Natural Language Processing for Prediction of Disaster Tweets using Machine Learning Methods

Technical Report · June 2021

DOI: 10.13140/RG.2.2.20115.20005

2 authors:

Samane Sharifi Monfared
Bahçeşehir University
15 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Bilal Sedef
Borusan Otomotiv
5 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Natural Language Processing for Prediction of Disaster Tweets using Machine Learning Methods

Samane Sharifi Monfared*, Bilal Sedef*

**Bahcesehir University, Besiktas, Istanbul and 34349, Turkey*

Abstract

Nowadays, with the increase in digitalization, the use of social media by people who have access to the internet has become very common. The use of social media at such a high rate has both accelerated the interaction between people and greatly increased the spread of social events. It has been inevitable for social scientists and computer scientists to benefit from this widespread use and speed of information. Since the texts shared by people on social media can be categorized according to the emotional expressions they contain, they can be easily used to describe and predict events thanks to computer science. This estimation process can be performed by NLP (Natural Language Processing) techniques. In this article, we analyzed tweets containing keywords related to disasters with classification models, using various pre-processing techniques and TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction method. We compared these classification algorithms (SVC, MultinomialNB, LogisticR, XGBClassifier, RandomForest, DecisionTreeClassifier, KNeighborsClas) according to their f1 scores, train accuracy and test accuracy.

Keywords: NLP; TF-IDF; Classification; Twitter

1. Introduction

Although comments on many topics are seen on social media, Twitter is a platform where opinions can be obtained on almost every subject. Sources indicate that Twitter has 336 million monthly active users [1], with around 500 million tweets being posted every day. This means a huge amount of text, uncategorized. This large amount of information is useful source for importing news in case of emergency. In this study on Twitter, we will analyze whether there is a real disaster or is it fake news from tweets containing raw texts

* Corresponding author. Tel.: +98-913-261-3715 / Tel.: +90-505-615-2277
E-mail address: samane.monfared@bahcesehir.edu.tr / bilal.sedef@bahcesehir.edu.tr

about disasters that concern society. The techniques we will focus on in this article are just some of the methods developed on text mining. Text mining or text classification is a set of algorithms that involves interpreting what the content is about.

In this study, which analyzes whether an event is really a "disaster" from raw texts, the Kaggle Twitter Dataset was used. In order to analyze a text with NLP techniques, it is necessary to apply pre-processing methods first. These pre-processing methods make raw text suitable for applying classification models. Here, pre-processing techniques such as Tokenization, Normalization, Stemming, Lemmezation, Stop word Removal, Noise removal have been used to adapt the data to the models. After applying these techniques, feature extraction was used on the data. In this study, TF-IDF was used as feature extraction method. There are many feature extraction methods other than TF-IDF. Finally, machine learning algorithms such as Support Vector Machine, Logistic Regression, Naive Bayes, Random Forest, Decision Tree, and K-Nearest Neighbor were applied on this noise-free and ranked data.

Nomenclature	
D	Document Collection
w	A word
d	Individual document
$f_{w,d}$	The number of times w appears in d
$\|D\|$	The size of the corpus
$f_{w,D}$	The number of documents in which w appears in D

2. Materials And Methods

In this data set, as can be seen in the image below, various methods were applied in a certain order. You can see information about these methods and their sorting logic below.

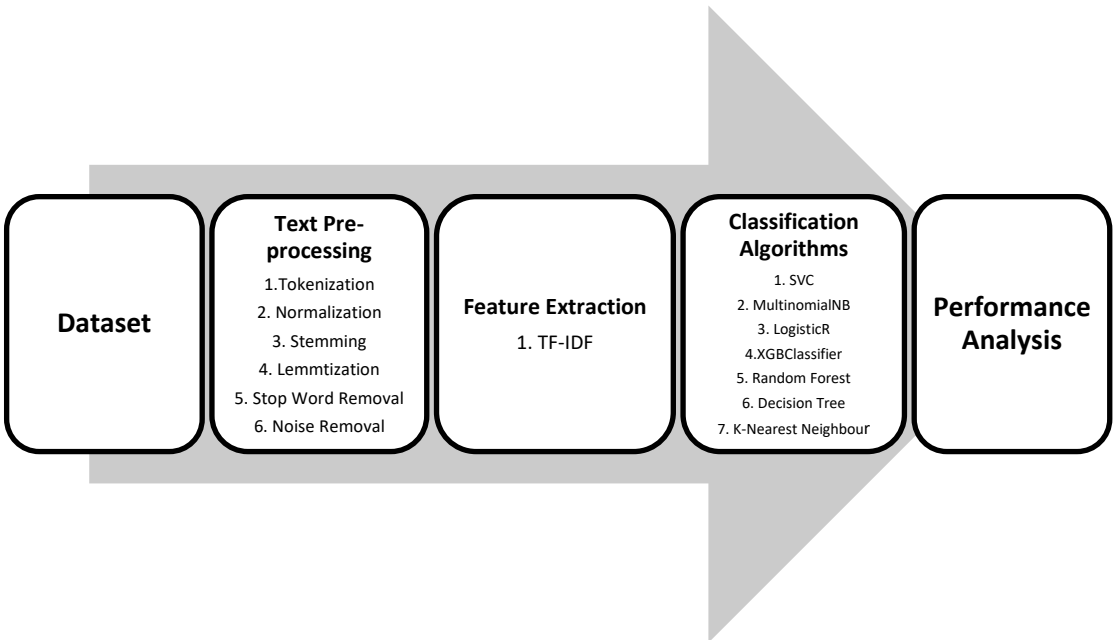

Fig. 1. NLP diagram in this paper

2.1 Dataset

Disaster related tweets dataset is supplied by Kaggle [2]. There are 7613 tweets and 5 columns. In this dataset there are 4342 tweets not related to disaster and 3271 tweets related to disaster.

2.2 Text Pre-processing

2.2.1 Tokenization

The Tokenization is the process of making large sentences into small sentences and into small words. As an example we can mention the sentence of "I have a hat". After Tokenization we would have "I", "have", "a", "hat"

2.2.2 Normalization

The Normalization is the process of applying many methods to make the data uniform such as removing noise, punctuation, links, etc.

2.2.3 Stemming

The Stemming is the process of replacing all tenses of a word with their base to decrease the chance of over processing for words with same tenses.

	original_word	stemmed_words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

Fig. 2. Stemming [3]

2.2.4 Lemmatization

The lemmatization is used to assign the words with same meaning to a unique word. It also conclude the process of removing the end of the words or making them into their normal form, like came to come.

	original_word	lemmatized_word
0	goose	goose
1	geese	goose

Fig. 3. Lemmatization [3]

2.2.5 Stop Word Removal

The Stop words are the most used words in English language like am, is, etc. Removing this words are a good way to increase the accuracy of our sentimental analysis.

2.2.6 Noise Removal

Our dataset is consisting of raw data. We had applied many noise removal operations on this data to have a clean and accurate dataset to work with.

