Competition

Twitter has become an important communication channel in times of emergency. The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Because of this, more agencies are interested in programatically monitoring Twitter (i.e. disaster relief organizations and news agencies).

But, it's not always clear whether a person's words are actually announcing a disaster. In this competition, we are challenged to build a machine learning model that predicts which Tweets are about real disasters and which one's aren't, having access to a dataset of 10,000 tweets that were hand classified.

Importing necessary libraries

In [84]:

```
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
import math
import os

import re
import string
from string import punctuation

import xgboost as xgb
from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import f1_score
from sklearn import preprocessing, decomposition, model_selection, metrics, pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import feature_extraction, linear_model, model_selection, preprocessing
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
```

Reading Datasets

In [85]:

```
train_set = pd.read_csv('C:/Users/kushagra/Downloads/train.csv')
```

In [78]:

```
test_set = pd.read_csv('C:/Users/kushagra/Downloads/test.csv')
```

In [87]:

```
train_set.shape
```

Out[87]:

```
(7613, 5)
```

In [88]:

```
test_set.shape
```

Out[88]:

```
(3263, 4)
```

Basic Exploratory data analysis

In [89]:

```
#Checking missing values
train_set.isnull().sum()
```

Out[89]:

```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

In [90]:

```
test_set.isnull().sum()
# A lot of missing values in location column both in test and train set and some in keyword
```

Out[90]:

```
id             0
keyword       26
location    1105
text           0
dtype: int64
```

After that I started exploring target column as we have to predict whether a given tweet is about real disaster or not, predict 1 if it is a real disaster and 0 if it's not a real disaster.

In [91]:

```
train_set.target.value_counts()
```

Out[91]:

```
0    4342
1    3271
Name: target, dtype: int64
```

As per the count, tweets related to real disaster are less than tweets that are not related to disaster

In [92]:

```
train_set.head()
```

Out[92]:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

In [93]:

```
#Let's see how often the word 'disaster' come in the dataset
train_set.loc[train_set['text'].str.contains('disaster', na=False, case=False)].target.value_counts
()
```

Out[93]:

```
1    102
0     40
Name: target, dtype: int64
```

Text Preprocessing

In text processing first I cleaned the data, we need to pre-process the data to get it all in a consistent format.We need to clean, tokenize and convert our data into a matrix.

In [94]:

```
train_set['text'][:5]
```

Out[94]:

```
0    Our Deeds are the Reason of this #earthquake M...
1               Forest fire near La Ronge Sask. Canada
2    All residents asked to 'shelter in place' are ...
3    13,000 people receive #wildfires evacuation or...
4    Just got sent this photo from Ruby #Alaska as ...
Name: text, dtype: object
```

In [13]:

```
# Remove non ASCII characters

def remove_ascii(text):
    text= ''.join([x for x in text if x in string.printable])

    # Removing URLs
    text = re.sub(r"http\S+", "", text)

    return text
```

In [95]:

```
train_set["text"] = train_set["text"].apply(lambda x: remove_ascii(x))
test_set["text"] = test_set["text"].apply(lambda x: remove_ascii(x))
```

In [96]:

```
 # Remove numbers from text
def clean(text):

  text = re.sub(r'[0-9]*','',text.lower())
  text = re.sub(r'[0-9]+[0-9.,]+[0-9]+','',text)

#Remove text in square brackets and punctuation
  text = re.sub('\[.*?\]', '', text)
  text = re.sub('<.*?>+', '', text)
  text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
  return text
```

In [97]:

```
# Applying the cleaning function to both test and training datasets
train_set['text'] = train_set['text'].apply(lambda x: clean(x))
test_set['text'] = test_set['text'].apply(lambda x: clean(x))
```

In [98]:

```
train_set['text'].head()
```

Out[98]:

```
0    our deeds are the reason of this earthquake ma...
```

```
1                forest fire near la ronge sask canada
2    all residents asked to shelter in place are be...
3     people receive wildfires evacuation orders in...
4    just got sent this photo from ruby alaska as s...
Name: text, dtype: object
```

Tokenization

Tokenization is a process of chopping a character into pieces, called as tokens, where the tokens can be word, sentence, paragraph etc

In [99]:

```
# Lets take a sentence
text = "My car is so fast"

tokenizer1 = nltk.tokenize.RegexpTokenizer(r'\w+')
tokenizer2 = nltk.tokenize.WordPunctTokenizer()
tokenizer3 = nltk.tokenize.TreebankWordTokenizer()
tokenizer4 = nltk.tokenize.WhitespaceTokenizer()
```

In [100]:

```
print("Tokenization by Regular expression:- ",tokenizer1.tokenize(text))
print("Tokenization by punctuation:- ",tokenizer2.tokenize(text))
print("Tokenization by Tree bank wordtokenizer:- ",tokenizer3.tokenize(text))
print("Tokenization by whitespace tokenizer :- ",tokenizer4.tokenize(text))
```

