Rapport disaster tweets mining

Oumaima KARMAM

ECOLE DE SCIENCE DE L'INFORMATION GDC

Introduction

Nous savons tous la puissance des réseaux sociaux et de Twitter spécialement. C'est l'un des principaux canaux de communication pour la plupart des gens sur terre, nous recevons la plupart de nos nouvelles quotidiennes à travers nos écrans via Twitter ces jours-ci...

Avec les téléphones intelligents entrant dans nos vies, notre fil d'actualité est immense mais il y a un problème : Comment saurions-nous si nous obtenons de vraies informations ? Et si on voulait séparer les situations réelles des données non pertinentes ? Nous avons de la chance! Nous avons des modèles NLP pour faire un travail lourd pour nous afin d'obtenir des informations distillées. Dans ce cahier, nous allons utiliser certaines approches PNL de base et courantes pour nous donner les résultats les plus précis.

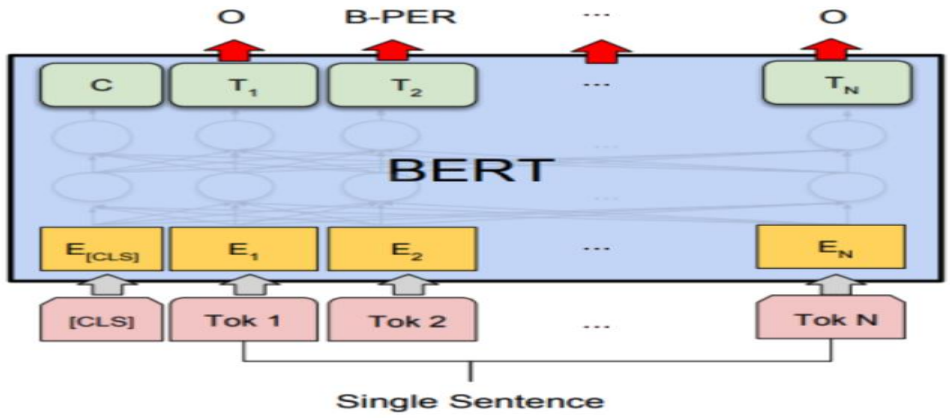
Conception

A. Model

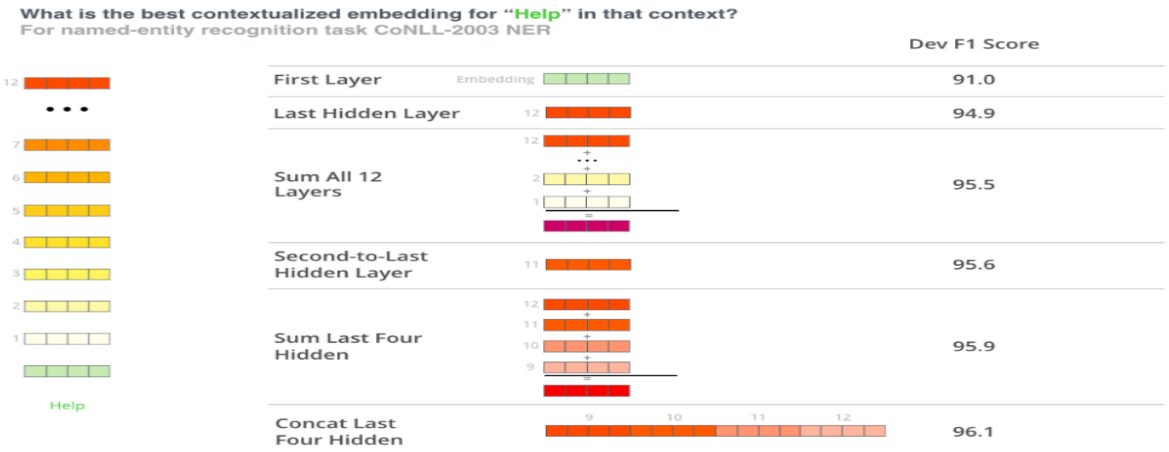
I. Fine-Tuning the BERT Model

1. Model Architecture :

We will be using bert-base-cased model, which has 12 layers of transformer encoders and ‘cased’ sequences.



In the init method of BertNer class, we create an object of BertModel, load the model weights using tf.train.Checkpoint. For Named Entity Recognition, we want the hidden states (the transformer output) of every input token. Hence we will be taking sequence_output from BertModel. Which layer should we actually choose for the best representation? The authors have shown the performance for different choices in their paper.



However, for our experimentation, we will be taking the representations from the last layer. These tokens are then fed to a dense layer with num_label units (number of tags present in the dataset in addition to 'CLS' and 'SEP' tags to get a prediction for every token. BERT used WordPiece tokenizer, which breaks some words into sub-words, in such cases we need only the prediction of the first token of the word. valid_ids is 0 for the sub-words, we use the valid_ids to get predictions only for words and first subword in case of split words.

II. Optimizer , Learning Rate Scheduler and Loss function:

We use AdamWeightDecay optimizer, with polynomial decay Learning Rate Schedule. We use the Sparse Categorical Crossentropy loss.

I. Custom training loop:

Input tokens, masks and labels are passed to the model. Loss and gradients are calculated and the parameters are updated using the optimizer.

II. Evaluation on valid dataset:

Load the model, preprocess the data to the required format. Predict on the valid dataset. We evaluate the performance of the model using precision, recall and f1 score. We use seqeval package. seqeval is a Python framework for sequence labeling evaluation. seqeval can evaluate the performance of chunking tasks such as named-entity recognition, part-of-speech tagging, semantic role labeling and so on. Classification report metric builds a text report showing the main classification metrics.

B. Conception ET réalisation:

I. Préparation du sujet :

Nous avons travaillé sur Tweets de catastrophe. Dans ce Notebook nous avons Analysé et exploré des données, en appliquant le modèle de BERT à l'aide de la bibliothèque Transformers avec Pytorch.