2.3 Feature Extraction

2.3.1 TF-IDF

TF–IDF (term frequency-reverse document frequency) is the method used to represent texts as vectors (for classification, clustering, visualization, etc.). In the text in which it is used, it weights the terms according to the frequency of use. In this way, terms are ranked according to their importance and healthier results are obtained when analysing. At the same time, with TF-IDF, the frequency of usage of these terms between documents is checked and if it appears in many documents, the weight of the term is penalized. If we think the opposite, if the term occurs specifically for some documents, the importance and weight given to the term will be increased. [4]

The mathematical framework of TF-IDF is as follows [5]. Given a document collection D, a word w, and an individual document d ϵ D, we calculate

$$\mathrm{wd} = \mathrm{fw,d} * \log(|\mathrm{D}|/\mathrm{fw,D}) \quad (1)$$

where fw,d equals the number of times w appears in d, |D| is the size of the corpus, and fw,D equals the number of documents in which w appears in D (Salton & Buckley, 1988, Berger, et al, 2000).

2.3 Classification Algorithms

2.3.1 SVC

Support Vector Machine Classifier is one of the most useful classifiers in this project due to the fact that this classifier put a nice and accurate hyperplane between two target values of our prediction. [6]

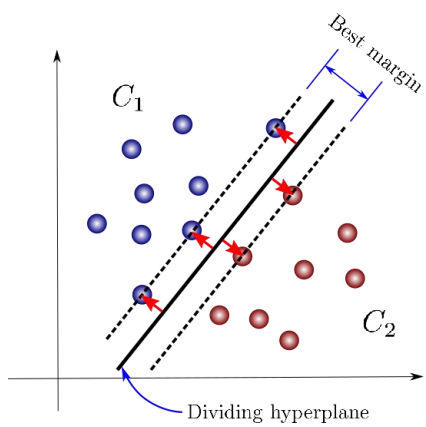

Fig. 4. SVM Classifier [11]

2.3.2 Multinomial NB

The Naive Bayes algorithm is an accurate and powerful algorithm based on probability. It works based on Bayes theorem which the prediction is the class with highest probability. Therefore, it can have a good performance in case of small datasets just like ours. [7]

2.3.3 LogisticR

The Logistic Regression algorithm is based on linear regression method which predict a model based on entropy whenever it is maximized.

2.3.4 XGBClassifier

The XGBoost classification method is based on a decision tree which uses a gradient boosting as a framework. According to the researches, this algorithm has a good performance in unstructured datasets like disaster tweets. [8]

2.3.5 Random Forest

The Random Forest algorithm is based on the decision tree algorithm. In this algorithm we select a random portion of the dataset and fit it to various decision trees and in the prediction we use the majority class.

2.3.6 Decision Tree

The decision tree algorithm is an amazing algorithm based on dividing the dataset into small trees. Thus, this algorithm can be useful in case of regression and classification.

2.3.7 K-Nearest Neighbour (KNN)

In the K-nearest neighbour algorithm as the name defines, we consider the majority class of K nearest neighbours of a sample point to classify it. This algorithm is one of the simplest algorithms in machine learning can be used in pattern recognition, sentimental analyses, etc.

2.4 Performance Analysis

2.4.1 Accuracy

Accuracy is simply ratio of correctly predicted observation to the total observations. Although it is a simple measure, it is quite effective for assessing the performance of the model. [9]

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

2.4.2 F-Score

F1 Score is the weighted average of Sensitivity and Recall. Since the F1 Score takes into account both false positives and false negatives, it is necessary to check the ratio of FP and FN in its use. F1 is often more useful than accuracy in an uneven distribution. [9]

$$F1\ Score = 2 * \frac{(Recall * Precision)}{Recall + Precision}$$

3. Dataset Description

Disaster tweets dataset is supplied by Kaggle. In the training dataset there are 7613 rows and 5 columns. These columns are: Id, keyword, location, text and target. In this dataset there are 4342 tweets not related to disaster and 3271 tweets related to disaster. In the training set, rows are labeled as 1 and 0 which is "Disaster Related" and "Non-Disaster Related". The detailed information about the training dataset can be seen below.

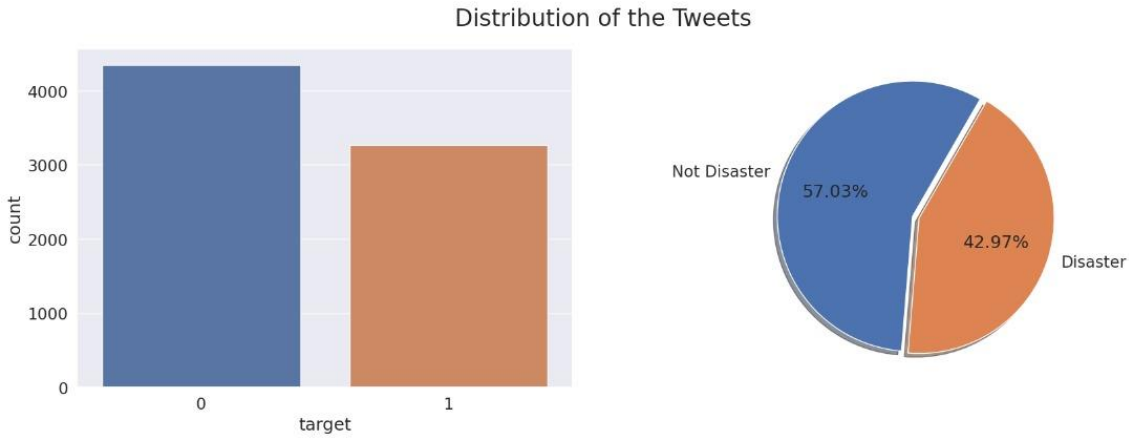

Fig. 5. Distribution of the training set tweets

Training set mostly consist of Not Disaster tweets. Still, due to the difference between different labeled texts are not significantly large, the models will train faster.

Before checking the text contents, we may look into the texts overall structure. Below, the text lengths of the tweets can be seen.

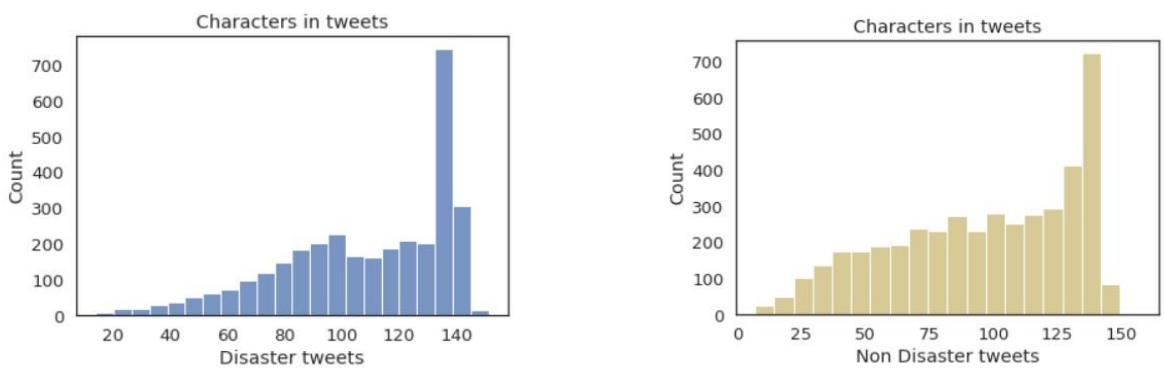

Fig. 6. Training set

Fig. 7. Test set

As can be seen in the Training set (Fig. 6) and the Test set (Fig. 7) texts are similarly distributed. This will give us clearer and more accurate results.

