Projet n° 7 : Réalisez une preuve de concept

Parcours ingénieur machine learning OPENCLASSROOMS

Céline Mendola

Introduction

Le Natural language processing ou NLP est un domaine de l'intelligence artificielle ayant pour but de traiter, comprendre et générer le langage humain. Les traducteurs automatiques, les correcteurs d'orthographe et de grammaire, les moteurs de recherche, les filtres spams, les chatbots sont quelques exemples d'application de méthodes de NLP complètement intégrés dans notre quotidien.

Les filtres spams font partie d'un des champs du NLP appelé classification de texte, on peut citer également la détection de fake news ou l'analyse des émotions. Nous verrons par la suite une étude de cas sur la classification de texte mais soulignons que les techniques aux différentes applications du NLP sont communes.

Le NLP est un sujet en pleine effervescence qui a connu de grandes avancées à partir des années 2 000. Avant cela, la 1ère expérience de système de traduction automatique 'The Georgetown-IBM experiment' a vu le jour en 1954. Pendant plus de 30 ans après cette expérience, les systèmes se baisaient sur des règles "au cas par cas" : Par exemple, pour un système de classification d'avis sur des produits, une règle au cas par cas serait de catégoriser une phrase comme positive si elle contient le mot 'efficace' mais négative si elle contient en plus une négation… mais comment classifier le commentaire "le produit n'est pas seulement bon marché mais il est aussi efficace". Ces règles au cas par cas ne permettaient pas de généralisation. Puis, à la fin des années 80 on a utilisé des méthodes statistiques et des modèles de machine learning. Enfin, depuis les années 2 000 on utilise le deep learning pour le NLP. Les réseaux de neurones offrent une méthode bien plus efficace mais aussi plus simple.

Interrogeons-nous sur les challenges du NLP. Tout d'abord le langage humain est complexe et peut être ambigu, c'est-à-dire que son interprétation n'est pas toujours unique. Une même phrase peut avoir plusieurs sens par exemple "j'ai vu une fille avec un téléscope" (qui avait le téléscope, moi, ou la fille que j'ai vu?) ou encore "j'ai une souris chez moi", parle-t-on de l'animal ou de l'accessoire informatique? Enfin "Je suis la reine d'Angleterre" (est-ce que c'est du sarcasme?). Le contexte joue un rôle clé dans l'interprétation. Nous verrons comment certains modèles peuvent le prendre en compte. Ensuite il faut une méthode pour représenter numériquement les phrases. Puis, contrairement au traitement d'images avec le deep learning, les phrases dans un corpus de texte n'ont pas en général la même longueur / dimension il faut donc réussir à traiter ces variations. Enfin, le traitement du langage naturel est un problème discret. La modification d'une lettre dans une phrase peut en changer son sens (ex Jolie voiture" et "Jolie toiture") et la différence de mémoire pour l'ordinateur n'est de qu'un seul bit. Pour une image, si on change la valeur des pixels sur un bit, la couleur changera peut-être mais son interprétation, elle, ne changera pas.

Pour découvrir comment les recherches ont pu faire face à tous ces challenges, nous ferons dans une première partie un état de l'art des différentes techniques du NLP. Puis dans une seconde partie, nous appliquerons ces techniques à un problème de classification de texte. Nous réaliserons un modèle qui classifie les émotions des tweets parmi la colère, la peur, la joie et la tristesse. Nous nous appuierons notamment sur les méthodes exposées dans cet article de recherche *Emotion and sentiment analysis of tweets using BERT* publié en mars 2021.

Les techniques du NLP

Nettoyer les données textuelles

Comme dans tout projet de Machine learning, la première étape consiste à nettoyer notre jeu de données. Sur les données textuelles, certaines opérations de nettoyage que l'on peut réaliser sont :

- supprimer les espaces multiples, tabulations, ponctuations, les emojis.
- supprimer les stopwords c'est-à-dire des mots très récurrents ou qui n'apportent pas beaucoup d'informations.
- Certains package comme spell-checker permettent de corriger les fautes d'orthographes.
- La lemmatization ou le stemming permettent d'avoir une racine commune pour les différentes variations d'un même mot.
- Enlever la case du texte.