Notre notebook comprend : Le prétraitement du texte, la visualisation des données traitées par plusieurs méthodes comme la longueur des tweet, le nombre de mots, la longueur moyenne des mots, les ngrammes(« Les N-grammes sont des séquences continues de mots, de symboles ou de jetons dans un document »), etc. Nous avons nettoyé les données avant de les visualiser. Ensuite, Nous avons utilisé d'autres techniques d'analyse comme Word cloud, les NER, etc. pour nous donner différents angles de vue. Enfin, nous allons implémenter le modèle BERT pour faire de la tokenisation, de la classification et de la prédiction en utilisant des transformateurs.

II. Réalisation :

1. Préparer les données du texte :

- Premièrement nous allons charger beaucoup de bibliothèques que nous allons utiliser dans notre notebook :

- Les éléments les plus fondamentaux pour l'analyse exploratoire des données.
- Paquets de base pour le traitement de texte.
- Bibliothèques pour le prétraitement des textes.
- Chargement de quelques paquets sklearn pour la modélisation.
- Quelques paquets pour word cloud et le NER.
- Chargement des paquets pytorch

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#text processing.
import string
import re
#text preprocessing.
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
#sklearn packaces for modelling.

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation, NMF
from sklearn.metrics import f1_score, accuracy_score
#packages for word clouds and NER
from wordcloud import WordCloud, STOPWORDS
from collections import Counter, defaultdict
from PIL import Image
import spacy
!pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.2.5/en_core_web_sm-2.2.5.tar.gz
import en_core_web_sm
import random
import warnings
import time
import datetime
from matplotlib.ticker import MaxNLocator
import matplotlib.gridspec as gridspec
import matplotlib.patches as mpatches
#pytorch packages
import torch
from transformers import BertTokenizer, BertForSequenceClassification, AdamW, BertConfig, get_linear_schedule_with_warmup
from torch.utils.data import TensorDataset, random_split, DataLoader, RandomSampler, SequentialSampler
stop = set(stopwords.words('english'))
plt.style.use('fivethirtyeight')
sns.set(font_scale=1.5)
pd.options.display.max_columns = 250
pd.options.display.max_rows = 250
warnings.filterwarnings('ignore')
seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
```

- Chargeons nos données de train.csv et de test.csv

Eh bien... Nous avons des keywords, des locations, du texte et nos étiquettes cibles. Nous allons utiliser la fonction texte pour notre modélisation ici.

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

```
    Mounted at /content/drive
```

```
[ ] # Loading the train and test data for visualization & exploration.

    trainv = pd.read_csv('/content/drive/MyDrive/train.csv')
    testv = pd.read_csv('/content/drive/MyDrive/test.csv')
```

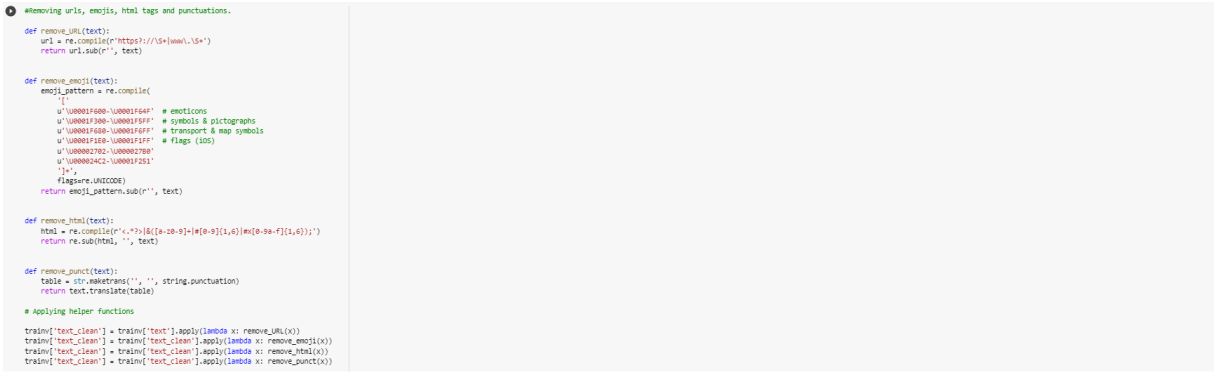
```
[ ] # Taking general look at the both datasets.