To use NLP methods to examine the texts, we should understand the contents that tweets consist. Firstly, we can check the most used keywords overall the data. In this way, before applying any method, we will see which words the computer will prioritize and the words that will affect our models when classifying. In such text mining problems, it is important to identify the most used words. However, not every large number of appeared words mean they are important. In order to manage this situation, it is necessary to make some applications before proceeding to the modeling phase. Before we move on to this step, let's check our training set for the most used words.

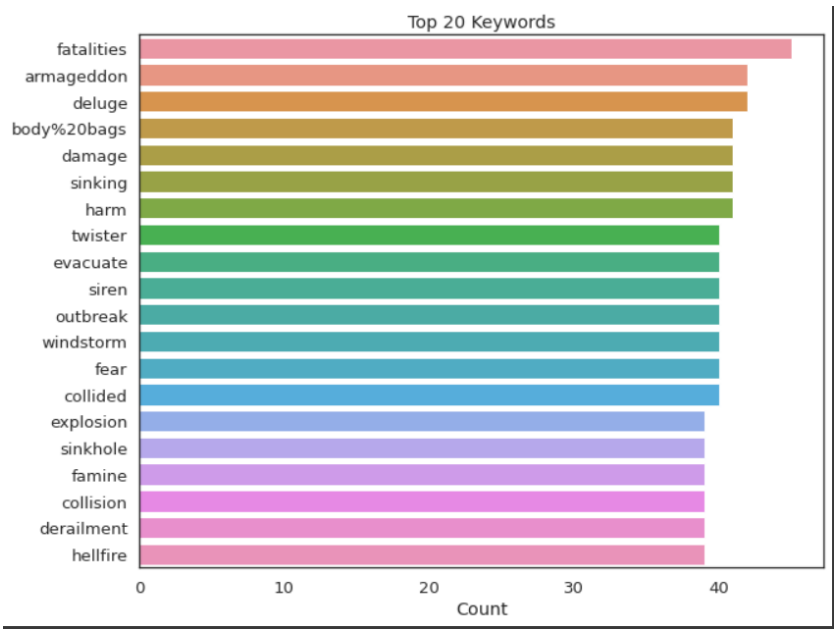

Fig. 8. Most used keywords overall the training data

Not surprisingly, the most used keywords generally carry dangerous and violent meanings. However, we need to check if the same keywords appear as the most used in both Not Disaster and Disaster related tweets.

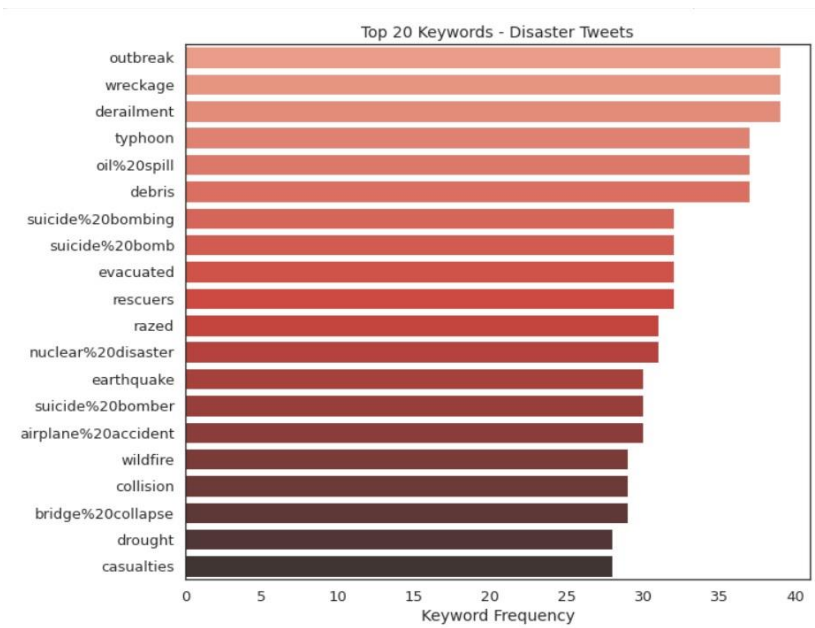

Fig. 9. Most used keywords in Disaster Related Tweets

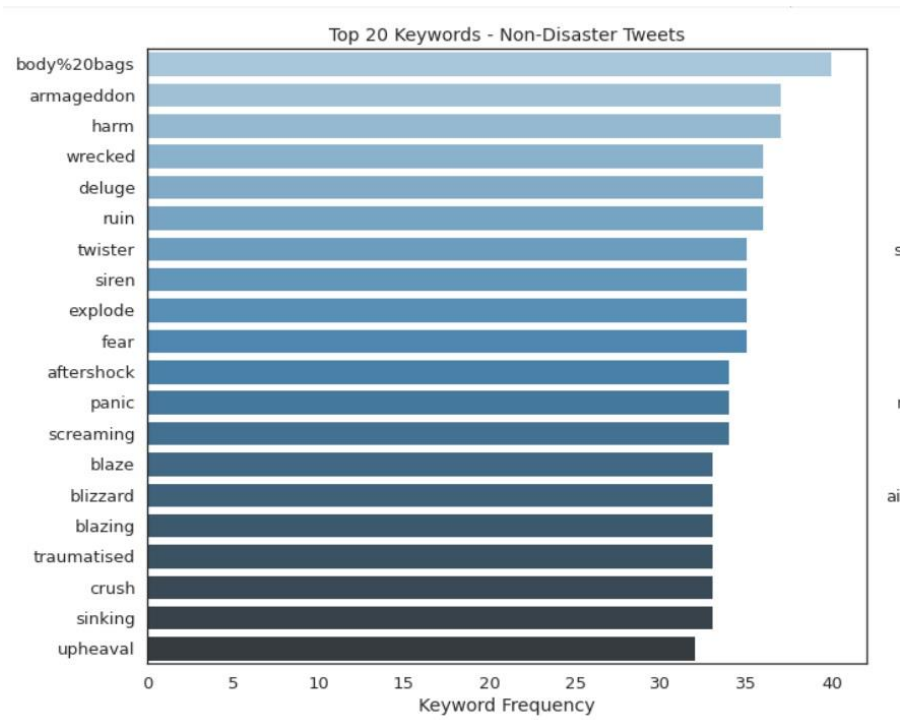

Fig. 10. Most used keywords in Not Disaster Related Tweets

As can be seen, in disaster related tweets and not disaster related tweets, the most used words are scattered differently. This situation, for now, gives us a clue that it will not be very difficult to separate tweets on different topics while defining our models. Of course, it would not be right to classify only according to this separation. We will need to design a model based on the weights of the words. In any case, these results were obtained without any pre-processing on the dataset. After we clear our dataset of noise and normalize it, these results may change. Therefore, before proceeding to model building, it is necessary to apply some pre-processing methods.

