EPL448: Data Mining on the Web Final Project

Real or Not? NLP with Disaster Tweets

Predict which Tweets are about real disasters and which ones are not

Marcos Antonios Charalambous, Sotiris Loizidis

University of Cyprus

Department of Computer Science

Nicosia, Cyprus

mchara01@cs.ucy.ac.ucy, sloizi02@cs.ucy.ac.ucy

Contents

Abstract -- In the age of social media, posting about a disaster or pretty much anything that happens in our everyday lives has become the social norm. Considering the massive amount of information, one must determine whether a tweet about a disaster is real or not in order to act (Government bodies trying to battle disasters), or just be correctly informed (avoid misinformation and its spread). We have developed a machine learning model and applied Natural Language Processing techniques in order to tackle this problem. Our report shows the procedure we followed in order to be able to produce a model capable of confidently predicting if a tweet is real or not with an F1 score of 81.492% on the official competition leader board and the small discoveries and difficulties we encountered during this process.

Introduction

This report is based on the work completed for the **"Real or Not? NLP with Disaster Tweets. Predict which Tweets are about real disasters and which ones are not."** which is an open competition on Kaggle. Since this is a brief report, further documentation can be referenced in the appendix and output files.

After a lot of research and careful studying of varying notebooks on Natural Language Processing we structured our course of action into dataset analysis using various Exploratory Data Analysis (EDA) techniques, data pre-processing specialized for tweets, model evaluation and subsequently a selection, parameter tuning for a specific model that yields best results and stands out from the rest, k-cross validation and ultimately, prediction on a never seen before test set using metrics such as precision, recall, F1 score and accuracy score.

Some of our most helpful sources are listed on the appendix but to name a few approaches and techniques that helped us a lot was Sentiment Analysis, Natural Language Processing, Exploratory Data Analysis, Data pre-processing for tweets, various Word Embeddings for turning tweets to vectors, pre-trained word embeddings such as GloVe, Classification Algorithms, Parameter Tuning, Dimensionality Reduction, Feature selection.

Our Goals

During the development of this project our goal was to familiarize with popular machine learning techniques in order to mine meaningful data from an unprocessed dataset. This process was a conduit in which we could apply many of the methodologies included in our curriculum and see in practise why certain techniques work and others do not.

Performance-wise, our goal was to achieve a confident accuracy improvement from a baseline null accuracy. What we define as null accuracy is basically determining which class is dominant (disaster or no-disaster) and then simply always predicting that a tweet will be the same class as the dominant class. For example, if we see that 7000 out of 10000 tweets are disasters, we predict that all test tweets are disasters and then proceed to measure the accuracy of our "prediction". This establishes a baseline accuracy. We were consistently above this baseline across all executions.

In addition, considering that some of the highest performing entries in the competition in Kaggle achieved a score F1 of about 85%, which was measured in the presented in the figure below, we tried as much as possible to explore extensively as many techniques as possible in our limited time frame and try to approach as much as possible that score.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

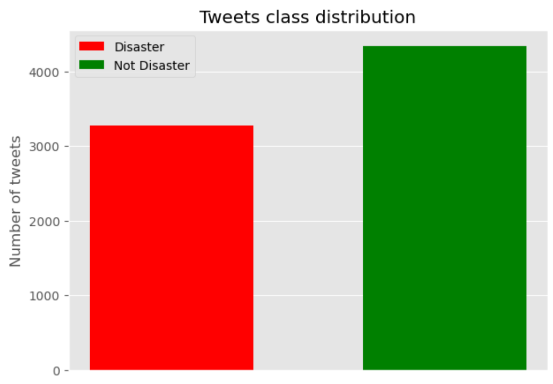
$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

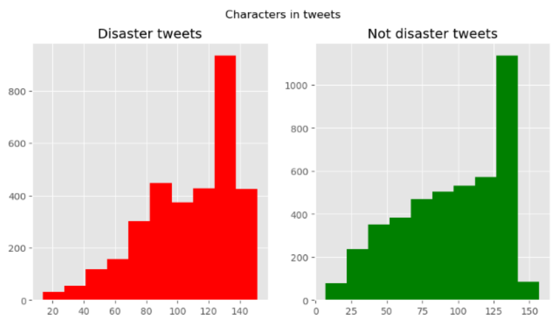
Exploratory Data Analysis (EDA)

After downloading the train.csv from the competition we wanted to inspect and analyse the data we were dealing with by generating simple statistics from tweets as well as plotting various essential information, previously in raw format, into a visual and more user-friendly way. An investigation with EDA would play a key role in how we will build our model. It is crucial to understand the properties of the subjects we are classifying.