    display(trainv.sample(5))
    display(testv.sample(5))
```

	id	keyword	location	text	target
2644	3796	destruction	NaN	So you have a new weapon that can cause un-ima...	1
2227	3185	deluge	NaN	The f&@ing things I do for #GISHWHES Just...	0
5448	7769	police	UK	DT @georgegalloway: RT @Galloway4Mayor: ÛÏThe...	1
132	191	aftershock	NaN	Aftershock back to school kick off was great ...	0
6845	9810	trauma	Montgomery County, MD	in response to trauma Children of Addicts deve...	0

	id	keyword	location	text
142	449	armageddon	1996???????????	UNIVERSAL ORDER OF ARMAGEDDON http://t.co/3fY4mGm
2672	8915	snowstorm	Los Angeles	@BigBang_CBS ...wow...ok...um...that was like ...
2605	8682	sinkhole	New York, New York	The sinkhole that ate Brooklyn http://t.co/2lSr...
2515	8381	ruin	Monroe, OH	Don't ruin a good today by thinking about a ba...
958	3187	deluge	West Powelton, Philadelphia	I'm havin previous life flashbacks of when i l...

2. Nettoyage de texte :

Avant de visualiser nos données textuelles, Nous avons voulu les améliorer avec quelques fonctions d'aide générales pour nettoyer des choses comme : urls, emojis, balises html, ponctuations... Nous les ajouterons tous dans une colonne appelée 'text_clean', puis nous passerons aux étapes suivantes. Lorsque nous avons un texte plus propre, nous pouvons appliquer notre tokenizer pour diviser chaque mot en un token. Nous avons appliqué cette étape et les étapes suivantes à des colonnes individuelles pour montrer chaque étape de notre progression. Ensuite, nous transformons tous les mots en minuscules, puis nous supprimons les Stop words, nous utilisons donc les stops words de NLTK.

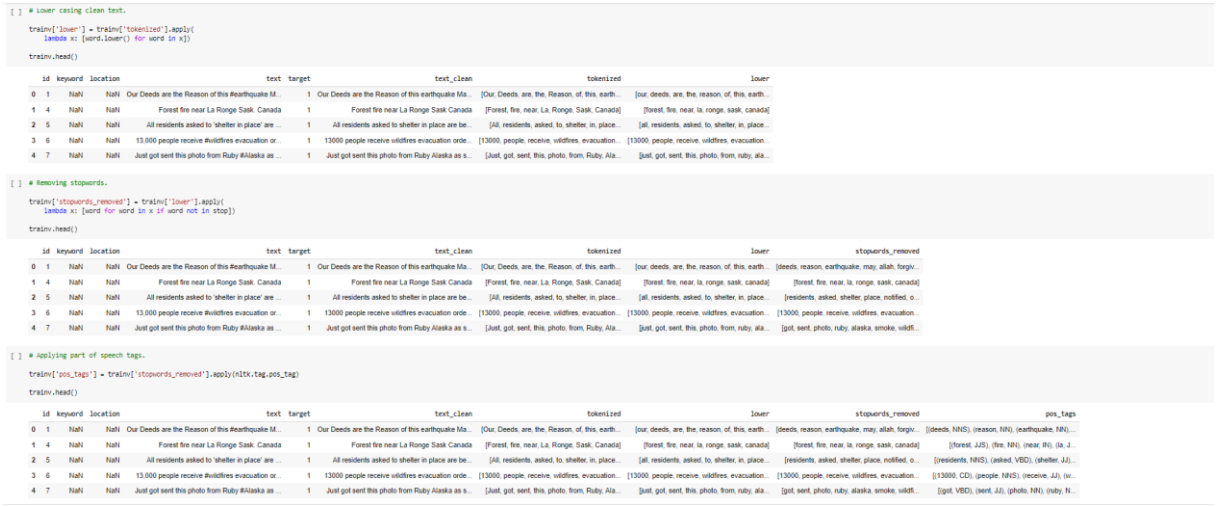


```
#Removing urls, emojis, html tags and punctuations.

def remove_URL(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    return url.sub(r'', text)


def remove_emoji(text):
    emoji_pattern = re.compile(
        '['
        u'\U0001F600-\U0001F64F'  # emoticons
        u'\U0001F300-\U0001F5FF'  # symbols & pictographs
        u'\U0001F680-\U0001F6FF'  # transport & map symbols
        u'\U0001F1E0-\U0001F1FF'  # flags (iOS)
        u'\U00002702-\U000027B0'
        u'\U000024C2-\U0001F251'
        ']+',
        flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)


def remove_html(text):
    html = re.compile(r'<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});')
    return re.sub(html, '', text)


def remove_punct(text):
    table = str.maketrans('', '', string.punctuation)
    return text.translate(table)

# Applying helper functions

trainv['text_clean'] = trainv['text'].apply(lambda x: remove_URL(x))
trainv['text_clean'] = trainv['text_clean'].apply(lambda x: remove_emoji(x))
trainv['text_clean'] = trainv['text_clean'].apply(lambda x: remove_html(x))
trainv['text_clean'] = trainv['text_clean'].apply(lambda x: remove_punct(x))
```

```
[ ] # Tokenizing the tweet base texts.

trainv['tokenized'] = trainv['text_clean'].apply(word_tokenize)

trainv.head()
```

	id	keyword	location	text	target	text_clean	tokenized
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	Our Deeds are the Reason of this earthquake Ma...	[Our, Deeds, are, the, Reason, of, this, earth...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	Forest fire near La Ronge Sask Canada	[Forest, fire, near, La, Ronge, Sask, Canada]
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	All residents asked to shelter in place are be...	[All, residents, asked, to, shelter, in, place...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	13000 people receive wildfires evacuation orde...	[13000, people, receive, wildfires, evacuation...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	Just got sent this photo from Ruby Alaska as s...	[Just, got, sent, this, photo, from, Ruby, Ala...

Après avoir supprimé ces mots, nous allons les lemmatiser mais pour cela, nous devons ajouter quelques étapes supplémentaires pour le faire correctement : Nous allons appliquer des balises de « speech tags » à notre texte (comme verbe, nom, etc.), puis nous les convertissons au format wordnet et enfin nous pouvons appliquer le lemmatiseur et le sauvegarder dans la colonne 'lemmatized'. Enfin, nous reconvertissons ces listes tokénisées en version stricte.



```
[ ] # Lower casing clean text.

trainv['lower'] = trainv['tokenized'].apply(
    lambda x: [word.lower() for word in x])

