Natural Language Processing
with Kaggle Disaster Tweets

Katarzyna Rogalska Jakub Półtorak Michał Pytel

May 2024

Abstract

In this project, our objective is to classify tweets related to disasters, determining whether they indicate a genuine threat or are misleading. The process will be divided into three main phases: Exploratory Data Analysis, Feature Engineering, and Modeling. Finally, we will compare the results from various models and select the most effective one. The code to our solution can be found in this Github repository.

1 Introduction

Social media platforms such as Twitter, Facebook, and Instagram play a significant role in contemporary life. They serve as major sources of information, capable of informing the public about dangers. Is it possible for a machine to distinguish the meaning of the word 'fire' in different tweets, such as 'Fire in New York, stay safe' and 'That's a fire photo'? This question inspired our project.

We fetched our data from Kaggle Disaster Tweets Competition. In our dataframe there are columns like unique Id, location, keyword and most importantly - Tweet's text. Having this information we built a machine learning model that can predict whether a given tweet informs about real danger or not.

2 Process description

Now, we will briefly summarize the milestones we reached during the development process.

2.1 Data analysis

The most important part of building a machine learning model is understanding and analyzing the data. We checked the data types and detected missing values. But how can we further analyze the text? Can we extract additional information just from a tweet's content? To gain new, deeper insights, we used the following techniques and we will present the most interesting conclusions.

Target analysis
The first thing we did was analyse the distribution of our binary target variable. Let's remind that tagert = 1 informs that a given Tweet was classified as informing about a true disaster. The table below shows the percentages of target values.

	disaster Tweets	non disaster Tweets
percentage:	43%	57%

Table 1: Distribution of the target variable

Special symbols analysis
How many exclamation marks, mentions, hashtags, links etc does every Tweet contain? We checked the relationship between those features and the target. Does the number of words in Caps Lock determine whether a Tweet is informing about a real disaster?

- if there are no hashtags, exclamation marks, words in caps lock the Tweet is more likely to be just a casual post
- we detected a visible pattern between the number of punctuation marks and the Tweet's intention, as shown in the plot below. It is evident that Tweets informing about some kind of danger tend to have more punctuation than casual Tweets

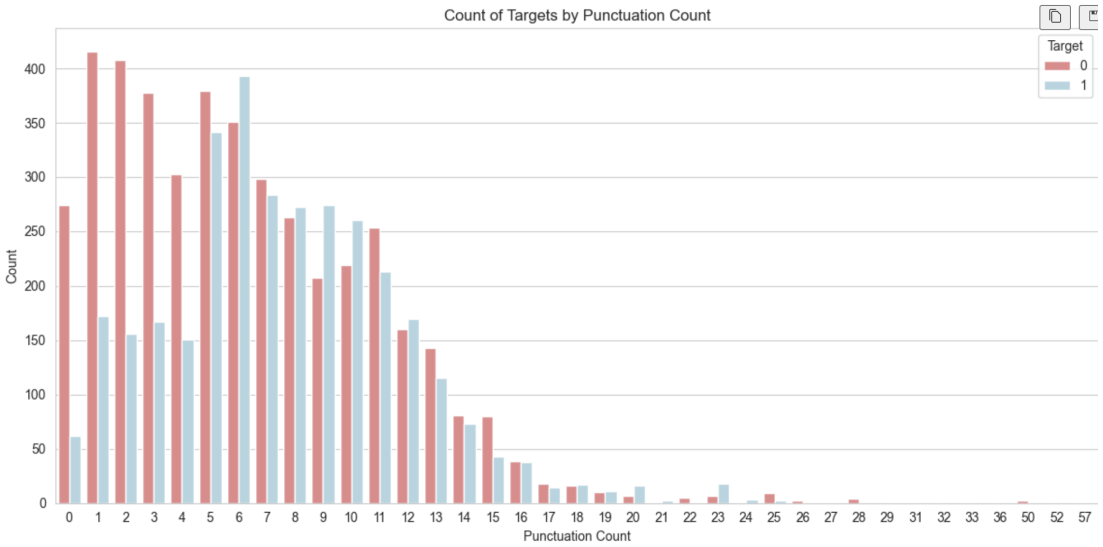

Figure 1: punctuation marks vs target

Number of words analysis
Does the length of a tweet affect it's reliability? The relationship is not clearly visible, but from the plot below we can observe that if a Tweet is very short of very long it is more likely to be just a casual post. Most of the disaster Tweets have a word count from 10 to 20.