Before we begin with anything else, we checked the class distribution of our data. There are only two classes in our data: 0 that corresponds to an irrelevant tweet and 1 that corresponds to a disaster tweet. In this situation our subjects are at an about 43/57 split in favour of irrelevant tweets. A simple but vital piece of information to remember when comparing visualizations between the two classes.

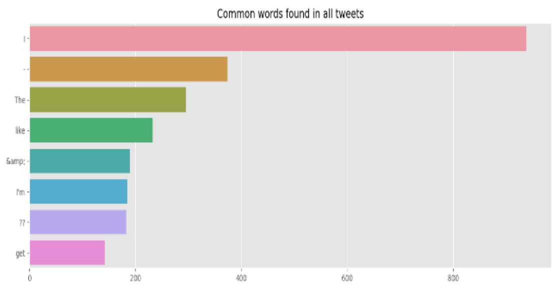


First, we start by doing a very basic analysis. That is: on character level, word level and sentence level analysis. On character level, the distribution of both seems to be almost the same. Around 120 to 140 characters in a tweet are the most common among both classes. Regarding words in a tweet, irrelevant tweets tend to have more words meaning that people are willing to be more talkative when writing non-disaster tweets. Analysis on sentence level is where we analyse the average word length in a tweet. By observing the produced plot of this we cannot extract any useful information as both classes of tweets have approximately the same average word length.



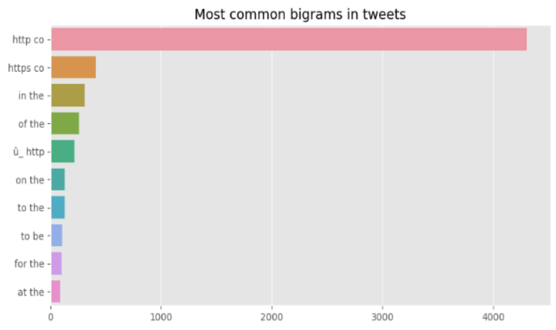
Moving on we search for common stop words in both classes of tweets. Across both, the word “the” dominates followed by “a” in class 0 and “in” in class 1. Also, we analysed the punctuations in both classes and found out that “-“ is the most frequent in both.

In addition, we plotted the common words between the two classes as you can see in the figure below.



The previous figure indicates that lots of cleaning will be needed in the pre-processing phase as some of the common words are punctuations, stop words and links which give no useful information about the class of the tweet.

We continue our EDA by doing a bigram analysis over the tweets to find the most common bigrams in the tweets found in our data.



As we can see, some of the common bigrams are links and combinations of stop words which give no information.

After this, we proceed to analyse the hashtags, to examine the possibility of them used as a possible discriminator for our task. We have observed that there is too much intersection between hashtags in positive and negative samples, meaning that a hashtag approach will not work that well.

We move on to examine the keyword and location columns. After manipulating these two columns and summarizing their main characteristics we understand that location cannot be used as it is too sparse and many keywords in negative tweets are also present in positive ones, meaning that a keyword approach is most likely to not work.

Data pre-processing phase

There are usually multiple steps involved in cleaning and pre-processing textual data. After extensive experimenting on our data we have selected a few cleaning functions and eliminated others that have proven unable to change the efficiency of our models on a later stage or even worsen it. However, in this section, we highlight some of the most important functions which are used heavily in NLP, especially when it comes to tweets.

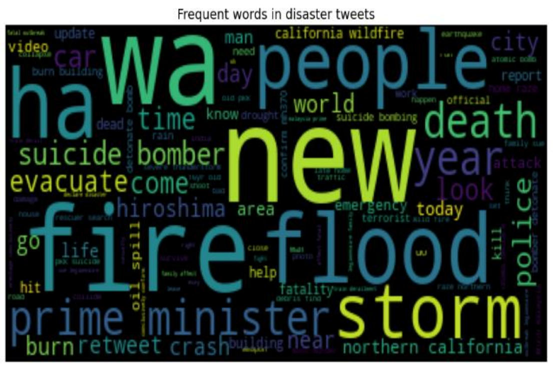
- **Stripping HTML tags:** HTML tags are typically one of these components which do not add much value towards understanding and analysing text. HTML encoding may not be converted to text and end up in the text field as

"&amp". We used the BeautifulSoup module for this.