trainv.head()
```

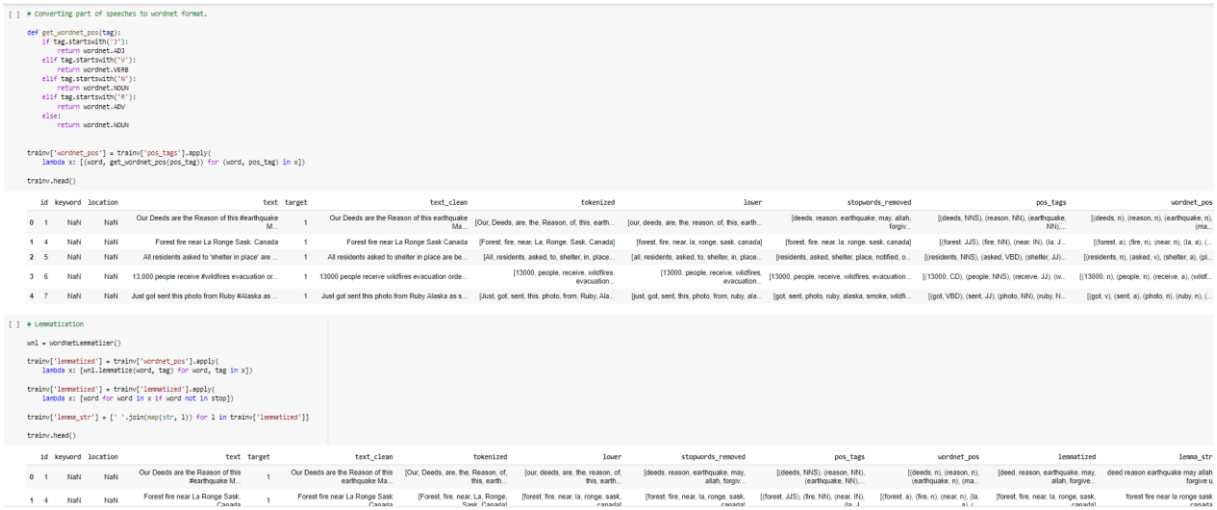
	id	keyword	location	text	target	text_clean	tokenized	lower
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	Our Deeds are the Reason of this earthquake Ma...	[Our, Deeds, are, the, Reason, of, this, earth...	[our, deeds, are, the, reason, of, this, earth...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	Forest fire near La Ronge Sask Canada	[Forest, fire, near, La, Ronge, Sask, Canada]	[forest, fire, near, la, ronge, sask, canada]
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	All residents asked to shelter in place are be...	[All, residents, asked, to, shelter, in, place...	[all, residents, asked, to, shelter, in, place...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	13000 people receive wildfires evacuation orde...	[13000, people, receive, wildfires, evacuation...	[13000, people, receive, wildfires, evacuation...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	Just got sent this photo from Ruby Alaska as s...	[Just, got, sent, this, photo, from, Ruby, Ala...	[just, got, sent, this, photo, from, ruby, ala...

```
[ ] # Removing stopwords.

trainv['stopwords_removed'] = trainv['lower'].apply(
    lambda x: [word for word in x if word not in stop])

trainv.head()
```

	id	keyword	location	text	target	text_clean	tokenized	lower	stopwords_removed
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	Our Deeds are the Reason of this earthquake Ma...	[Our, Deeds, are, the, Reason, of, this, earth...	[our, deeds, are, the, reason, of, this, earth...	[deeds, reason, earthquake, may, allah, forgiv...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	Forest fire near La Ronge Sask Canada	[Forest, fire, near, La, Ronge, Sask, Canada]	[forest, fire, near, la, ronge, sask, canada]	[forest, fire, near, la, ronge, sask, canada]
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	All residents asked to shelter in place are be...	[All, residents, asked, to, shelter, in, place...	[all, residents, asked, to, shelter, in, place...	[residents, asked, shelter, place, notified, o...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	13000 people receive wildfires evacuation orde...	[13000, people, receive, wildfires, evacuation...	[13000, people, receive, wildfires, evacuation...	[13000, people, receive, wildfires, evacuation...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	Just got sent this photo from Ruby Alaska as s...	[Just, got, sent, this, photo, from, Ruby, Ala...	[just, got, sent, this, photo, from, ruby, ala...	[got, sent, photo, ruby, alaska, smoke, wildfi...

```
[ ] # Applying part of speech tags.

trainv['pos_tags'] = trainv['stopwords_removed'].apply(nltk.tag.pos_tag)