Finally, the locations that most tweets are sent from, can be seen below.

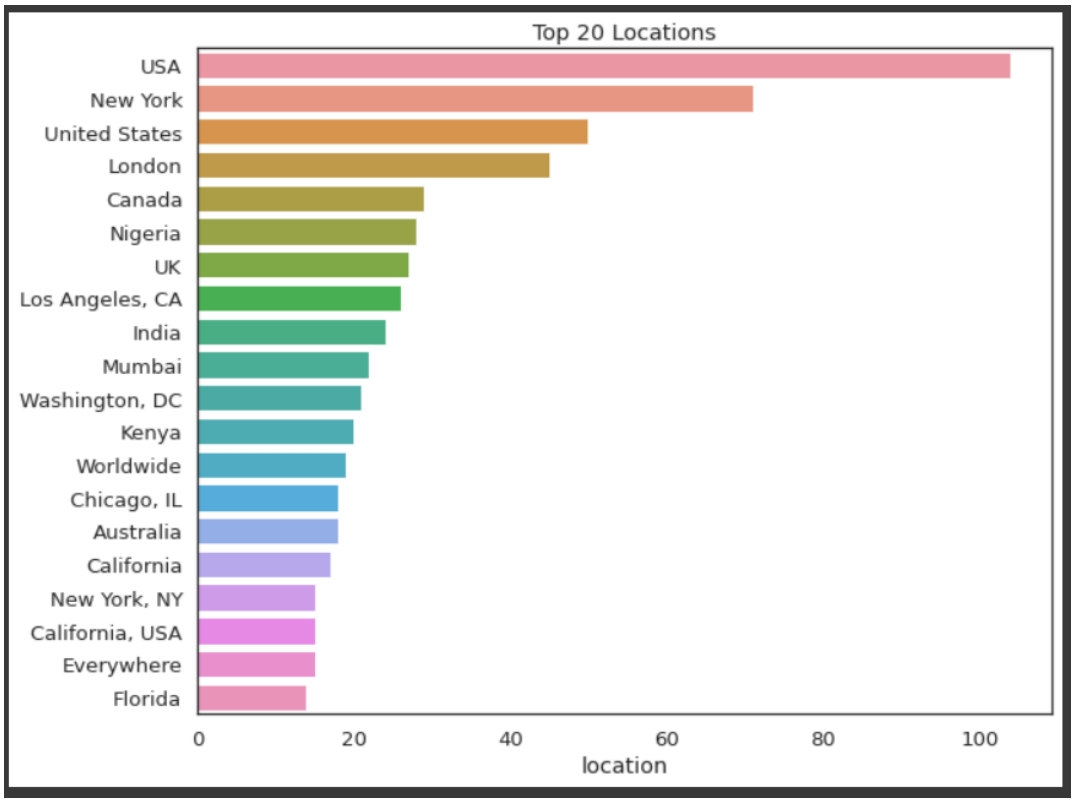

Fig 11. Locations that most disaster tweets are sent from

4. Methods

4.1 Text pre-processing

First, we applied some pre-processing techniques before using the training dataset directly. Our goal in applying these techniques is to normalize the training dataset as much as possible and make sure that any model we train will be as efficient as possible. In the normalization process, we first divided the sentences into small pieces with the tokenization technique, made all texts lower-case, cleaned the stop words in English that had no meaning, and eliminated the punctuation marks that were found in all texts in large numbers. We've cleaned up internet-related terms like "http://, @, #, www, com" that don't tell us about the text content. We defined the location and keywords parameters that contain too much NaN and inconsistent value as noise, thinking that it would not help us much in the modeling, and we eliminated them. We reduced the words to their main roots with the stemming technique, thus preventing the words that have suffixes from being processed as if they were a different word over and over. With the lemmatization technique, we reduced different words with the same meaning to a single unique word, thus we planned to increase accuracy. Below you can see the image of the data we have before and after these normalization processes.

	id	keyword	location	text	target	tweet_length	processed_text
100	144	accident	UK	NorwayMFA Bahrain police had previously died i...	1	124	police previously died road accident they were...
101	145	accident	Nairobi, Kenya	I still have not heard Church Leaders of Kenya...	0	139	still have heard church leaders kenya coming f...
102	146	aftershock	Instagram - @heyimginog	afterShockDeLo scuf ps live and the game cya	0	49	aftershockdelo scuf live game
103	149	aftershock	304	The man who can drive himself further once the...	0	110	drive himself further once effort gets will roger
104	151	aftershock	Switzerland	320 IR ICEMOON AFTERSHOCK djicemoon Dubste...	0	138	icemoon aftershock djicemoon dubstep trapmusic...
105	153	aftershock	304	There is no victory at bargain basement prices...	0	73	there victory bargain basement prices david
106	156	aftershock	US	320 IR ICEMOON AFTERSHOCK djicemoon Dubste...	0	138	icemoon aftershock djicemoon dubstep trapmusic...
107	157	aftershock	304	Nobody remembers who came in second Charles Sc...	0	53	nobody came second charles
108	158	aftershock	Instagram - @heyimginog	afterShockDeLo im speaking from someone that i...	0	125	aftershockdelo speaking from someone that usin...
109	159	aftershock	304	The harder the conflict the more glorious the ...	0	69	conflict more glorious thomas

Fig. 12. Texts before pre-processing and after pre-processing

The below image is acquired after the pre-processing steps completed. It shows the most frequent words bigger in all the texts.