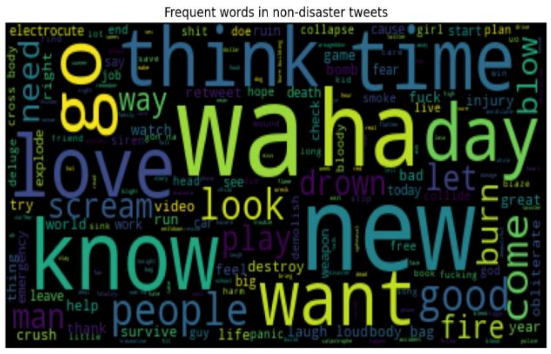
- **Unescape Tweets:** Unescaping any escaped characters found in tweets.
- **Removing accented characters:** We make sure that characters are converted and standardized into ASCII characters. A simple example is converting é to e.
- **Expanding Contractions:** Contractions are shortened versions of words. These contractions of words are created by removing specific letters. They are often created by removing one of the vowels from the word. A simple example is "do not" to "don't". Converting a contraction to its original form helps with text standardization.
- **Replacing special characters:** Here we replace the characters that will make the job of tweet cleaning a lot harder with another character or an empty string. An example is replacing the new line character with a space.
- **Hashtag expansion:** With the help of the ekphrasis module, a text processing tool geared towards text from social networks, such as Twitter, which can be found in the reference section, we use its word segmentation for splitting hashtags and expand words used in a hashtag which can provide valuable information about the class of the tweet.
- **Remove mentions:** Remove all mentions in a tweet that start with the character @. This information does not add value to build a disaster classification model.
- **Removing URLs:** Dealing with URL links, same with @mention, even though it carries some information, for the disaster classification analysis purpose, this can be ignored.
- **Converting abbreviations:** Here we collected manually some abbreviations from the tweets and saved them in the file abbreviation.py which we use to convert any abbreviations found in tweets to its expanded form. For example, "brb" will be expanded to "be right back".
- **Remove emojis:** Remove any emojis found in a tweet.
- **Remove stop words:** Words which have little or no significance, especially when constructing meaningful features from text, are known as stop words. Here all the stop words found, such as "the", "for" and "at" will be removed. This function has been set to false during the cleaning phase as results gathered at a later stage of the project shown that **removing stop words worsened the performance.**
- **Lemmatize:** Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. **Thus, we opted not to apply stemming on our tweets.** In Lemmatization root word is called Lemma. Replace words in tweets with their lemmas. For this function, we use the module twokenize which is a twitter specialized tokenizer.
- **Remove punctuation.** Remove any punctuation left as we do not gain any information out of them.
- **Remove whitespaces:** Last part of the tweet cleaning process is to remove both trailing and leading whitespaces or more than one continued spaces left during the previous steps of the process.
- **Lowercase:** Covert all the tweets to lowercase for easy of processing.

For this phase we use the function tweet cleaner which can be found in our code, to pre-process text with default option set to true for most steps shown above. You can see that in this function, besides the above steps, we use some functionality provided by another great module named spaCy, used for advanced natural language processing, to remove any stop words that have not been detected and also remove numbers that are either in their numerical or text form.

For the text visualisation we use world cloud. A word cloud represents word usage in a document by resizing individual words proportionally to its frequency, and then presenting them in random arrangement. Textual analysis is the most important analysis in NLP projects, and word cloud provides a general idea of what kind of words are frequent in the corpus and figure out what other methods can be used for further tweet cleaning. A word cloud produced from our data post data cleaning is the following:



Above we can view some frequent words from tweets in the disaster category such as "fire", "flood", "death", "evacuate" and "storm" which can indicate a disaster. Some of the big words can be interpreted quite neutral, such as "people", "world", "retweet" and "new". We repeat this for the opposite category.



Another thing worth mentioning is that we have also fixed some tweets that have been wrongly classified in the train.csv after gathering them from various other participants notebooks.

Model Selection

Problem understanding

Our dataset is a file containing approximately seven and a half thousand tweets which are structured in four columns: id, keyword, location, text and an additional column, which labels the corresponding tweet as a disaster and as irrelevant, named target. Our plan is to treat the problem as one of classification, that means determining the class of the tweet. This led us to classification algorithms. Since our sample size is quite small, we moved on to try models such as Linear SVC with not particularly encouraging results. We then moved to Naïve Bayes which is a family of models which famously work with a small amount of data and are used in real-world situations namely document classification and spam filtering.