trainv.head()
```

	id	keyword	location	text	target	text_clean	tokenized	lower	stopwords_removed	pos_tags
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	Our Deeds are the Reason of this earthquake Ma...	[Our, Deeds, are, the, Reason, of, this, earth...	[our, deeds, are, the, reason, of, this, earth...	[deeds, reason, earthquake, may, allah, forgiv...	[(deeds, NNS), (reason, NN), (earthquake, NN),...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	Forest fire near La Ronge Sask Canada	[Forest, fire, near, La, Ronge, Sask, Canada]	[forest, fire, near, la, ronge, sask, canada]	[forest, fire, near, la, ronge, sask, canada]	[(forest, JJS), (fire, NN), (near, IN), (la, J...
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	All residents asked to shelter in place are be...	[All, residents, asked, to, shelter, in, place...	[all, residents, asked, to, shelter, in, place...	[residents, asked, shelter, place, notified, o...	[(residents, NNS), (asked, VBD), (shelter, JJ),...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	13000 people receive wildfires evacuation orde...	[13000, people, receive, wildfires, evacuation...	[13000, people, receive, wildfires, evacuation...	[13000, people, receive, wildfires, evacuation...	[(13000, CD), (people, NNS), (receive, JJ), (w...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	Just got sent this photo from Ruby Alaska as s...	[Just, got, sent, this, photo, from, Ruby, Ala...	[just, got, sent, this, photo, from, ruby, ala...	[got, sent, photo, ruby, alaska, smoke, wildfi...	[(got, VBD), (sent, JJ), (photo, NN), (ruby, N...



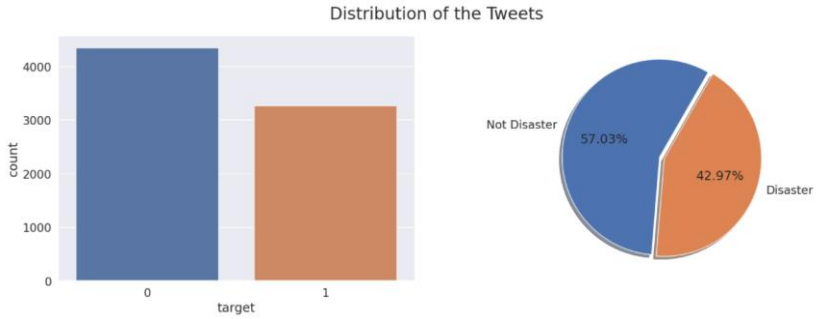
- Donc, en gros, ce que nous avons fait est :
- Suppression des urls, emojis, balises html et ponctuations,
- On a tokenisé les textes de base des tweets,
- On a mis en minuscules les textes propres,
- Suppression des stop words,
- Appliquer les balises de speech tags,
- Conversion des parties de discours au format wordnet,
- Application du lemmatiseur de mots,
- Convertir à nouveau le texte tokénisé en chaîne de caractères.

3. Visualisation des données :

Maintenant nous pouvons commencer à visualiser les données pour voir si nous pouvons trouver des relations visibles entre les classes de tweet.

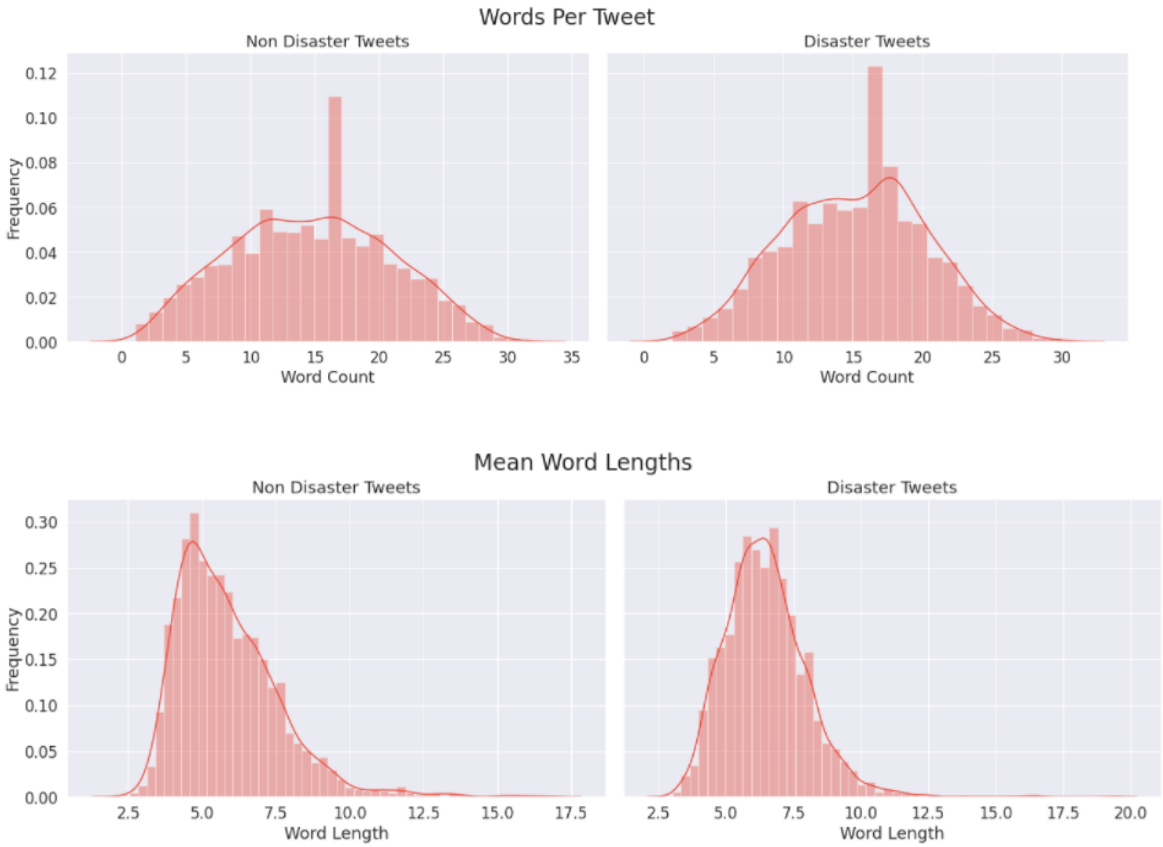
- Distribution des cibles

Lorsque nous vérifions nos variables cibles et regardons comment elles se répartissent, nous pouvons dire que ce n'est pas mauvais. Il n'y a pas d'énorme différence entre les classes, nous pouvons dire que c'est un bon signe pour la modélisation.