Ces différents traitements à réaliser peuvent varier d'un problème à un autre et d'un modèle à un autre.

Modéliser les mots

La méthode Bags of Words

Le Bag of words est une technique permettant de vectoriser le corpus de texte afin de le donner en entrée de nos algorithmes.

- Une méthode consiste à calculer le nombre d'occurrences des mots dans chaque document (phrases, posts ect.. du corpus de texte). On obtient une matrice des "term frequency".

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

BOW on Surfin' Bird

L'inconvénient de cette méthode est que pour des tokens qui n'apparaissent que dans très peu de documents on aura une colonne quasi remplie de zeros, et donc on risque d'obtenir une matrice "sparse".

- Pour donner plus de poids aux tokens qui apparaissent dans peu de documents et moins de poids aux mots très fréquents, une seconde méthode consiste à créer la matrice "TFIDF".

Pour un problème de classification de texte, utiliser l'une de ces vectorisation avec un modèle de machine learning de classification (KNN, SVM, Naive Bayes, Random Forest …)

peut s'avérer assez efficace. Cependant on voit que la méthode des bag of words ne conserve pas de lien sémantique entre les mots.

Les word embeddings

En 2013, une nouvelle technique de vectorization des mots fait son apparition: les word embeddings. Cette méthode permet de garder les similarités sémantiques entre les mots. Par exemple, observons une représentation 2D des vecteurs obtenus avec l'une des méthode nommée word2vec :

Word similarity according to Word2vec

On remarque que les mots ayant une similarité sémantique sont proches sur le graphe. Notons qu'une représentation 2d ne peut récupérer toute l'information de vecteurs de bien plus grande dimension.
De plus, les word embeddings permettent de conserver les analogies entre les mots. Par exemple, le mot "femme" est au mot "reine" ce que le mot "homme" est au mot "roi". Ou bien, "Paris" est à "France" ce que "Berlin" est à "Allemagne". En écriture vectorielle,

$$\overrightarrow{queen} - \overrightarrow{woman} = \overrightarrow{king} - \overrightarrow{man}$$
$$\overrightarrow{France} - \overrightarrow{Paris} = \overrightarrow{Germany} - \overrightarrow{Berlin}$$

Plusieurs méthodes existent pour créer ces words embeddings: word2vec, glove, fastText, negative sampling. Détaillons un peu l'idée derrière les deux premières.

- Word2vec (2013. Mikolov et al. Efficient estimation of word representations in vector spaces, Google)

Idée : entraîner un modèle supervisé sur un corpus de texte qui, avec un petit réseau de neurones, à partir d'un mot pris aléatoirement dans une phrase (target) essaye de prédire un mot qui lui est proche dans la phrase.

exemple : “Je veux un verre de jus d’orange pour aller avec mes céréales”

target	context word to predict
orange	jus
orange	pour
orange	céréales

Les vecteurs des word embeddings correspondront aux poids appris par ce modèle. La méthode est coûteuse en calculs.

- Glove (Pennington et. al 2014: GloVe : Global vector representation, Stanford)

Idée: minimiser une fonction de la forme : $J(\theta, e) = \sum_{i,j=1}^{|V|} f(X_{ij})(\theta_i^t e_j + b_i + b_j' - log(X_{ij}))^2$ où |V| est la taille du vocabulaire et X_{ij} est le nombre de fois où le mot i apparaît proche du mot j dans le corpus.

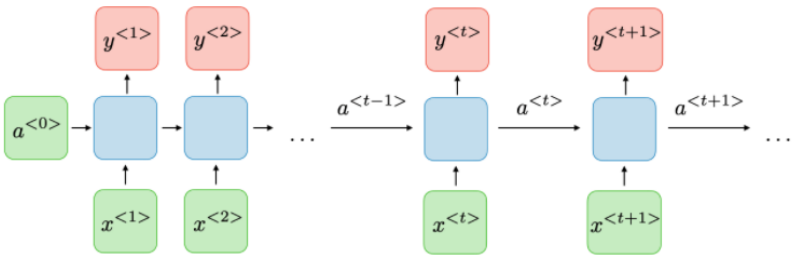
Des modèles pré-entraînés sur des données textuelles (twitter, wikipedia, gigaword…) sont disponibles. On peut les utiliser directement pour nos modèles, en les ré-entrainant si nécessaire sur notre corpus de texte et les adapter au mieux.