Milestone

We reached a milestone when our test results showed great promise with different versions of Naïve Bayes such as Multinomial, Bernoulli or Complement, in our cross-validation phase with an average accuracy approximately 73%. This family of algorithms work by taking strong assumptions about the importance of discrete features. Since our features are discrete unigrams or bigrams or both, we indeed managed to get satisfying results. Multinomial is good for features with integer counts which is why it performs so well with countVectorizer and in practise with TF-IDF as well. Bernoulli also performs great since its only difference is that it is designed for binary features, but both seem to perform well with or without TF-IDF vectors which was interesting. Complement is just a more normalized version of Multinomial meaning that it more careful with edgy assumptions unlike Multinomial, but the results were comparable.

```
PREDICTION SCORES WITH TF-IDF
----------------------------
LogisticRegression: accuracy = 0.800, precision = 0.805, recall = 0.800, f1 = 0.797
RidgeClassifier: accuracy = 0.790, precision = 0.790, recall = 0.790, f1 = 0.788
SGDClassifier: accuracy = 0.787, precision = 0.787, recall = 0.787, f1 = 0.786
Linear SVC: accuracy = 0.778, precision = 0.777, recall = 0.778, f1 = 0.776
MultinomialNB: accuracy = 0.806, precision = 0.813, recall = 0.806, f1 = 0.803
ComplementNB: accuracy = 0.793, precision = 0.793, recall = 0.793, f1 = 0.792
BernoulliNB: accuracy = 0.806, precision = 0.817, recall = 0.806, f1 = 0.802
SVC: accuracy = 0.556, precision = 0.309, recall = 0.556, f1 = 0.397
PassiveAggressiveClassifier: accuracy = 0.739, precision = 0.739, recall = 0.739, f1 = 0.739
MLPClassifier: accuracy = 0.739, precision = 0.739, recall = 0.739, f1 = 0.739
RandomForestClassifier: accuracy = 0.784, precision = 0.791, recall = 0.784, f1 = 0.779
```

Exploring different models

In addition to naïve Bayes models we investigated voting models (ensemble methods) specifically the Random Forest classifier which was very time consuming and did not yield particularly promising accuracy. We also tried the SGD classifier just to lengthen our options but due to a small number of samples we could not achieve enough accuracy which was expected considering that the SGD classifier works well with over one hundred thousand samples.

Finally, we tried the simple Logistic Regression model which is a very simple linear approach and proved to yield strikingly similar results with the naïve Bayes models and in most of our metrics, beat it by a very small margin. This is logical considering linear regression is designed for binary classification and thus represents our dataset very well.

Final Model

We quickly realised the words (or combinations of words) contained in each tweet are a strong indicator of whether they're about a real disaster or not, for example features like "suicide bomber", "wildfire", "disaster" etc.

Obviously, a linear connection between these uni-grams or bi-grams and class exists and we decided to exploit that with the linear model of Logistic Regression but the choice between some variation of Bayes and Logistic Regression is very close.

Evaluation Methodology

All models were evaluated only after conducting K-cross-evaluation in order to confidently support our prediction for each model. We made sure that all predictions were unbiased by making use of several different assortments of the split function and using train sets, evaluation sets and test sets. We ultimately chose a 15% test set split after experimenting with 20% as well. Our dataset is quite small so we could make use of the extra 5% of samples for training.

Optimization techniques

Parameter tuning

Pipelining

Since choosing Logistic Regression as our go-to model we implemented a pipeline in which we will provide a range of different parameters for different points of processing also known as the Grid, either that be the vectorization of data with the use of Count Vectorizer or TF-IDF Vectorizer, the normalization technique of the TF-IDF when chosen, the n-gram range, and model regularization to avoid overfitting.

Checking different inputs

We provided the differentiation between using the column of text or different versions of columns where the keyword was appended to the text column, or the location column, or both. This shouldn't really affect performance, but it was one additional procedure we followed in order to consider cases were there was meaningful keywords, but the text would fail to showcase that. Evidently that was not the case and often we observed a decrease in scores rather than increase.

Exhaustive Parameter Search

A Grid Search was performed which basically is an exhaustive method for finding the best finely tuned parameters. We were able to successfully execute such demanding and brute calculations only because we did a lot of pre-processing and feature selection in order to reduce our features otherwise the computational time would be forbidding. It needs to be said that parameter tuning was rather disappointing since we managed no better results, but we tried it, nonetheless. We then performed predictions with the best parameters, provided by GridSearchCV.predict(X) which by default predicts with the best-found parameters for each of the inputs, and our classification report showed that usually the best scores were achieved using all three columns and TF-IDF along with Logistic Regression but our best submission was with the text column and default parameters. We suspect that this is because of some differentiation in the unseen test dataset.

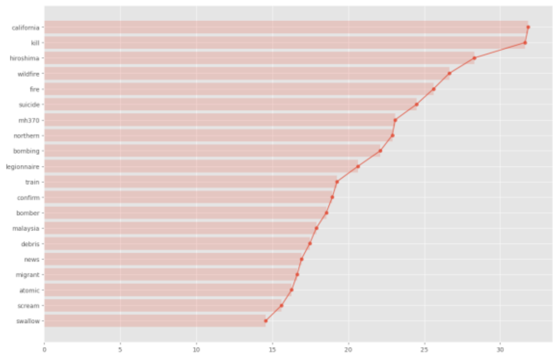
```
Results with the following columns: text

# Tuning hyper-parameters for f1

Best parameters set found on development set:

{'clf__C': 50, 'tfidf__norm': 'l1', 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}
```

Data Splitting Variations

It should be noted that many of these techniques such as overfitting control were bound to be mildly unsuccessful since we couldn't possibly over train our model with such a small dataset especially when we used 15% of our train dataset for predicting scores, that left us with less than six and a half thousand samples to work with.