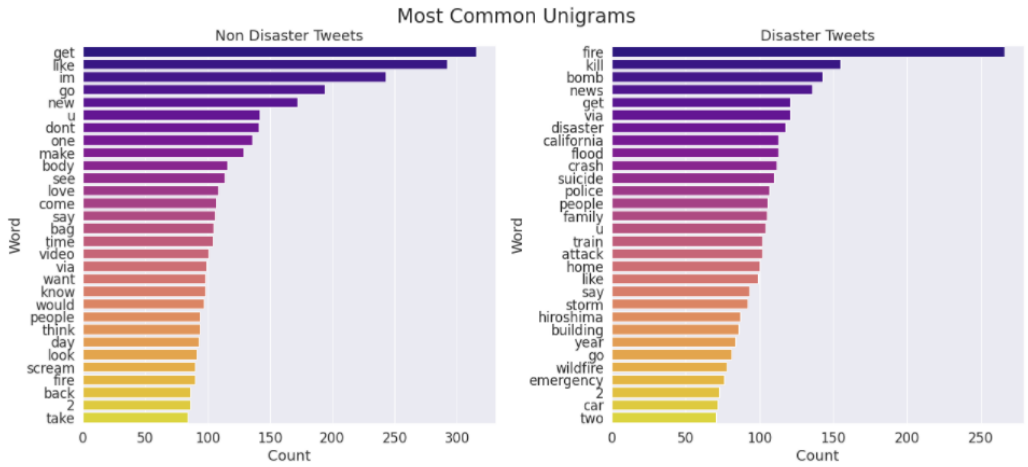
- Longueur des tweets

Il semble que les tweets liés à une catastrophe soient plus longs que les tweets non liés à une catastrophe en général. Nous pouvons supposer que les tweets plus longs sont plus susceptibles d'être liés à des catastrophes, mais ce n'est qu'une hypothèse et cela pourrait ne pas être vrai...



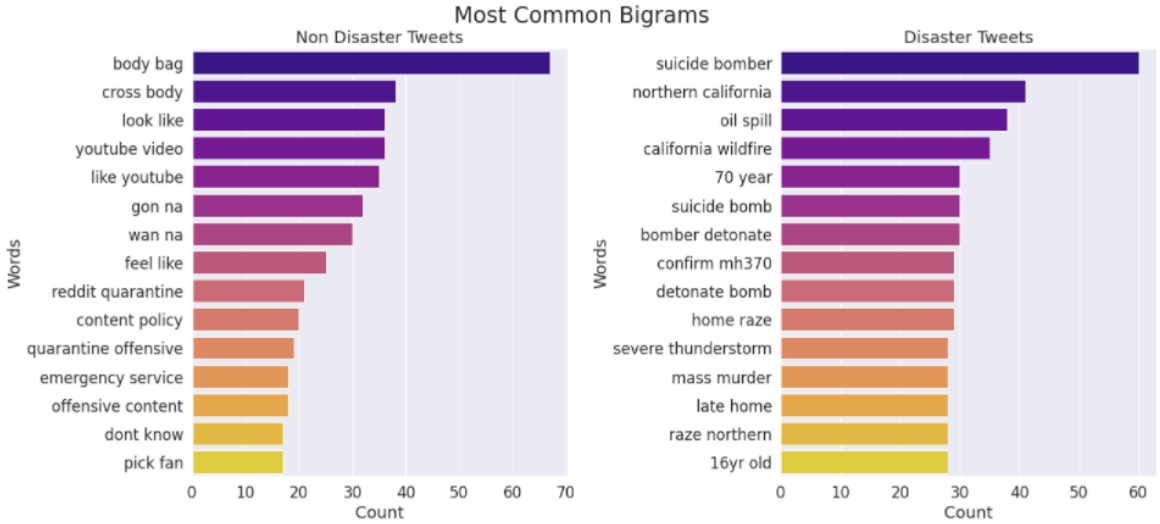
- Les mots les plus courants

Les tweets de catastrophe ont des mots comme feu, tuer, bombe indiquant des catastrophes. Alors que ceux qui ne le sont pas semblent plutôt génériques.



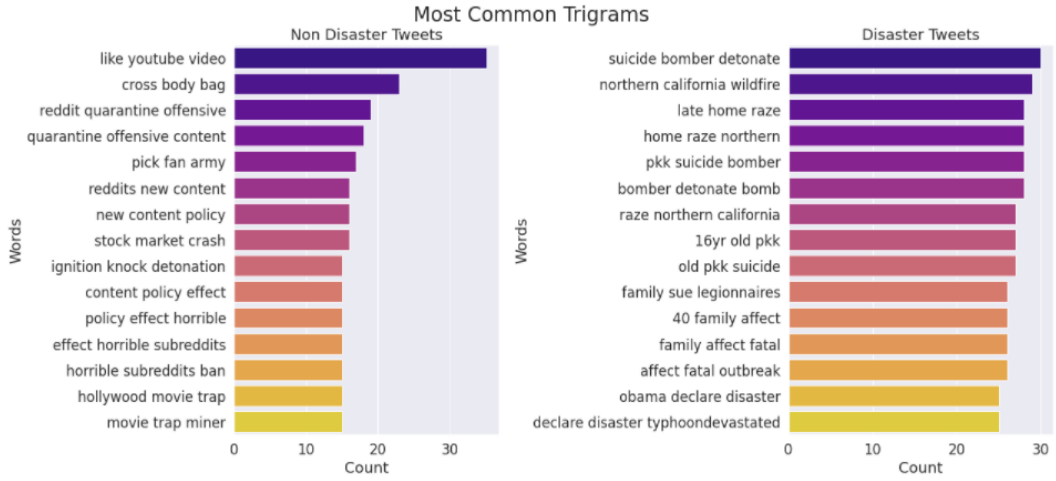
- Les bigrammes les plus courants

Examinons cette fois les bigrammes, qui sont des séquences de deux mots adjacents. Encore une fois, il est assez évident de séparer deux classes si elles sont liées à une catastrophe ou non. Il existe quelques bigrammes déroutants dans les catégories non liées aux catastrophes, comme sac mortuaire, service d'urgence, etc., qui nécessitent des recherches plus approfondies, mais nous en resterons là puisque nous avons obtenu ce que nous recherchions en général.



- Trigrammes les plus courants

Très bien ! Les choses sont beaucoup plus claires avec les séquences de 3 mots. Les sacs de corps qui prêtaient à confusion étaient des sacs de corps croisés (qui les utilise de nos jours de toute façon !), Quoi qu'il en soit, nous pouvons voir que les catastrophes sont maintenant hautement séparables de celles qui ne le sont pas, ce qui est génial !



Quelques analyses supplémentaires :

Dans cette partie, nous allons appliquer d'autres techniques d'analyse pour mieux comprendre notre jeu de données.

- Détermination des thèmes

Nous allons utiliser une méthode appelée Factorisation Matricielle Non Négative (NMF) pour voir si nous pouvons obtenir des sujets définis à partir de notre matrice TF-IDF. De cette façon, TF-IDF diminuera l'impact des mots à haute fréquence, ce qui nous permettra d'obtenir des sujets plus spécifiques.

Lorsque nous inspectons nos dix premiers sujets, nous devons faire preuve d'un peu d'imagination pour nous aider à les comprendre. En fait, ils sont assez distincts, je dirais que les sujets de catastrophes sont beaucoup plus clairs à lire, nous pouvons voir les sujets directement en les regardant, tandis que les sujets de non catastrophe sont plus personnels...