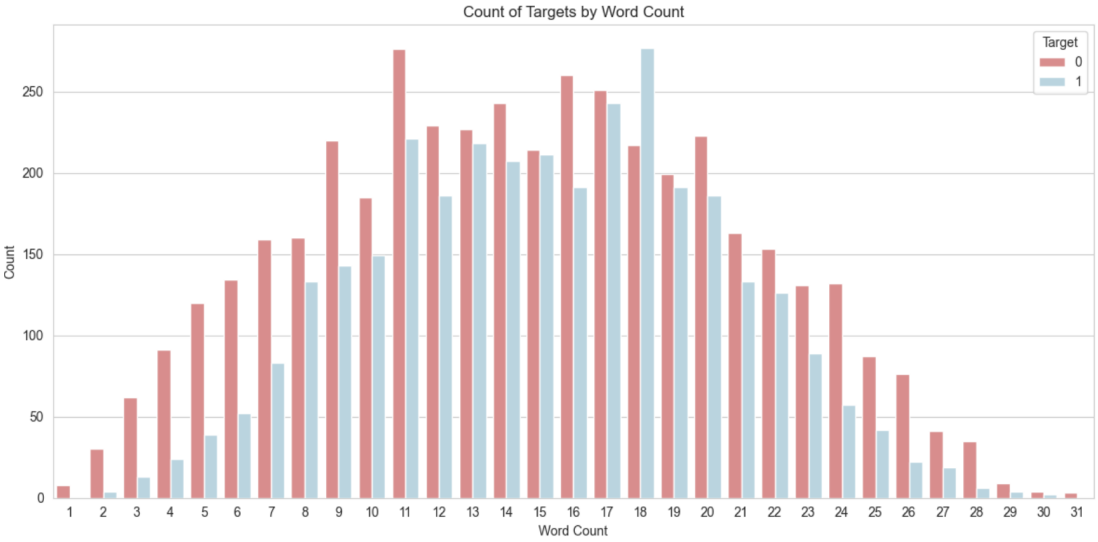

Figure 2: word count vs target

Stopwords detection
Stopwords are words in English like 'a', 'the', 'is' etc, that don't have any significant meaning. We checked the relationship between the number of stopwords and tweet's informativeness.We didn't discover any relevant relatinships from the plots, as well as from correlation with target which was only -0.08.

Parts of speech division
Detecting the number of nouns, verbs, adjectives, and adverbs in a given tweet was an important step in this process. We later discovered that the number of nouns in a tweet has a quite high correlation of 0.2 with the target variable what we can also observe from the plots in Figure 3 - true disaster Tweets tend to have more nouns.