Feature Selection

In this section we performed a series of different tests in order to improve our execution time and/or our model's accuracy. We got quite a bit of help from the Chi2 future selection model. This model was used in the TF-IDF vectors in order to rank features based on the chi-squared statistic. The higher the statistic the more depended the class is to that feature which means it has a more important role in the prediction of the class.



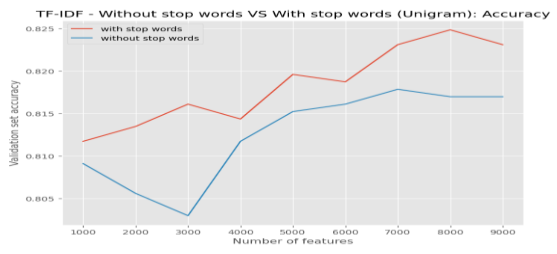
As we can see, words such as "California" or "kill" ranked very high with the chi2 metric and are probably strong indicators of the disasters class. As odd as it seems California ranked first but that was since most of these tweets were collected during the California Wildfires.

Vectors and n-grams

The first thing we experimented with was applying different n-gram ranges to both a countVectorizer and a TF-IDF Vector. We specifically tried unigrams, unigrams and bigrams and then unigrams, bigrams and trigrams together. This of course produces different features on each occasion, so we plotted the different features and corresponding accuracies for all different vectors. We found that TF-IDF vectors performed slightly better.

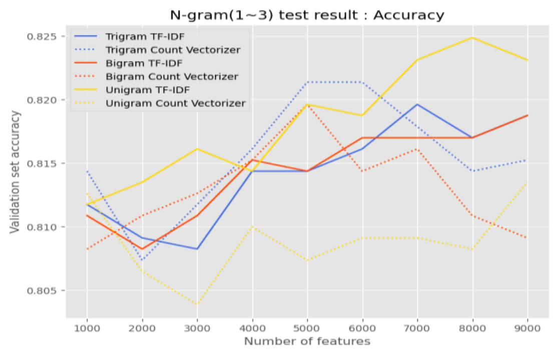
Stop-words

Stop-words played a major role in our analysis. In our observations we found out that many stop-words prove quite useful in accuracy improvement and their removal can hurt score performance. This can be confirmed in the figure below.

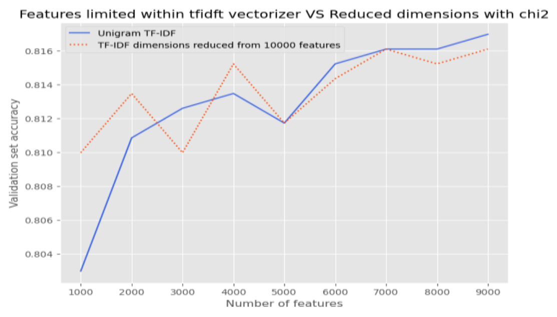


TF-IDF feature selection

We had to make a choice between TF-IDF and Count Vectorizer and find which n-gram range produces the best accuracy on both.



As we can see the combination of unigrams using TF-IDF yields the best results using the max number of features. To further experiment we wanted to compare the accuracy achieved by limiting the features on the TF-IDF vectorizer or by using feature selection with chi score. What we were trying to find was the best performance in accuracy and in the end TF-IDF vector along with unigrams while stating the max features upon initialization, won the race.

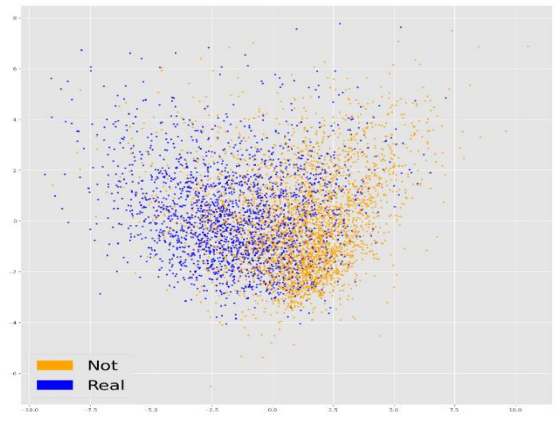


GloVe

A completely different approach that we tinkered with was pre-trained word vectors. More specifically we used GloVe. GloVe coined from Global Vectors, is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word statistics from a corpus. Basically, a vast set of tweets was iterated and each time a pair of words occurred in a tweet the corresponding cell in the co-occurrence matrix was updated. The word vectors returned is the object matrix of the GloVe model.