```
Tokenization by Regular expression:-  ['My', 'car', 'is', 'so', 'fast']
Tokenization by punctuation:-  ['My', 'car', 'is', 'so', 'fast']
Tokenization by Tree bank wordtokenizer:-  ['My', 'car', 'is', 'so', 'fast']
Tokenization by whitespace tokenizer :-  ['My', 'car', 'is', 'so', 'fast']
```

In [101]:

```
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
train_set['text'] = train_set['text'].apply(lambda x: tokenizer.tokenize(x))
test_set['text'] = test_set['text'].apply(lambda x: tokenizer.tokenize(x))
```

In [102]:

```
train_set['text'].head()
```

Out[102]:

```
0    [our, deeds, are, the, reason, of, this, earth...
1        [forest, fire, near, la, ronge, sask, canada]
2    [all, residents, asked, to, shelter, in, place...
3    [people, receive, wildfires, evacuation, order...
4    [just, got, sent, this, photo, from, ruby, ala...
Name: text, dtype: object
```

Token Normalization

Converting different tokens to their base forms is called Token normalization, can be done by these methods:

1. Stemming : Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes, for instance cats – cat, wolves – wolv
2. Lemmatization : Lemmatization is a process that returns the base form of word, known as lemma, it involves looking for interesting patterns in the text or to extract data from the text to be inserted into a database.

Stemming

In [103]:

```
from nltk.stem.porter import PorterStemmer
nltk.download('stopwords')
porterstem = PorterStemmer()
from nltk.stem.porter import PorterStemmer
# Removing unwanted words and removing stopwords then stemming it and then appended cleaned tweet
to corpus
corpus  = []
for i in range(train_set['text'].shape[0]):
    text = re.sub("[^a-zA-Z]", ' ', str(train_set['text'][i]))
    text = [porterstem.stem(word) for word in text if not word in set(stopwords.words('english'))]
    text = ' '.join(text)
    corpus.append(text)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\kushagra\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Lemmatizing

In [104]:

```
import nltk
nltk.download('wordnet')
token = tokenizer3.tokenize(text)
lemmatizer=nltk.stem.WordNetLemmatizer()
print("Lemmatizing the sentence: ", " ".join(lemmatizer.lemmatize(token) for token in token))
```

```
Lemmatizing the sentence:  h e l e r e h e r z e b n r h e r n c l f r n w l f r e b c n e w
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\kushagra\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [106]:

```
#Combining all the texts and make it a large one
def large_text(list_of_text):
    large_text = ' '.join(list_of_text)
    return large_text
```

In [107]:

```
train_set['text'] = train_set['text'].apply(lambda x : large_text(x))
test_set['text'] = test_set['text'].apply(lambda x : large_text(x))
train_set['text']
train_set.head()
```

Out[107]:

	id	keyword	location	text	target
0	1	NaN	NaN	our deeds are the reason of this earthquake ma...	1
1	4	NaN	NaN	forest fire near la ronge sask canada	1
2	5	NaN	NaN	all residents asked to shelter in place are be...	1
3	6	NaN	NaN	people receive wildfires evacuation orders in ...	1
4	7	NaN	NaN	just got sent this photo from ruby alaska as s...	1

In [108]:

```
train_set['text'].dropna(inplace=True)
train_set['text'] = [entry.lower() for entry in train_set['text']]
```

A text preprocessing function concludes the pre-processing part. For better reusability, it would be better to convert all the steps

undertaken into a function

In [109]:

```
#Cleaning and parseing the entire text
def text_preprocessing(text):
    tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')

    nopunctuation = clean(text)
    tokenized_text = tokenizer.tokenize(nopunctuation)
    remove_stopwords = [w for w in tokenized_text if w not in stopwords.words('english')]
    large_text = ' '.join(remove_stopwords)
    return large_text
```

Tokens to vectors

We need to transform text into a meaningful vector or arrays of numbers.

In [110]:

```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
train_vectors = cv.fit_transform(train_set['text'])
test_vectors = cv.transform(test_set["text"])
```

Term Frequency-Inverse Document Frequency Features- TFIDF

TFIDF features describes the occurrence of rare words in a document by measuring the frequency of each word in a document against how infrequently each word appears in other documents, in other words highly frequent words start to dominate in the document, but may not contain much “informational content”. This approach is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach to scoring is called Term Frequency-Inverse Document Frequency, or TF-IDF,

eg: Corpus :

the quick fox jumped the brown fox jumped over the brown dog fox jumped high the dog d black

document: the brown fox jumped over the brown dog

vocabulary: [a, the, of, brwn, fox, house, jumped, over, with, dog]

feature vector: [0, 2*log(4/2), 1*log(4/3), 0, 1*log(4/3), 1*log(4/2)] [0, 0.575, 0, 1.386, 0.288, 0, 0.288, 1.386, 0, 0.693]

In [111]:

```
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
train_tfidf = tfidf.fit_transform(train_set['text'])
test_tfidf = tfidf.transform(test_set["text"])
```

Building a classification model

Let's see how our model performs, starting with K-Nearest model

K - Nearest model

In [33]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import model_selection
from sklearn import preprocessing, decomposition, model_selection, metrics, pipeline
from sklearn.model_selection import GridSearchCV,StratifiedKFold,RandomizedSearchCV


classifier_knn = KNeighborsClassifier()
scores = model_selection.cross_val_score(classifier_knn, train_vectors, train_set["target"], cv=5,
scoring="f1")
scores
```

Out[33]:

```
array([0.13461538, 0.14058355, 0.14965986, 0.11034483, 0.22279793])
```

In [112]:

```
classifier_knn.fit(train_vectors, train_set["target"])
```

Out[112]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [113]:

```
classifier_knn_tfidf = KNeighborsClassifier()
scores = model_selection.cross_val_score(classifier_knn_tfidf, train_tfidf, train_set["target"],
cv=5, scoring="f1")
scores
```

Out[113]:

```
array([0.00911854, 0.03598201, 0.00608828, 0.        , 0.07591241])
```

Logistic Regression

In [114]:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C=1.0)
scores = model_selection.cross_val_score(clf, train_vectors, train_set["target"], cv=5, scoring="f1
")
scores
```

```
C:\Users\kushagra\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[114]:

```
array([0.62417994, 0.55230126, 0.61097461, 0.58152174, 0.71641791])
```

In [115]:

```
clf.fit(train_vectors, train_set["target"])
```

Out[115]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [116]:

```
 # Fitting a simple Logistic Regression on TFIDF
clf_tfidf = LogisticRegression(C=1.0)
scores = model_selection.cross_val_score(clf_tfidf, train_tfidf, train_set["target"], cv=5, scoring
="f1")
```

```
scores
```

Out[116]:

```
array([0.61293532, 0.56635514, 0.61333333, 0.59558117, 0.72497897])
```

Countvectorizer gives a better performance than TFIDF in Logistic Regression.

Gradient Boosting

In [46]:

```
from sklearn.ensemble import GradientBoostingClassifier

classifier_Gradient = GradientBoostingClassifier()
scores = model_selection.cross_val_score(classifier_Gradient, train_vectors, train_set["target"], c
v=5, scoring="f1")
scores
```

Out[46]:

```
array([0.45753114, 0.33369214, 0.43444227, 0.40674157, 0.54155496])
```

In [47]:

```
classifier_Gradient.fit(train_vectors, train_set["target"])
```

Out[47]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [48]:

```
classifier_Gradient_tfidf = GradientBoostingClassifier()
scores = model_selection.cross_val_score(classifier_Gradient_tfidf, train_tfidf,
train_set["target"], cv=5, scoring="f1")
scores
```

Out[48]:

```
array([0.5060241 , 0.40245148, 0.44726562, 0.45556691, 0.55272727])
```

Naives Bayes Classifier

Naive Bayes Classifier is said to work well with text data

In [117]:

```
from sklearn.naive_bayes import MultinomialNB

classifier_NB = MultinomialNB()
scores = model_selection.cross_val_score(classifier_NB, train_vectors, train_set["target"], cv=5, s
coring="f1")
scores
```

Out[117]:

```
array([0.64970314, 0.63157895, 0.68717949, 0.64433812, 0.74269819])
```

In [118]:

```
classifier_NB.fit(train_vectors, train_set["target"])
```

Out[118]:

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [123]:

```
clf_NB_TFIDF = MultinomialNB()
scores = model_selection.cross_val_score(classifier_NB_TFIDF, train_tfidf, train_set["target"], cv=
5, scoring="f1")
scores
```

Out[123]:

```
array([0.59642147, 0.59016393, 0.62620932, 0.60548723, 0.75324675])
```

naive bayes on TFIDF features scores much better than logistic regression model.

In [124]:

```
clf_NB_TFIDF.fit(train_tfidf, train_set["target"])
```

Out[124]:

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Submissions

In [131]:

```
#Making submission
def submission(submission_file_path,model,test_vectors):
    sample_submission = pd.read_csv(submission_file_path)
    sample_submission["target"] = model.predict(test_vectors)
    sample_submission.to_csv("submission.csv", index=False)
```

In [132]:

```
submission_file_path = ("C:/Users/kushagra/Downloads/sample_submission.csv")
test_vectors=test_tfidf
submission(submission_file_path,clf_NB_TFIDF,test_vectors)
```

In [133]:

```
from IPython.display import FileLink, FileLinks
FileLink('submission.csv')
```

Out[133]:

submission.csv

Submission.csv file is submitted on kaggle and received a score of 0.79447 on public leaderboard.