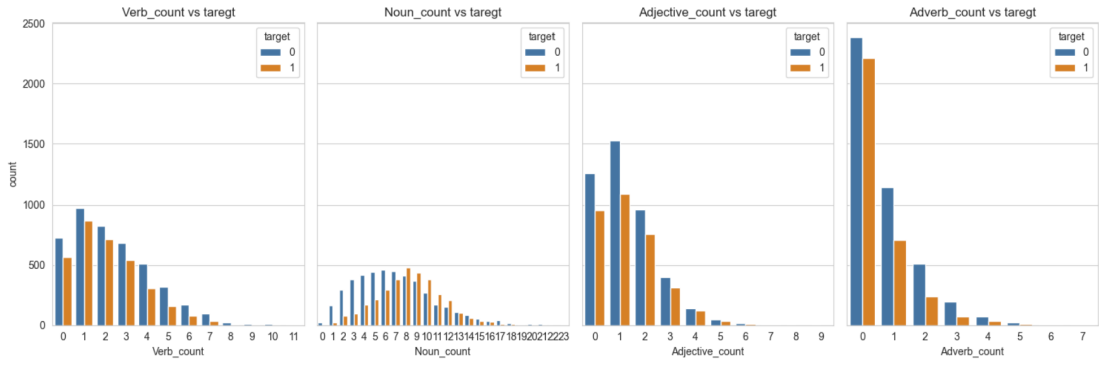

Figure 3: parts of speech vs target

Sentiment analysis
We used available tools to determine each Tweet's sentiment and objectivity. Additionally, we took a step further by extracting all the emotions detected in each tweet. The plot on Figure 4 shows the distribution of polarity score (-1 - negative, 0- neutral, 1 - positive sentiment) and objectivity score (0-objective, 1 - subjective)

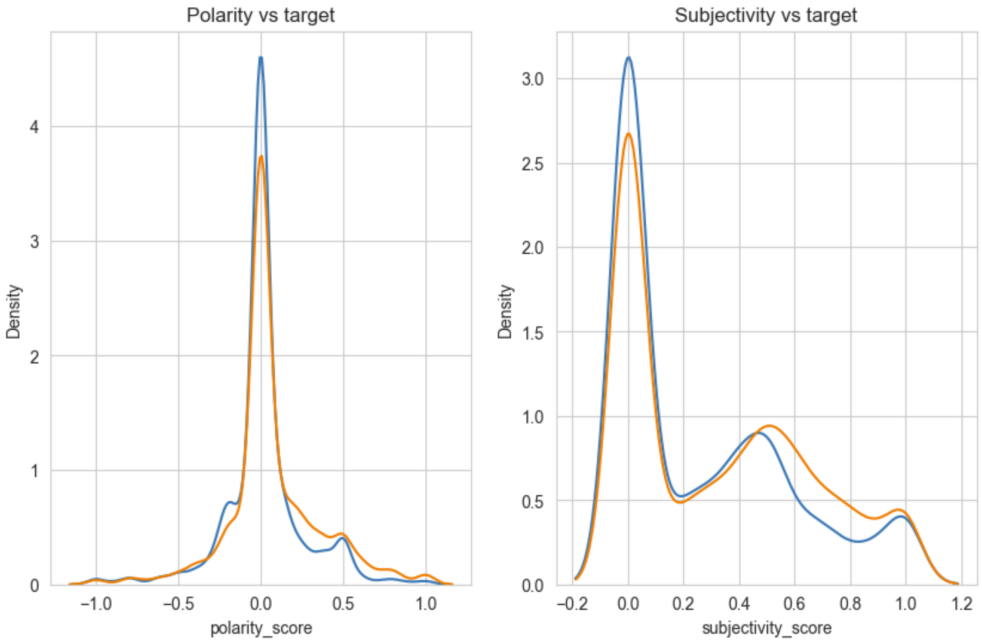

Figure 4: sentiment vs target

We came to the conclusion that the majority of the Tweets are neutral and objective. Furthermore, the distribution of these features is very similar for both disaster and non-disaster Tweets.

Specific word's relationship with target
We were interested in identifying which words appear most frequently in true and fake tweets, as well as understanding the impact of "God-like" words on a tweet's reliability.Firstly, we detected five keywords that frequently appear in Tweets using re package. In the table below, we present these keywords and their presence in Tweets informing about a real danger and in casual Tweets. We observed that many of the keywords appear more frequently in casual Tweets, so we made a visualisations of words that appear most frequently in disaster as well as non-disaster Tweets:

Keyword	Real Danger Tweets	Casual Tweets
'fatalities'	57.8%	42.2%
'deluge'	14.3%	85.7%
'armageddon'	11.9%	88.1%
'sinking'	19.5%	80.5%
'damage'	46.3%	53.7%