We found this very interesting and decided that we should give it a try. They have four different versions of Tweet vectors each with different dimensions (25, 50, 100, 200) trained on 2 billion Tweets. During our experiments we were limited to trying only the 25- and 50-dimension vectors as the bigger ones had huge memory demands that where not available to us and needed a lot more time and resources to perform their necessary calculations. The results that will be shown are obtained with 50 dimensions pre-trained GloVe vectors. The Gensim module has made loading and the use of the pre-trained GloVe vectors easy. A link can be found in the reference section

As we already mentioned, GloVe is a model that was pre-trained on a very large tweets corpus and provides embeddings that map words that are semantically similar, close to each other. A quick way to get a sentence embedding for our classifier, is to sum the scores of all words in our sentence. After creating these embeddings, we visualize them using LSA and see if we can identify some structure. In a perfect world, our embeddings would be so distinct that the two classes would be perfectly separated. Since visualizing data into thousands of dimensions is hard, we project it down to the two classes.

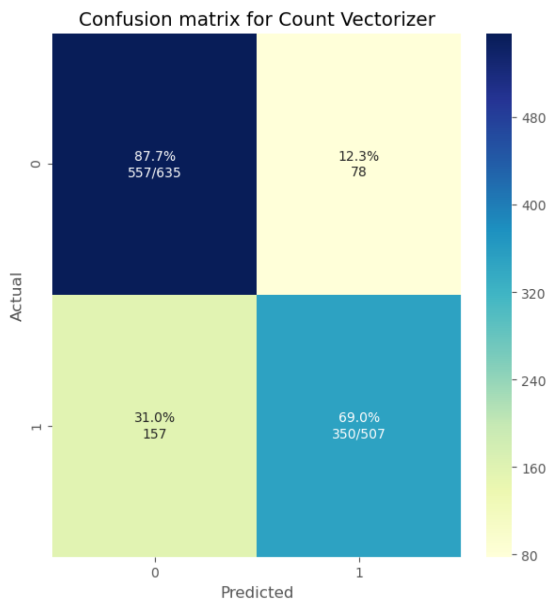


As we can observe from the figure above, the two classes look promisingly separated. After running the Logistic Regression on our word embeddings, we saw that the prediction accuracy and F1 score are slightly improved. So far, the best prediction accuracy from our train set split with TF-IDF vectors gave us approximately 80% accuracy. GloVe vectors yield 80.7% for sum. In the final submission our own model performed slightly better than GloVe.

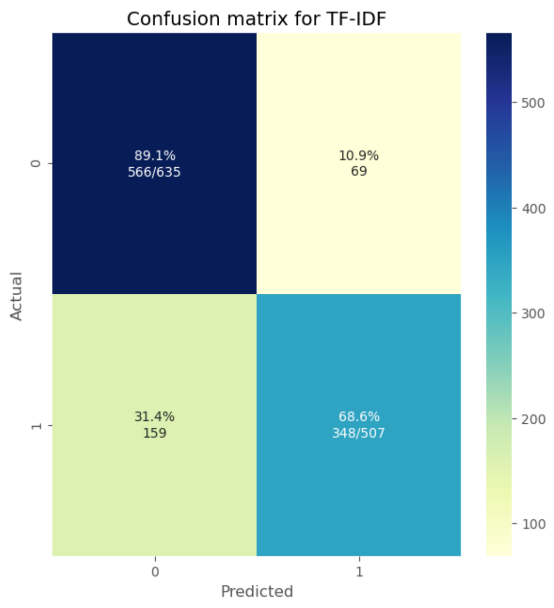
References

- https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html
- https://spacy.io/usage/linguistic-features
- https://www.kdnuggets.com/2018/08/practitioners-guide-processing-understanding-text-2.html
- https://gist.github.com/jiahao87/d57a2535c2ed7315390920ea9296d79f
- https://github.com/cbaziotis/ekphrasis
- https://pydoc.net/ekphrasis/0.4.7/ekphrasis.classes.preprocessor/
- https://towardsdatascience.com/another-twitter-sentiment-analysis-bb5b01ebad90
- https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-2-333514854913
- https://www.kaggle.com/rftexas/text-only-bert-keras
- https://www.datacamp.com/community/tutorials/stemming-lemmatization-python
- http://www.cs.cmu.edu/~ark/TweetNLP/#pos
- https://spacy.io/usage/linguistic-features
- https://en.wikipedia.org/wiki/GloVe_(machine_learning)
- https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-10-neural-network-with-a6441269aa3c
- https://github.com/RaRe-Technologies/gensim-data
- https://rare-technologies.com/new-download-api-for-pretrained-nlp-models-and-datasets-in-gensim/
- https://en.wikipedia.org/wiki/Latent_semantic_analysis
- https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825
- https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79
- https://github.com/sismetanin/sentiment-analysis-of-tweets-in-russian/blob/master/Sentiment%20Analysis%20of%20Tweets%20in%20Russian%20using%20Multinomial%20Naive%20Bayes.ipynb
- https://www.kaggle.com/philculliton/nlp-getting-started-tutorial?fbclid=IwAR0yzaayW3s14go01j0-e74gyLXa9ny_DXwvZNEl_POnxjSW0D9yRa13kK0
- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB
- https://developers.google.com/machine-learning/crash-course/framing/ml-terminology