```
def display_topics(text, no_top_words, topic):

    """ A function for determining the topics present in our corpus with nmf """

    no_top_words = no_top_words
    tfidf_vectorizer = TfidfVectorizer(
        max_df=0.90, min_df=25, max_features=5000, use_idf=True)
    tfidf = tfidf_vectorizer.fit_transform(text)
    tfidf_feature_names = tfidf_vectorizer.get_feature_names()
    doc_term_matrix_tfidf = pd.DataFrame(
        tfidf.toarray(), columns=list(tfidf_feature_names))
    nmf = NMF(n_components=10, random_state=0,
              alpha=.1, init='nndsvd').fit(tfidf)
    print(topic)
    for topic_idx, topic in enumerate(nmf.components_):
        print('Topic %d:' % (topic_idx+1))
        print(' '.join([tfidf_feature_names[i]
                        for i in topic.argsort()[:-no_top_words - 1:-1]]))


display_topics(lis[0], 10, 'Non Disaster Topics\n')
```

```
Non Disaster Topics

Topic 1:
get lol blow good bomb first day demolish someone play
Topic 2:
like video youtube look feel back fire fatality sink mudslide
Topic 3:
im traumatise still disaster na gon attack drown dead weapon
Topic 4:
new emergency full read content post quarantine many storm re
Topic 5:
body bag cross shoulder woman full lady read ebay really
Topic 6:
dont one see know come say make want think fire
Topic 7:
scream fuck phone face good song loud hit baby time
Topic 8:
via youtube god change obliteration news story stop service video
Topic 9:
go content quarantine many explode make reddit let top deluge
Topic 10:
love crush collide woman much death military armageddon would check
```

- Word cloud

Les nuages de mots sont une approche populaire dans les tâches NLP. Nous allons utiliser la bibliothèque exactement conçue pour cela, appelée "WordCloud", nous avons également voulu la masquer avec la forme du logo twitter et des couleurs grises.

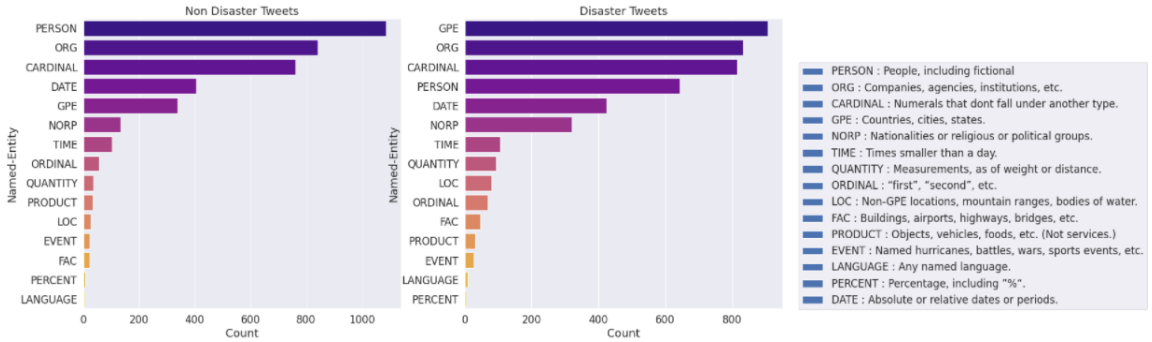
- Chargement du modèle de l'extraction des données

```
[ ]  # Loading NER.
     nlp = en_core_web_sm.load()
```

```
[ ]  def plot_named_entity_barchart(text):

         """A function for extracting named entities and comparing them"""

         def _get_ner(text):
             doc = nlp(text)
             return [X.label_ for X in doc.ents]

         ent = text.apply(lambda x: _get_ner(x))
         ent = [x for sub in ent for x in sub]
         counter = Counter(ent)
         count = counter.most_common()

         x, y = map(list, zip(*count))
         sns.barplot(x=y, y=x)
```

- Visualisation de la fonction plot named entity barchart :

Cette fonction nous a permis de comparer les nom d'entités les plus communs dans les tweets catastrophiques et non catastrophiques



On remarque que le mot le plus utilisé en cas d'un désastre c'est GPE qui signifie les pays et les villes.

- Vérification du GPU

Vérification de la présence d'un GPU dans l'ordinateur pour l'utiliser. Nous n'avons pas un GPU donc nous avons utilisé CPU

```
[ ]
    if torch.cuda.is_available():

        # Tell PyTorch to use the GPU.

        device = torch.device('cuda')


        print('There are %d GPU(s) available.' % torch.cuda.device_count())

        print('We will use the GPU:', torch.cuda.get_device_name(0))

    # If not...

    else:
        print('No GPU available, using the CPU instead.')
        device = torch.device('cpu')
```

```
No GPU available, using the CPU instead.
```

- Chargement et affichage des données qui vont être trainées

```
[ ]  # Loading the data for modelling.

     train = pd.read_csv('/content/drive/MyDrive/train.csv')
     test = pd.read_csv('/content/drive/MyDrive/test.csv')

     print(f'Number of training tweets: {train.shape[0]}\n')
     print(f'Number of training tweets: {test.shape[0]}\n')

     display(train.sample(10))
```

```
Number of training tweets: 7613

Number of training tweets: 3263
```

	id	keyword	location	text	target
4094	5819	hail	Calgary, AB	@HeyItsEpark heavy rain and hail	1
3506	5010	explosion	Kauai, Hawaii	The government is concerned about the populati...	0
6141	8760	siren	Michel Delving.	Apparently they're going to have a WW2 siren t...	0
657	951	blaze	Rio de Janeiro	I liked a @YouTube video from @iamrrsb http://...	0

- La tokenization avec bert tokenizer :

```
[ ]  # Tokenizing the combined text data using bert tokenizer.