Table 2: presence of keywords in real danger and casual Tweets

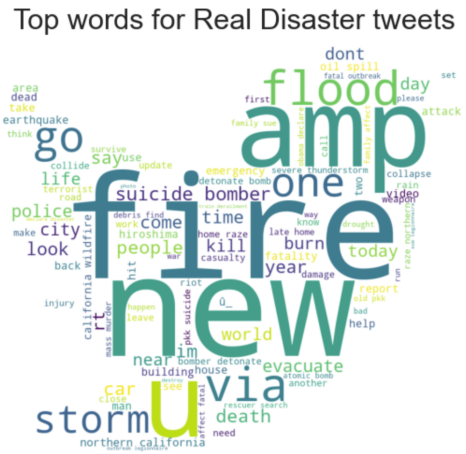

Figure 5: Frequent words in disaster Tweets

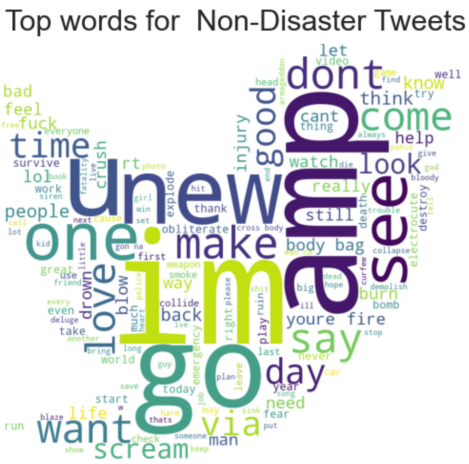

Figure 6: Frequent words in non-disaster Tweets

Then we used another method of extracting frequent phrases : Term Frequency Score and also visualised the results:

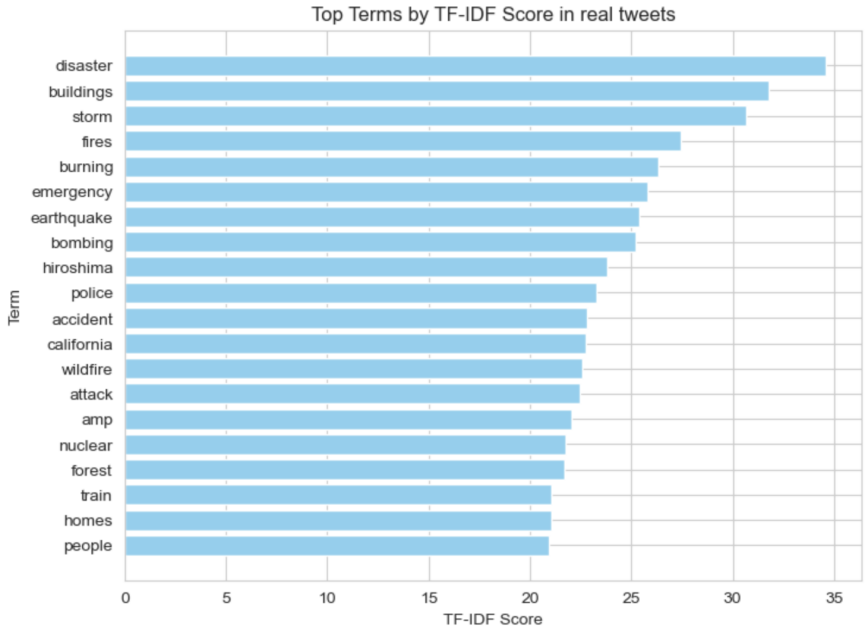

Figure 7: Frequent words in disaster Tweets

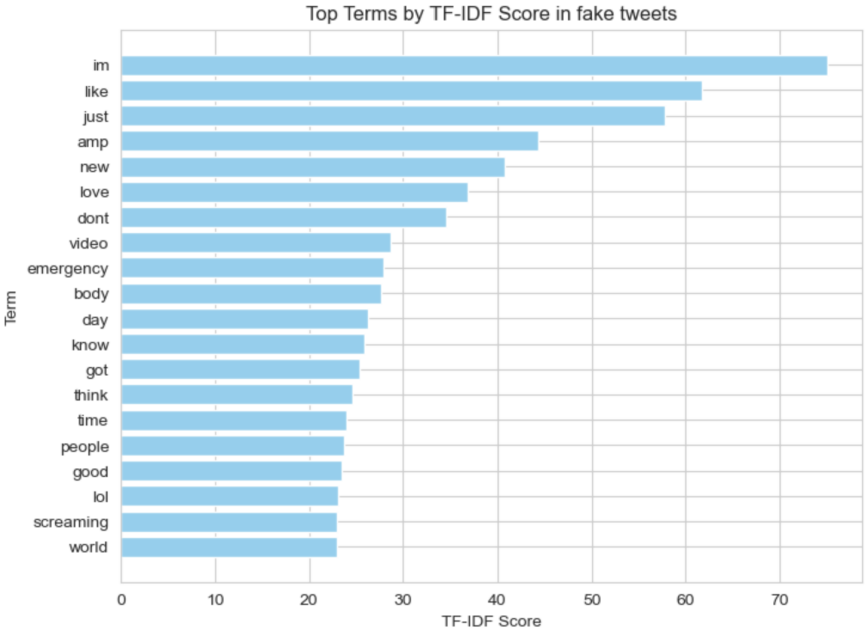

Figure 8: Frequent words in non-disaster Tweets

The results from token visualisation and Frequency score were a little different. We were also

interested in a relationship between 'God-like' words ('Jezus','God','allah' etc) vs target. The plot below shows that in summary more words like that are present in casual Tweets.

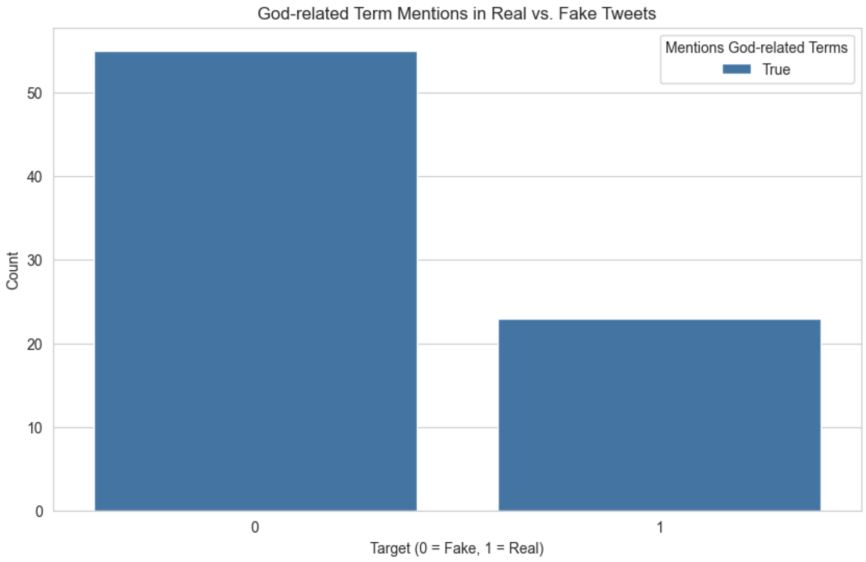

Figure 9: 'God-like' words vs target

2.2 Feature engineering

In this part our goal was to prepare the text to the form that can be processed by machine learning models. To achieve that we took the following steps:

Text cleaning
We started by removing links,mentions,punctuation and hashtags. Furthermore, in the previous step we didn't observe a high correlation between words in Caps Lock and the target, so we also transformed every word to lowercase.

Tokenization
Tokenization is a technique that extracts single words from a whole sentence. This step was necessary for further text simplification.

Lemmatization
Lemmatization is a process of transforming a word to it's base form. For example 'singing' or 'sings"will result in 'sing'.

Removing Stopwords
As we mentioned before stopwords are considered 'noise' in machine learning algorithms, so we detected and removed them.

Words vectorization
The final step before approaching modeling is to convert words into vectors, allowing a model can process them

2.3 Feature selection

In previous parts we have created many features to discover new patterns. This columns however can be a 'distraction' for our model, so the next step is features selection. How relevant are other columns beside 'text'?

We used 4 feature importance methods listed below to create 3 different sets of data for later model building:

- **Correlation Matrix**
  We checked the correlations between different features and the target. We noticed that some of the new features have low correlation with the target, but high among each other so we will remove some of them to avoid redundancy.