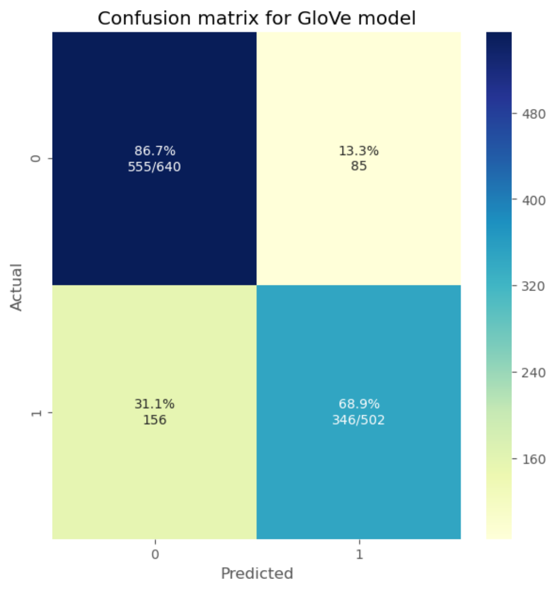
APPENDIX



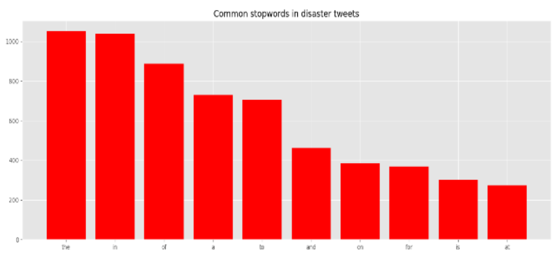
Confusion matrix for Count Vectorizer



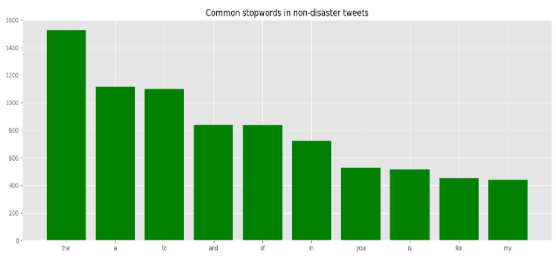
Confusion matrix for TF-IDF



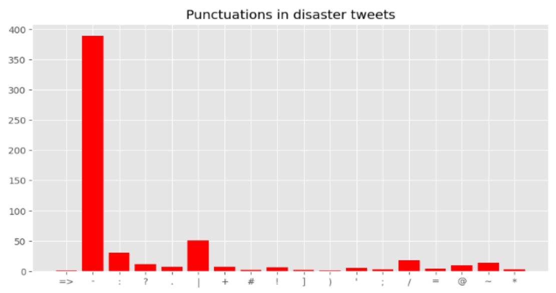
Confusion matrix for GloVe



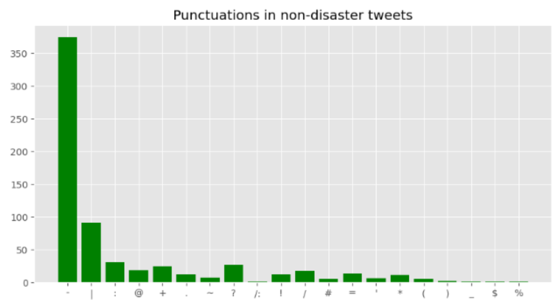
Stop Words in disaster tweets



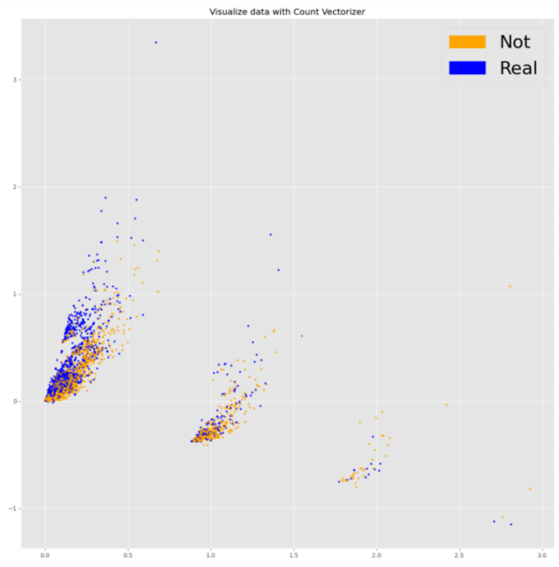
Stop Words in irrelevant tweets



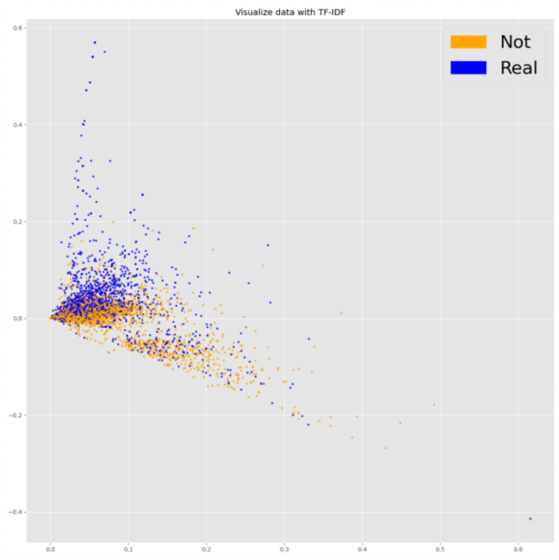
Punctuations in disaster tweets



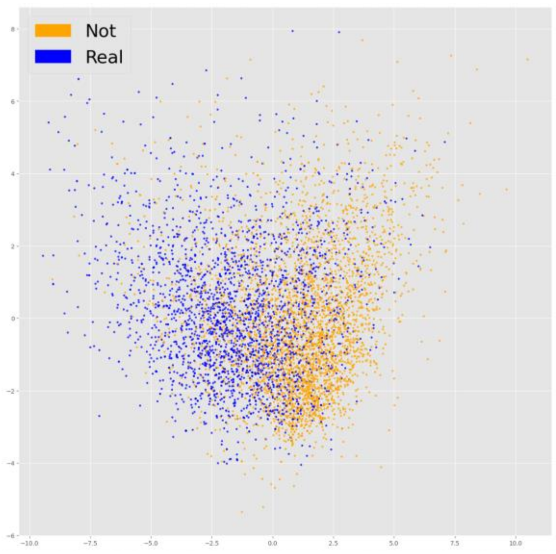