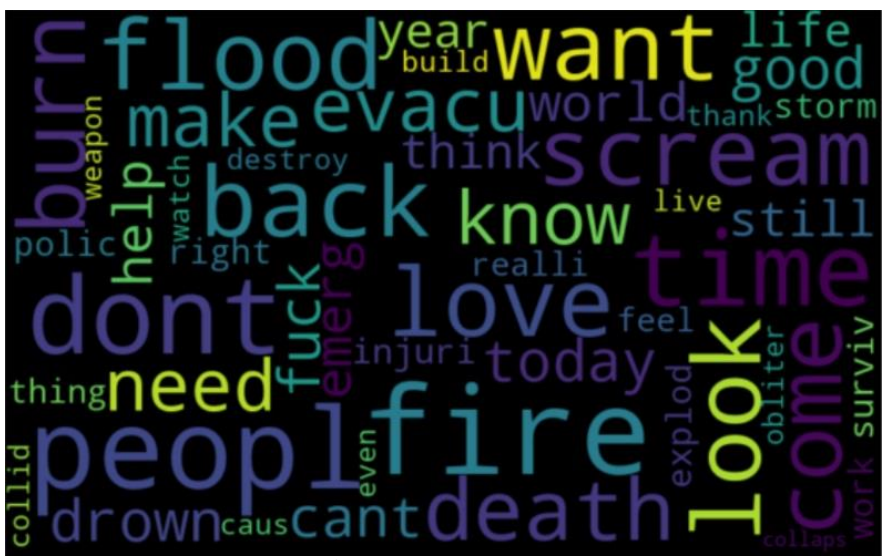

Fig. 13. The most frequent words are represented by their sizes

4.2 Feature extraction with TF-IDF

TF-IDF Term Frequency - stands for Reverse Document Frequency. It is a method for calculating how important a term is to the document it is in. At the same time, it takes into account how often it is seen in other documents where it is compared.

It does this calculation by looking at how often or how many times the word appears in the document it is in. If this word occurs very frequently in other documents, it interprets that the word being processed is not specific to a particular document and penalizes the importance of the word according to this calculation.

In our case, we applied TF-IDF to texts that were already pre-processed and vectorized the words according to their importance weights. After the TF-IDF stage, our data set became suitable for classification models.
The Figure 10 and Figure 11 shows the results of a sample study on TF-IDF vectorization [10]. In below graphs, it can be seen how TF-IDF regularized the dataset and how the labels are separated cleanly.

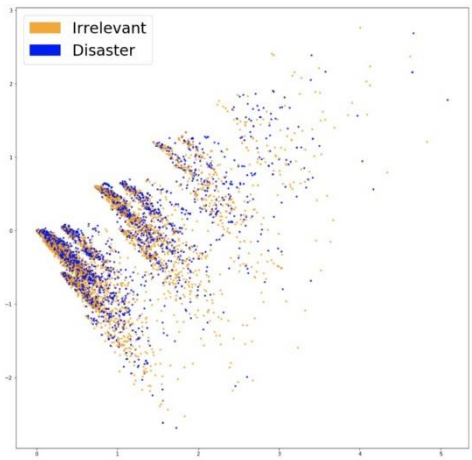

Fig. 14. Before TF-IDF

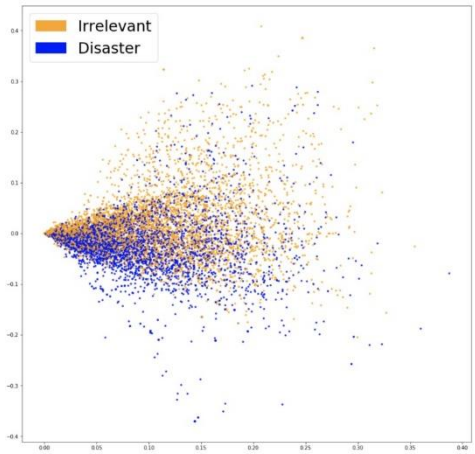

Figure 15. After TF-IDF

4.3 Classification algorithms

In this paper, we used Support Vector Classifier (SVC), MultinomialNB, Logistic Regression, XGBClassifier, Random Forest, Decision Tree and K-Means Clustering as classification algorithms. Most of our models worked with high success rate, since it is a regularly distributed dataset and this is a binary classification problem. We ranked our results in terms of statistical evaluation methods such as accuracy, precision, recall and F-Score. Our aim is to see which classification model will work most successfully in a binary classification problem in a properly distributed and normalized data set. As mentioned earlier in this paper, different results can be obtained with different feature extraction methods and NLP classification models. In order to keep this research simple and to observe the results, we preferred to apply the TF-IDF feature extraction method and 7 different classification models.

5. Results

As can be seen from the table, the best performing classification model on the test set was SVM with 81% accuracy and 80% F-score. The worst performing model with 65% accuracy and 33% F-score was KNN.

Table 1. Model Performances

Kaggle Disaster Tweets Dataset Results				
ML Algorithms	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F-Score (%)**
SVM	81	82	79	80
Multinomial NB	79	80	77	78
Logistic Regression	79	80	78	78
Xgboost Classifier	79	78	77	77
Decision Tree	73	68	69	69
Random Forest	78	80	64	71
K-Nearest Neighbour(KNN)	65	93	20	33

6. Conclusion

SVM performs well when information about the dataset is very limited. It works well even with unstructured and semi-structured data such as texts. It scale successfully to high-dimensional data, being effective when the number of dimensions is larger than the number of samples. It can be applied to both linear and non-linear data and have high accuracy. SVM is good at modeling complex decision boundaries.

Vectors are lists of numbers that represent a set of coordinates in a space, and in some cases can reach very large dimensions. In NLP applications, we can apply the SVM algorithm to text classification problems and get very good results if the vector representations of the texts are as large as possible.

Acknowledgements

We thank to Ravinder Ahujaa, Aakarsha Chuga, Shruti Kohlia, Shaurya Guptaa, and Pratyush Ahujaa from Jaypee Institute of Information Technology, Noida 201301, India for their valuable work “The Impact of Features Extraction on the Sentiment Analysis” that inspired us to make further research about the sentiment analysis and NLP topics.

References

[1] Ravinder Ahuja, Aakarsha Chug, Shruti Kohli, Shaurya Gupta, Pratyush Ahuja, The Impact of Features Extraction on the Sentiment Analysis, Procedia Computer Science, Volume 152, 2019, Pages 341-348, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2019.05.008.

[2] Kaggle (2021, May). Natural Language Processing with Disaster Tweets, Retrieved May 15, 2021 from https://www.kaggle.com/c/nlp-getting-started

[3] Kavita Ganesan. Text Preprocessing for Machine Learning & NLP, Retrieved May 15, from https://kavita-ganesan.com/text-preprocessing-tutorial/#.YLPte_n7RPZ

[4] (2011) TF–IDF. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_832

[5] Ramos, Juan. (2003). Using TF-IDF to determine word relevance in document queries.

[6] Pradhan, Ashis. (2012). Support vector machine-A survey. IJETAE. 2.

[7] Xu, Shuo & Li, Yan & Zheng, Wang. (2017). Bayesian Multinomial Naïve Bayes Classifier to Text Classification. 347-352. 10.1007/978-981-10-5041-1_57.

[8] Santhanam, Ramraj & Uzir, Nishant & Raman, Sunil & Banerjee, Shatadeep. (2017). Experimenting XGBoost Algorithm for Prediction and Classification of Different Datasets.