- **Recursive Feature Elimination**
  We checked feature importance for simple LogisticRegression model. This method takes all features from training dataframe, and then in each iteration drops the least relevant one. We visualised the most important features on the plot:

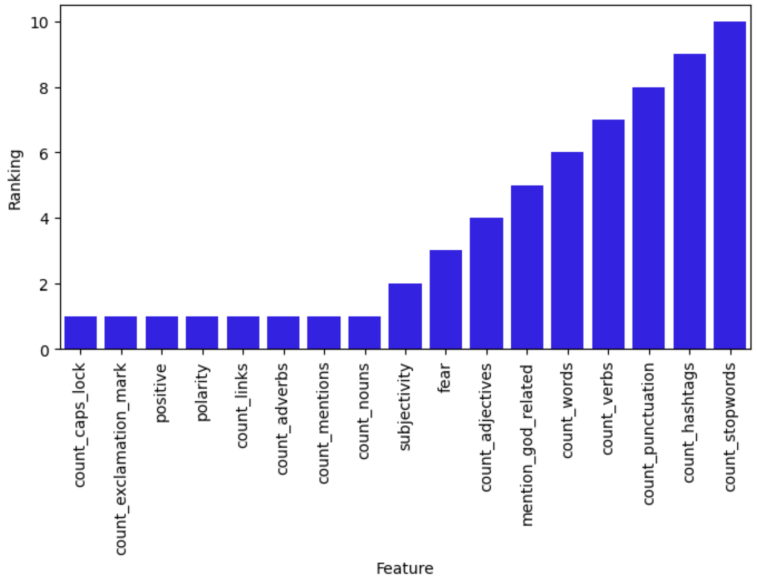

Figure 10: Recursive Elimination feature ranking

- **Random Forest Feature Importance**
  Another method we used was to count Random Forest feature importance scores. This is one of the embedded methods, that checks how much improvement does every feature add to the model. Scores for 10 most important features are present on the plot:

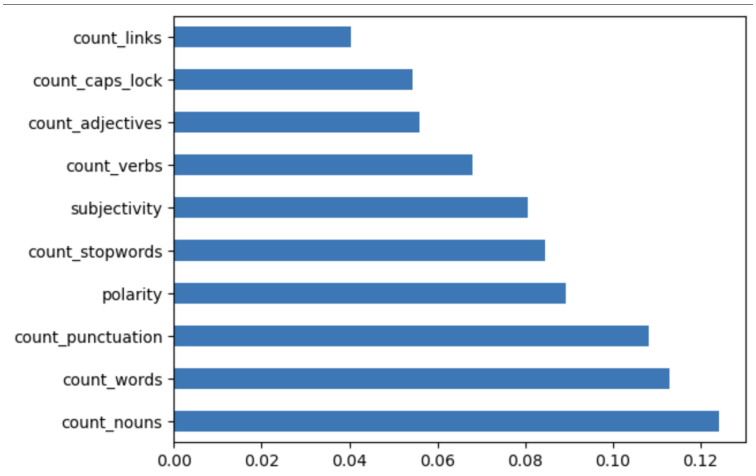

Figure 11: Random Forest feature importance scores

- **SelectKBest feature extraction**
  Lastly we used SelectKBest with k parameter set to 5, to select 5 most relevant features using mutual information. Mutual information is a numeric value that gives us a sense of how 'close' two variables are based on calculating joined probabilities. Features that were selected are : word count, punctuation count, link count, noun count and polarity.

We then created 3 transformers to select different feature subsets.

2.4 Model building

The final part of the project was to build a few models and select the best one. Our idea was to use different datasets from previous step, as well as check the performance for different word-vectorization methods. We focused on F1 score that is calculated as:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

where precision and recall are respectively:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

TP corresponds to True-Positive predictions (model correctly classified 1 as 1) and FN to False-Negative (model incorrecly classified 1 as 0).