Punctuations in irrelevant tweets



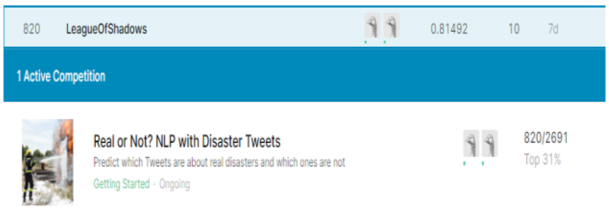
LSA with CountVectorizer



LSA with TF-IDF



LSA with GloVe

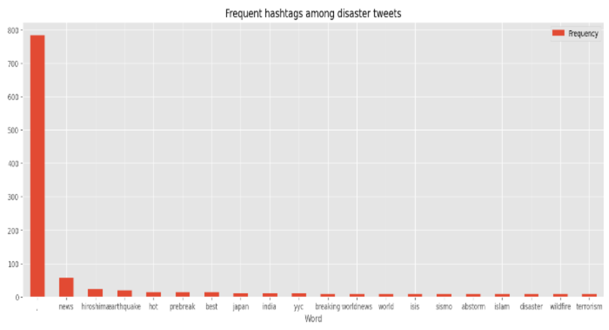


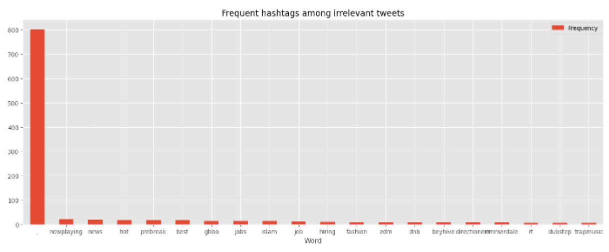
820 LeagueOfShadows 0.81492 10 7d

1 Active Competition

Real or Not? NLP with Disaster Tweets
Predict which Tweets are about real disasters and which ones are not
Getting Started · Ongoing
820/2691
Top 31%

Competition Results





Too much intersection between the two to be used as a possible discriminator for the two classes.