[9] Aditya Mishra. (2018, Feb). Metrics to Evaluate your Machine Learning Algorithm from https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

[10] Emmanuel ants. (2018, Jan). How to solve 90% of NLP problems: a step-by-step guide from https://blog.insightdatascience.com/how-to-solve-90-of-nlp-problems-a-step-by-step-guide-fda605278e4e

[11] Oscar Contreras Carrasco. (2019, Jul). Support Vector Machines for Classification, Retrieved May 15, 2021 from https://towardsdatascience.com/support-vector-machines-for-classification-fc7c1565e3

[12] Ghaith khlifi. (2021, Feb). Zindi competition Ensemble Models tweet prediction. Retrieved May 18, 2021 from https://www.kaggle.com/ghaithkhlifi/zindi-competition-ensemble-models-tweet-prediction

[13] gautam iruvanti. (2020, Jul). Real or Not? NLP with Disaster Tweets (Classification using Google BERT), Retrieved May 18, 2021 from https://levelup.gitconnected.com/real-or-not-nlp-with-disaster-tweets-classification-using-google-bert-76d2702807b4

[14] Zineb KHANJARI. (2021, Jan). Disaster Tweets : multiple vectorizers and models, Retrieved May 19 from https://www.kaggle.com/zinebkhanjari/disaster-tweets-multiple-vectorizers-and-models?select=logisticregression_using_cbow_vector.joblib

[15] AhmedMazenAhmedMurad. (2020, Jun). NLP_Disaster_Tweets with ELECTRA Base, Retrieved May 20 from https://www.kaggle.com/ahmedmurad1990/nlp-disaster-tweets-with-electra-base

[16] Ghaiyur. (2021, Feb). Ensemble Models-versiong, Retrieved May 20 from https://www.kaggle.com/ghaiyur/ensemble-models-versiong?select=randomforestclassifier_using_cbow_vector.joblib

Appendix A

-*- coding: utf-8 -*-
"""Natural_Language_Processing_for_Prediction_of_Disaster_Tweets_using_Machine_Learning_Methods (6).ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1Vmp2PoCrH8adw0BUYUCGVXrDyqdL7akZ

#Natural_Language_Processing_for_Prediction_of_Disaster_Tweets_using_Machine_Learning_Methods

```
### In this project we aim to compare Machine Learning methods in the case of Natural language processing on disaster tweets dataset
"""

# Commented out IPython magic to ensure Python compatibility.
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
import re
import nltk

nltk.download('wordnet')
import pickle
import gc
import os
import time
import math
import random
import torch
import torch.nn as nn
import torch.utils.data
import torch.nn.functional as F
import string
import folium
from sklearn.metrics import confusion_matrix
import scipy as sp
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report

# %matplotlib inline
from joblib import dump
sns.set(style="white", font_scale=1.2)
plt.rcParams["figure.figsize"] = [10,8]
pd.set_option.display_max_columns = 0
pd.set_option.display_max_rows = 0
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from collections import Counter
from itertools import chain
from sklearn.feature_extraction.text import TfidfVectorizer,HashingVectorizer

from datetime import date
```

```
from tqdm.notebook import tqdm

import networkx as nx
from pandas import Timestamp

from PIL import Image
from IPython.display import SVG

import requests
from IPython.display import HTML

from tqdm import tqdm
import matplotlib.cm as cm

tqdm.pandas()

import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots


import tensorflow as tf

from tensorflow.keras.callbacks import Callback
from sklearn.metrics import accuracy_score, roc_auc_score
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, CSVLogger

from tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import activations
from tensorflow.keras import constraints
from tensorflow.keras import initializers
from tensorflow.keras import regularizers


from sklearn import metrics
from sklearn.utils import shuffle
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from textblob import TextBlob
from nltk.corpus import wordnet
from nltk import WordNetLemmatizer
from nltk.stem import WordNetLemmatizer,PorterStemmer
from wordcloud import WordCloud
from nltk.sentiment.vader import SentimentIntensityAnalyzer

"""## Reading the data"""

from google.colab import drive
drive.mount('/content/drive')

train  = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Machine Learning/Final Project/train.csv')
test = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Machine Learning/Final Project/test.csv')

"""## Preprocessing"""

train

test

#compare tweets in train and test file
length_train=train['text'].str.len()
length_test=test['text'].str.len()
plt.hist(length_train, bins=20, label="train_tweets", color = 'red', alpha = 0.5)
plt.hist(length_test, bins=20, label="test_tweets", color = 'Blue', alpha = 0.5)
plt.legend()
plt.show()

train["tweet_length"] = train["text"].apply(len)
test["tweet_length"] = test["text"].apply(len)
sns.histplot(train["tweet_length"])
plt.title("Histogram of Tweet Length")
plt.xlabel("Number of Characters")
plt.ylabel("Density")
plt.show()
sns.histplot(test["tweet_length"], color='y')
plt.title("Histogram of Tweet Length")
plt.xlabel("Number of Characters")
plt.ylabel("Density")
plt.show()

"""## *Disaster or *Non* disaster*"""

tweet_len=train[train['target']==1]['text'].str.len()
sns.histplot(tweet_len,color = 'b')
```

```
plt.title('Characters in tweets')
plt.xlabel("Disaster tweets")
tweet_len=train[train['target']==0]['text'].str.len()
sns.histplot(tweet_len,color = 'y')
plt.title('Characters in tweets')
plt.xlabel("Non Disaster tweets")
plt.show()

train['target'].value_counts()

"""### In this dataset we have 4342 tweets not related to disaster and 3271 tweets related to disaster

57% not disaster
        42% disaster

### Analysing the keywords
"""

keywords_vc = pd.DataFrame({"Count": train["keyword"].value_counts()})
sns.barplot(y=keywords_vc[0:20].index, x=keywords_vc[0:20]["Count"], orient='h')
plt.title("Top 20 Keywords")
plt.show()

disaster_keywords = train.loc[train["target"] == 1]["keyword"].value_counts()
nondisaster_keywords = train.loc[train["target"] == 0]["keyword"].value_counts()

fig, ax = plt.subplots(1,2, figsize=(20,8))
sns.barplot(y=disaster_keywords[0:30].index, x=disaster_keywords[0:30], orient='h', ax=ax[0],
palette="Reds_d")
sns.barplot(y=nondisaster_keywords[0:30].index, x=nondisaster_keywords[0:30], orient='h', ax=ax[1],
palette="Blues_d")
ax[0].set_title("Top 30 Keywords - Disaster Tweets")
ax[0].set_xlabel("Keyword Frequency")
ax[1].set_title("Top 30 Keywords - Non-Disaster Tweets")
ax[1].set_xlabel("Keyword Frequency")
plt.tight_layout()
plt.show()

"""### Analysing the locations"""

locations_vc = train["location"].value_counts()
sns.barplot(y=locations_vc[0:20].index, x=locations_vc[0:20], orient='h')
plt.title("Top 20 Locations")
plt.show()

"""## Null values"""

# Checking Null values
```

```
missing_train = train.isnull().sum()
missing_test = test.isnull().sum()
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(15,5))
missing_train = missing_train[missing_train>0].sort_values()
ax1.pie(missing_train,autopct='%1.1f%%',startangle=30,explode=[0.9,0],labels=["keyword","location"],color
s=['yellow','cyan'])
ax1.set_title("Null values present in Train Dataset")
missing_test = missing_test[missing_test>0].sort_values()
ax2.pie(missing_test,autopct='%1.1f%%',startangle=30,explode=[0.9,0],labels=["keyword","location"],colors
=['yellow','#66ff00'])
ax2.set_title("Null values present in Test Dataset")
plt.suptitle("Distribution of Null Values in Dataset")
plt.tight_layout()
plt.show()

"""### it is observed that columns, Keyword and Location contains missing values.

### the column having maximum missing values is: location while Keywords column has the minimum count
of missing values for both sets of data.

## Tokenizing
"""

from tensorflow.keras.preprocessing.text import Tokenizer
token = Tokenizer()
token.fit_on_texts(train)
token.fit_on_texts(test)

"""## Normalization"""

def remove_URL(text):
    url = re.compile(r"https?://\S+|www\.\S+")
    return url.sub(r"", text)
#Bilal, we should add this refrence and then remove this
# https://stackoverflow.com/questions/34293875/how-to-remove-punctuation-marks-from-a-string-in-python-
3-x-using-translate/34294022
def remove_punct(text):
    translator = str.maketrans("", "", string.punctuation)
    return text.translate(translator)

string.punctuation

#Removing links
pattern = re.compile(r"https?://(\S+|www)\.\S+")
for t in train.text:
    matches = pattern.findall(t)
    for match in matches:
        print(t)
```

```
        print(match)
        print(pattern.sub(r"", t))
    if len(matches) > 0:
        break

#removing links in test too
pattern = re.compile(r"https?://(\S+|www)\.\S+")
for t in test.text:
    matches = pattern.findall(t)
    for match in matches:
        print(t)
        print(match)
        print(pattern.sub(r"", t))
    if len(matches) > 0:
        break

"""###Punction"""

train["text"] = train.text.map(remove_URL) # map(lambda x: remove_URL(x))
train["text"] = train.text.map(remove_punct)
# for test too
test["text"] = test.text.map(remove_URL)
test["text"] = test.text.map(remove_punct)

# Importing HTMLParser
from html.parser import HTMLParser
html_parser = HTMLParser()

# Created a new columns i.e. processed_text contains the same tweets but cleaned version
train['processed_text'] = train['text'].apply(lambda x: html_parser.unescape(x))
test['processed_text'] = test['text'].apply(lambda x: html_parser.unescape(x))

# Apostrophe Dictionary
apostrophe_dict = {
"ain't": "am not / are not",
"aren't": "are not / am not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
```