Firslty we built our models using only vectorized text. Best F1 scores for different model building methods are present in a table below. Keep in mind that in each execution of this code values can be slightly different as we used Randomized Search to determine best parameters. Then we checked if adding new generated features will imporove the model's performance. F1 score we got

Vectorizer	Model	F1 score
Count	SVM	0.796
Count	Logistic Regression	0.796
Count	MultinomialNB	0.795
TFidf	SVM	0.795
TFidf	Logistic Regression	0.793

Table 3: Best F1 scores for different model-building methods.

after adding new features was circa **0.75** for SVM model with best parameters and Count Vectorizer. The score slightly dropped which can be a result of Randomized Search, but also the fact that new features can be acting as a 'noise' for our model. As the performance didn't improve it is best to keep the model simple. Therefore we decided to use **only the 'prcessed_text_str' column**, which is the cleaned and tokenized text, that we changed to string.

2.5 Parameter tuning with Optuna

We tried to improve final score by implementing optuna, as the parameter tuner, as it is much better than randomized search. Results that we got were slightly better than the ones we have recieved before. We decided to keep this for our final model.

3 Model Architecture and final results

According to the table above results were best for Count vectorizer. How does it work? Let's look at the example dataframe below:

1.	'Huge fire on Mainstreet'
2.	'This photo is fire'

This vectorizer will create a matrix where every column is an unique word and then every row corresponds to a row in a dataframe. Then it counts how many times each word appeared in a given Tweet like the example below:

	huge	**fire**	**on**	**Mainstreet**	**this**	**photo**	**is**
1.	1	1	1	1	0	0	0
2.	0	1	0	0	1	1	1

Table 4: Count vectorizer matrix

After word vectorization it was time to select the final model. We decided to pick SVC model, which is part of the Support Vector Machine (SVM) family that is known for its robustness in dealing with high-dimensional data and complex decision boundaries. Here is the detailed explanation of the chosen method:

- **Objective** The main goal of SVC is to find a hyperplane that best separates the data points of different classes in a high-dimensional space.
- **Hyperplane** A hyperplane in an n-dimensional space (where n is the number of features) is a flat affine subspace of dimension $n - 1$. For instance, in a 2D space, a hyperplane is a line; in a 3D space, it is a plane.

- **Margin** SVC aims to maximize the margin, which is the distance between the hyperplane and the nearest data points of any class. These nearest data points are called support vectors.
- **Formulation** The optimization problem can be formulated as:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, \quad \forall i$$

  where, w is the normal vector to the hyperplane, b is the bias term, y_i is the class label of the ith data point, and x_i is the feature vector of the ith data point.
- **Soft Margin** In real-world data, perfect separation may not be possible due to noise and overlapping classes. SVC introduces slack variables ξ_i to allow some misclassifications:

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i$$

  C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error.
- **Kernel Trick** SVC can be extended to handle non-linear decision boundaries using the kernel trick. A kernel function $K(x_i, x_j)$ transforms the input space into a higher-dimensional space where a linear separation is possible. Common kernels include:
  1. Linear Kernel
  $$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$
  2. Polynomial Kernel
  $$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d$$
  3. Radial Basis Function (RBF) Kernel
  $$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$$
  4. Sigmoid Kernel
  $$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\alpha\mathbf{x}_i \cdot \mathbf{x}_j + c)$$

  In our final model, we have selected the sigmoid kernel.
- **Training and Prediction** During training, the SVC algorithm solves the optimization problem to find the optimal w and b. For a new data point x, the class label is predicted using:

$$sign(wx + b)$$

3.1 Parameter tuning with Optuna

We decided to increase the final score of our model by implementing optuna as the parameter tuner, as it is much better than randomized search. Results that we got were slightly better than the ones we have recieved before.

3.2 Kaggle Submission

Now that we have decided to use CountVectorizer and SVC, we needed to transform the test data and predict the values. Predictions were saved as a csv file and later we submitted it on Kaggle. **The final result was: 0.79313 F1-Score.** This placed us in the top 500s on the Leaderboard.