     tokenizer = BertTokenizer.from_pretrained('bert-large-uncased', do_lower_case=True)
```

```
[ ]  # Print the original tweet.

     print(' Original: ', combined[0])

     # Print the tweet split into tokens.

     print('Tokenized: ', tokenizer.tokenize(combined[0]))

     # Print the sentence mapped to token ID's.

     print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(combined[0])))
```

```
 Original:  Our Deeds are the Reason of this #earthquake May ALLAH Forgive us all
Tokenized:  ['our', 'deeds', 'are', 'the', 'reason', 'of', 'this', '#', 'earthquake', 'may', 'allah', 'forgive', 'us', 'all']
Token IDs:  [2256, 15616, 2024, 1996, 3114, 1997, 2023, 1001, 8372, 2089, 16455, 9641, 2149, 2035]
```

- On a essayé d'identifier le maximum d'une phrase par le code

```
[ ]  max_len = 0
     for text in combined:

         # Tokenize the text and add `[CLS]` and `[SEP]` tokens.

         input_ids = tokenizer.encode(text, add_special_tokens=True)

         # Update the maximum sentence length.

         max_len = max(max_len, len(input_ids))

     print('Max sentence length: ', max_len)
```

```
Max sentence length:  84
```

Nous avons trouvé que la largeur maximale d'une phrase est 84

- Après la tokenization des tweets, nous avons essayé de mapper les tokens avec leurs ids, et puis nous avons entraîné les données avec TensorDataset qui fait une génération arbitraire des inputs de la dataset

```
[ ]  # Combine the training inputs into a TensorDataset.

     dataset = TensorDataset(input_ids, attention_masks, labels)

     # Create a 80-20 train-validation split.

     # Calculate the number of samples to include in each set.

     train_size = int(0.8 * len(dataset))
     val_size = len(dataset) - train_size

     # Divide the dataset by randomly selecting samples.

     train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

     print('{:>5,} training samples'.format(train_size))
     print('{:>5,} validation samples'.format(val_size))
```

```
6,090 training samples
1,523 validation samples
```

- Après l'identification de la taille du batch, nous avons défini notre model Bert pour faire la classification par le code suivant :

```
[ ] # Load BertForSequenceClassification, the pretrained BERT model with a single linear classification layer on top.

    model = BertForSequenceClassification.from_pretrained(
        'bert-large-uncased',
        num_labels = 2,
        output_attentions = False,
        output_hidden_states = False,
    )


    model.to(device)
```

```
Some weights of the model checkpoint at bert-large-uncased were not used when initializing BertForSequenceClassification: ['cls.predictions.transform.LayerNorm
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializi
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-large-uncased and are newly initialized: ['classifier.weigh
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 1024, padding_idx=0)
      (position_embeddings): Embedding(512, 1024)
      (token_type_embeddings): Embedding(2, 1024)
      (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
```

- Extraction des paramètres du modèle :

```
[ ] The BERT model has 393 different named parameters.

    ==== Embedding Layer ====

    bert.embeddings.word_embeddings.weight                  (30522, 1024)
    bert.embeddings.position_embeddings.weight               (512, 1024)
    bert.embeddings.token_type_embeddings.weight               (2, 1024)
    bert.embeddings.LayerNorm.weight                           (1024,)
    bert.embeddings.LayerNorm.bias                             (1024,)

    ==== First Transformer ====

    bert.encoder.layer.0.attention.self.query.weight        (1024, 1024)
    bert.encoder.layer.0.attention.self.query.bias             (1024,)
    bert.encoder.layer.0.attention.self.key.weight          (1024, 1024)
    bert.encoder.layer.0.attention.self.key.bias               (1024,)
    bert.encoder.layer.0.attention.self.value.weight        (1024, 1024)
    bert.encoder.layer.0.attention.self.value.bias             (1024,)
    bert.encoder.layer.0.attention.output.dense.weight      (1024, 1024)
    bert.encoder.layer.0.attention.output.dense.bias           (1024,)
    bert.encoder.layer.0.attention.output.LayerNorm.weight     (1024,)
    bert.encoder.layer.0.attention.output.LayerNorm.bias       (1024,)
    bert.encoder.layer.0.intermediate.dense.weight          (4096, 1024)
    bert.encoder.layer.0.intermediate.dense.bias               (4096,)
    bert.encoder.layer.0.output.dense.weight                (1024, 4096)
    bert.encoder.layer.0.output.dense.bias                     (1024,)
    bert.encoder.layer.0.output.LayerNorm.weight               (1024,)
    bert.encoder.layer.0.output.LayerNorm.bias                 (1024,)

    ==== Output Layer ====

    bert.pooler.dense.weight                                (1024, 1024)
    bert.pooler.dense.bias                                     (1024,)
    classifier.weight                                        (2, 1024)
    classifier.bias                                                (2,)
```

On trouve que nous avons des paramètres dans Embedding layer, dans le first transformer, et dans le Output layer.

- L'optimisation : Nous avons travaillé avec l'algorithme Adam qui est un excellent optimisateur pour donner une meilleur optimisation du training des données.

```
[ ]
    optimizer = AdamW(model.parameters(),
                      lr = 6e-6,
                      eps = 1e-8
                    )
```

- Calcul de l'accuracy, f1 score

epoch	Training Loss	Valid. Loss	Valid. Accur.	Val_F1	Training Time	Validation Time
1	0.50	0.45	0.81	0.77	0:02:43	0:00:13
2	0.38	0.42	0.83	0.78	0:02:42	0:00:13
3	0.33	0.43	0.83	0.78	0:02:42	0:00:13

Nous avons aussi calculé l'accuracy du modèle qui fait la classification des tokens des tweets.

Pour le 1er epoch le training loss, qui signifie comment le modèle convient les données traînées est de 0.5, par contre la validation loss est de 0.45, ce qui signifie que le modèle convient les nouvelles données d'un score de 0.45.