"haven't": "have not",
"he'd": "he had / he would",
"he'd've": "he would have",
"he'll": "he shall / he will",
"he'll've": "he shall have / he will have",
"he's": "he has / he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how has / how is",
"i'd": "I had / I would",
"i'd've": "I would have",
"i'll": "I shall / I will",
"i'll've": "I shall have / I will have",
"i'm": "I am",
"i've": "I have",
"isn't": "is not",
"it'd": "it had / it would",
"it'd've": "it would have",
"it'll": "it shall / it will",
"it'll've": "it shall have / it will have",
"it's": "it has / it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she had / she would",
"she'd've": "she would have",
"she'll": "she shall / she will",
"she'll've": "she shall have / she will have",
"she's": "she has / she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",

"so's": "so as / so is",
"that'd": "that would / that had",
"that'd've": "that would have",
"that's": "that has / that is",
"there'd": "there had / there would",
"there'd've": "there would have",
"there's": "there has / there is",
"they'd": "they had / they would",
"they'd've": "they would have",
"they'll": "they shall / they will",
"they'll've": "they shall have / they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we had / we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what shall / what will",
"what'll've": "what shall have / what will have",
"what're": "what are",
"what's": "what has / what is",
"what've": "what have",
"when's": "when has / when is",
"when've": "when have",
"where'd": "where did",
"where's": "where has / where is",
"where've": "where have",
"who'll": "who shall / who will",
"who'll've": "who shall have / who will have",
"who's": "who has / who is",
"who've": "who have",
"why's": "why has / why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",

```
"y'all've": "you all have",
"you'd": "you had / you would",
"you'd've": "you would have",
"you'll": "you shall / you will",
"you'll've": "you shall have / you will have",
"you're": "you are",
"you've": "you have",
"forgiv":"forgive",
"peopl":"people"
}

def lookup_dict(text, dictionary):
    for word in text.split():
        if word.lower() in dictionary:
            if word.lower() in text.split():
                text = text.replace(word, dictionary[word.lower()])
    return text

train['processed_text'] = train['processed_text'].apply(lambda x: lookup_dict(x,apostrophe_dict))
test['processed_text'] = test['processed_text'].apply(lambda x: lookup_dict(x,apostrophe_dict))

short_word_dict = {
"121": "one to one",
"a/s/l": "age, sex, location",
"adn": "any day now",
"afaik": "as far as I know",
"afk": "away from keyboard",
"aight": "alright",
"alol": "actually laughing out loud",
"b4": "before",
"b4n": "bye for now",
"bak": "back at the keyboard",
"bf": "boyfriend",
"bff": "best friends forever",
"bfn": "bye for now",
"bg": "big grin",
"bta": "but then again",
"btw": "by the way",
"cid": "crying in disgrace",
"cnp": "continued in my next post",
"cp": "chat post",
"cu": "see you",
"cul": "see you later",
"cul8r": "see you later",
"cya": "bye",
"cyo": "see you online",
"dbau": "doing business as usual",
"fud": "fear, uncertainty, and doubt",
```

