           dense_6[0][0]
________________________________________________________________________________________________________________
dense_7 (Dense)                  (None, 2)            1026        dropout_59[0][0]
================================================================================================================
Total params: 109,876,994
Trainable params: 109,876,994
Non-trainable params: 0
________________________________________________________________________________________________________________
None
```

As shown in the figure below, this model gave an even better performance with an accuracy of 95% and a recall of 0.93. However, it needed more time to train.



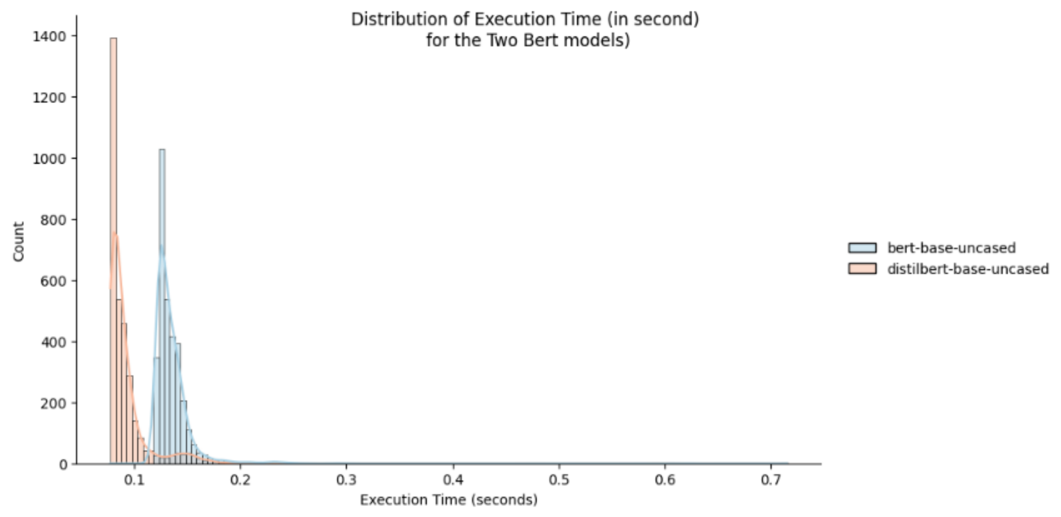
```
Details:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       858
           1       0.96      0.93      0.94       665

    accuracy                           0.95      1523
   macro avg       0.95      0.95      0.95      1523
weighted avg       0.95      0.95      0.95      1523
```

Comparing models speed performance in production

The prediction execution time for both models was measured by predicting the test set of 3236 samples. The average execution time for bert_base_uncased is 0.135 seconds and for distilbert 0.092.

	count	mean	std	min	25%	50%	75%	max
bert-base-uncased	3263.0	0.135093	0.022171	0.120632	0.124457	0.130588	0.139596	0.715436
distilbert-base-uncased	3263.0	0.092004	0.023754	0.077504	0.080438	0.085043	0.093514	0.603825

The figure below shows the distribution of the prediction execution time for both models. As expected, the distilbert is faster than bert_base. The table below sowed that bert_base_uncased did pretty well on predicting the unlabeled tweets test set.



	text	Disaster Prediction
0	Just happened a terrible car crash	1
1	Heard about #earthquake is different cities, stay safe everyone.	1
2	there is a forest fire at spot pond, geese are fleeing across the street, I cannot save them all	1
3	Apocalypse lighting. #Spokane #wildfires	1
4	Typhoon Soudelor kills 28 in China and Taiwan	1
5	We're shaking...It's an earthquake	1
6	They'd probably still show more life than Arsenal did yesterday, eh? EH?	0
7	Hey! How are you?	0
8	What a nice hat?	0
9	Fuck off!	0
10	No I don't like cold!	0
11	NOOOOOOOOO! Don't do that!	0
12	No don't tell me that!	0
13	What if?!	0
14	Awesome!	0
15	Birmingham Wholesale Market is ablaze BBC News - Fire breaks out at Birmingham's Wholesale Market	1
16	@sunkxssedharry will you wear shorts for race ablaze ?	0
17	#PreviouslyOnDoyinTv: Toke Makinwa◗Ûªs marriage crisis sets Nigerian Twitter ablaze...	1
18	Check these out: #nsfw	0
19	PSA: I◗Ûªm splitting my personalities.\n\n?? techies follow @ablaze_co\n?? Burners follow @ablaze	0
20	beware world ablaze sierra leone &amp; guap.	0
21	Burning Man Ablaze! by Turban Diva via @Etsy	0
22	Not a diss song. People will take 1 thing and run with it. Smh it's an eye opener though. He is about 2 set the game ablaze @CyhiThePrynce	0
23	Rape victim dies as she sets herself ablaze: A 16-year-old girl died of burn injuries as she set herself ablaze◗Û_	1
24	SETTING MYSELF ABLAZE	1
25	@CTVToronto the bins in front of the field by my house wer set ablaze the other day flames went rite up the hydro pole wonder if it was him	1
26	#nowplaying Alfons - Ablaze 2015 on Puls Radio #pulsradio	0
27	'Burning Rahm': Let's hope City Hall builds a giant wooden mayoral effigy 100 feet tall &amp; sets it ablaze. @John_Kass	0
28	@PhilippaEilhart @DhuBlath hurt but her eyes ablaze with insulted anger.	0
29	Accident cleared in #PaTurnpike on PATP EB between PA-18 and Cranberry slow back to #traffic	1

Conclusion and recommendations

- Both BertDistill and Bert_large_uncased have produced the best performance.

- Bert_large_uncased is the winning model with slightly better performance.

- However, training Bert_large_uncased with 109,876,994 trainable parameters for 30 epochs required more than 9 hours on a 16GB Ram / 4 cores CPU laptop.
- Although Distilled Bert provided slightly less prediction performance, it is faster considerably faster than Bert base model.

- For predicting streaming tweets in real-time, scalability and latency are a big deal. Depending on the production environment, a trade-off needs to be considered between slightly more performance and faster execution time.

- **Future recommendations**

- Test the model in the production environment and determine which provides more scalability.

- The model is trained on a small labeled data set. The performance should be monitored and improved by using more representative data.