4 Interpretation

The most important step after building a model is interpretation. Which words affected the prediction? In what way? To answear those questions we used 'shap' package for model interpretability. We generated waterfall plots for both positive and negative prediction. Results and their interpretation are present on the figures below.

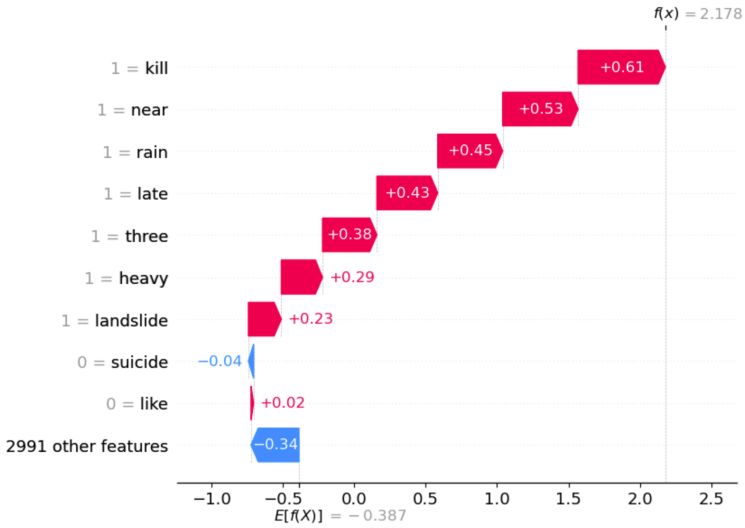

Figure 12: shap waterfall plot for positive prediction

On the plot above (Firgure 12) we can see a Tweet that was classified as informing about a real danger. Values 0 and 1 on the left side inform how many times a particular word appeared in a given Tweet. In this example we have words like : kill, near, rain, late, three, heavy, landslide that are the most important and shift the probability of a Tweet being a disaster Tweet.

Now let's examine the Tweet that was labeled as just a casual tweet.

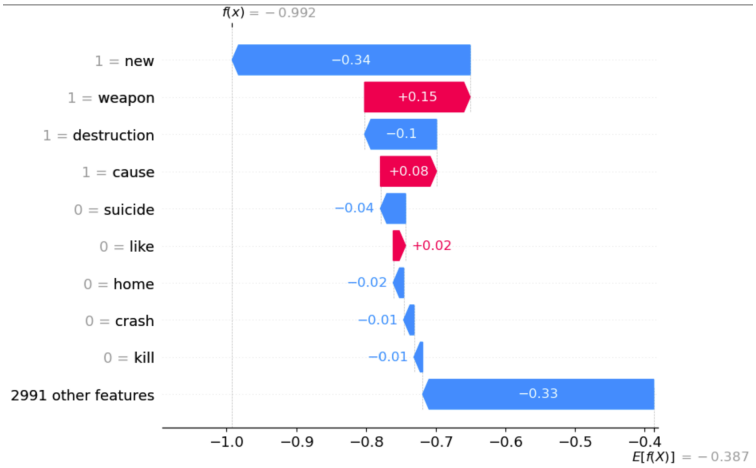

Figure 13: shap waterfall plot for negative prediction

In the Tweet that is present on the figure above there were words like new, weapon, distruction, cause. The appearance of the words 'weapon' and 'cause' shifted the prediction towards the positive one, but the word 'new' was the most important and shifted the probability towards the casual Tweet. Why? As we looked closer to the Tweet's text it said : "So you have a new weapon that can cause unimaginable distruction". As we can see it doesn't inform about any real time disaster.

5 Conclusion

Compared to other papers written about this particular topic, we are satisfied with the model's final performance. What could be improved? For sure, we could try more models and different parameter tuning methods. Another interesting aspect is the impact of newly generated features on the model. In our minds, there is still a question: could we have generated a feature that would improve the score? For now, this question remains unanswered.

6 References

As we explored various nlp methods we did some reaserch that helped us during the developement process. Some of the websites we used are listed below.

1. KerasNLP Getting Started Guide
2. KerasNLP Getting Started Notebook
3. Support Vector Machine
4. Full NLP Guide