"fwiw": "for what it's worth",
"fyi": "for your information",
"g": "grin",
"g2g": "got to go",
"ga": "go ahead",
"gal": "get a life",
"gf": "girlfriend",
"gfn": "gone for now",
"gmbo": "giggling my butt off",
"gmta": "great minds think alike",
"h8": "hate",
"hagn": "have a good night",
"hdop": "help delete online predators",
"hhis": "hanging head in shame",
"iac": "in any case",
"ianal": "I am not a lawyer",
"ic": "I see",
"idk": "I don't know",
"imao": "in my arrogant opinion",
"imnsho": "in my not so humble opinion",
"imo": "in my opinion",
"iow": "in other words",
"ipn": "I'm posting naked",
"irl": "in real life",
"jk": "just kidding",
"l8r": "later",
"ld": "later, dude",
"ldr": "long distance relationship",
"llta": "lots and lots of thunderous applause",
"lmao": "laugh my ass off",
"lmirl": "let's meet in real life",
"lol": "laugh out loud",
"ltr": "longterm relationship",
"lulab": "love you like a brother",
"lulas": "love you like a sister",
"luv": "love",
"m/f": "male or female",
"m8": "mate",
"milf": "mother I would like to fuck",
"oll": "online love",
"omg": "oh my god",
"otoh": "on the other hand",
"pir": "parent in room",
"ppl": "people",
"r": "are",
"rofl": "roll on the floor laughing",
"rpg": "role playing games",
"ru": "are you",

```
"shid": "slaps head in disgust",
"somy": "sick of me yet",
"sot": "short of time",
"thanx": "thanks",
"thx": "thanks",
"ttyl": "talk to you later",
"u": "you",
"ur": "you are",
"uw": "you’re welcome",
"wb": "welcome back",
"wfm": "works for me",
"wibni": "wouldn't it be nice if",
"wtf": "what the fuck",
"wtg": "way to go",
"wtgp": "want to go private",
"ym": "young man",
"gr8": "great"
}

train['processed_text'] = train['processed_text'].apply(lambda x: lookup_dict(x,short_word_dict))
test['processed_text'] = test['processed_text'].apply(lambda x: lookup_dict(x,short_word_dict))

emoticon_dict = {
":)": "happy",
":-)": "happy",
":-]": "happy",
":-3": "happy",
":->": "happy",
"8-)": "happy",
":-}": "happy",
":o)": "happy",
":c)": "happy",
":^)": "happy",
"=]": "happy",
"=)": "happy",
"<3": "happy",
":-(": "sad",
":(": "sad",
":c": "sad",
":<": "sad",
":[": "sad",
">:[": "sad",
":{": "sad",
">:(": "sad",
":-c": "sad",
":-< ": "sad",
":-[": "sad",
":-||": "sad"
```

```
}

train['processed_text'] = train['processed_text'].apply(lambda x: lookup_dict(x,emoticon_dict))
test['processed_text'] = test['processed_text'].apply(lambda x: lookup_dict(x,emoticon_dict))

#make all words lower case
train['processed_text'] = train['processed_text'].str.lower()
test['processed_text'] = test['processed_text'].str.lower()

#Remove punctuation
table = str.maketrans('', '', string.punctuation)
train['processed_text'] = [train['processed_text'][row].translate(table) for row in
range(len(train['processed_text']))]
test['processed_text'] = [test['processed_text'][row].translate(table) for row in
range(len(test['processed_text']))]

# remove hash tags
train['processed_text'] = train['processed_text'].str.replace("#", " ")
test['processed_text'] = test['processed_text'].str.replace("#", " ")

#remove words less than 1 character
train['processed_text'] = train['processed_text'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))
test['processed_text'] = test['processed_text'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))

"""## Lemmatization"""

lemm = WordNetLemmatizer()
lemm_text = [lemm.lemmatize(word) for word in train['processed_text']]
train['processed_text']=lemm_text

lemm_text2 = [lemm.lemmatize(word) for word in test['processed_text']]
test['processed_text']=lemm_text2

"""## Stemmering"""

stemmer = PorterStemmer()
words = stopwords.words("english")

train['processed_text'] = train['text'].apply(lambda x: " ".join([stemmer.stem(i)
for i in re.sub("[^a-zA-Z]", " ", x).split() if i not in words]).lower())

test['processed_text'] = test['text'].apply(lambda x: " ".join([stemmer.stem(i)
for i in re.sub("[^a-zA-Z]", " ", x).split() if i not in words]).lower())

"""### Stopwords"""

#put frequent words in a mosiac
freq_words = ' '.join([text for text in train['processed_text']])
```

```
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110,
max_words=50).generate(freq_words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()

#Dropping words whose length is less than 3
train['processed_text'] = train['processed_text'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))
test['processed_text'] = test['processed_text'].apply(lambda x: ' '.join([w for w in x.split() if len(w)>3]))

"""### Remove rare words"""

# split words into lists
v = train['processed_text'].str.split().tolist()
# compute global word frequency
c = Counter(chain.from_iterable(v))
# filter, join, and re-assign
train['processed_text'] = [' '.join([j for j in i if c[j] > 1]) for i in v]

# split words into lists for test too
v = test['processed_text'].str.split().tolist()
# compute global word frequency
c = Counter(chain.from_iterable(v))
# filter, join, and re-assign
test['processed_text'] = [' '.join([j for j in i if c[j] > 1]) for i in v]

"""# **preprocessing ended**

### Implementation of Machine Learning methods
"""

def fit_and_predict(model,x_train,x_test,y_train,y_test,vectoriser):

    classifier = model
    classifier_name = str(classifier.__class__.__name__)
    classifier.fit(x_train,y_train)
    filename = classifier_name +" using "+ str(vectoriser)+".joblib"
    filename = filename.lower().replace(" ","_")
    dump(model, filename=filename)
    y_pred = classifier.predict(x_test)
    cmatrix = confusion_matrix(y_test,y_pred)
    f,ax = plt.subplots(figsize=(3,3))
    sns.heatmap(cmatrix,annot=True,linewidths=0.5,cbar=False,linecolor="red",fmt='.0f',ax=ax)
    plt.xlabel("y_predict")
    plt.ylabel("y_true")
    ax.set(title=str(classifier))
    plt.show()
```

```
    f1score = f1_score(y_test,y_pred,average='weighted')
    train_accuracy = round(classifier.score(x_train,y_train)*100)
    test_accuracy =  round(accuracy_score(y_test,y_pred)*100)
    print(str(classifier.__class__.__name__) +" using "+ str(vectoriser))
    print(classification_report(y_test,y_pred))
    print('Accuracy of classifier on training set:{}%'.format(train_accuracy))
    print('Accuracy of classifier on test set:{}%' .format(test_accuracy))

models=[
        XGBClassifier(max_depth=6, n_estimators=1000),
        LogisticRegression(random_state=5),
        SVC(random_state=5),
        MultinomialNB(),
        DecisionTreeClassifier(random_state = 5),
        KNeighborsClassifier(),
        RandomForestClassifier(random_state=5),
       ]

"""## Implementation of TfIdf Feature extraction"""

def tfidf_vector(data):
    tfidf_vectorizer = TfidfVectorizer()
    vect = tfidf_vectorizer.fit_transform(data)
    return vect, tfidf_vectorizer

X_train_tfidf, tfidf_vectorizer = tfidf_vector(train['processed_text'])

X_test_tfidf = tfidf_vectorizer.transform(test['processed_text'])

with open('tfidf_vectorizer.pickle', 'wb') as handle:
    pickle.dump(tfidf_vectorizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

"""## **Fitting the models**"""

np.random.seed(0)
random_state = 29

for model in models:
    y=train.target
    x = X_train_tfidf
    x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3)
    fit_and_predict(model,x_train,x_test,y_train,y_test, 'Tfidf vector')

"""### **Apply SVM on the Kaggle test**"""

classifier = SVC(random_state=5)
    classifier.fit(x_train,y_train)
    y_pred = classifier.predict(X_test_tfidf)
```

```
"""## Submission in Kaggle"""

submission = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Machine Learning/Final
Project/sample_submission.csv")
submission['target'] = y_pred
submission.to_csv('submission.csv', index=False)

submission
```