Remarquons que la nature des textes peut amener à certains biais de genre, d’ethnicité, d’orientation sexuelle, d’âge ect dans les word embeddings. Des méthodes pour réduire ces biais sont exposés dans l’article Bolukbasi et al. 2016, *Man is to computer programmer as woman is to homemaker ?*

Les réseaux de neurones récurrents

Généralités

Les réseaux de neurones récurrents (RNN pour recurrent neural networks) sont un type de réseau de neurones adaptés aux problèmes séquentiels. Ils permettent de traiter des entrées aux dimensions variables.

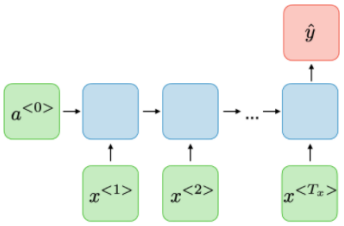


For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

Nous aurons plusieurs configurations selon le problème sur le nombre de sorties. Par exemple, pour une classification de texte on aura une configuration “many-to-one”:

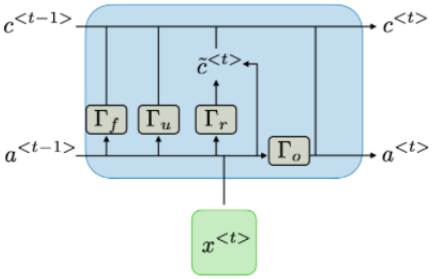


L'un des inconvénients des RNN, est, qu'en raison du nombre de couches on peut faire face au problème du "vanishing gradient" ou du "exploding gradient" qui correspondent respectivement au cas où le gradient devient très petit est stagne ou bien qu'il explose. On peut alors avoir des difficultés à capter les dépendances entre des termes éloignés dans les phrases.
Par exemple, garder la cohérence grammaticale entre le début et la fin d'une phrase comme "Le chat, qui avait déjà mangé …. , s'est endormi." Si on passe le sujet au pluriel cela donnera: "Les chats, qui avaient déjà mangé …. , se sont endormis."

Le cas du "exploding gradient" peut être réglé avec le clipping gradient que nous ne détaillerons pas ici. Pour faire face au problème du vanishing gradient, intéressons-nous aux réseaux de neurones LSTM.

Les LSTM

Les LSTM (long short term memory units) sont un type de réseau de neurones qui permet de faire face au problème du "vanishing gradient".



A chaque étape t, nous avons une cellule mémoire $c^{<t>}$ qui est actualisée qui va permettre de gérer les informations précédentes

Pour cela, on utilise des ponts (forgate gate, update gate, relevance gate et output gate) définis sous la forme $\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$ où σ est la fonction sigmoid. Ils sont mis à jour lors de la backpropagation. Le candidat à la nouvelle cellule mémoire $c^{tilde<t>}$,la cellule mémoire $c^{<t>}$ et la couche d'activation sont définis respectivement par:

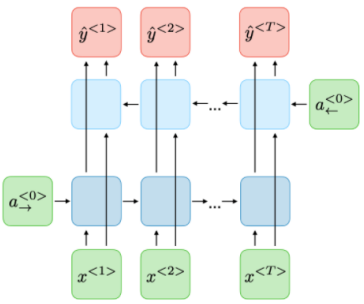
$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

Les réseaux de neurones bidirectionnels

Pour les termes placés en début de phrase, il peut être plus intéressant d'avoir les informations en partant de la fin de la phrase. Une solution est de créer des couches d'activation "backward" c'est à dire allant dans l'autre sens.



Les Transformers

Présentation

Les Transformer network sont un type de réseau de neurones qui ont complètement révolutionné le domaine du NLP. Ils se basent sur deux concepts clés :

- le mécanisme du self attention
- le fait de traiter les entrées non plus de manière séquentielle mais en parallèle.

Les Transformer ont été introduits dans l'article "Vaswani et al; 2017 Attention is all you need".

Self attention

Détaillons le mécanisme du "self attention" . Il permet en fait d'avoir une représentation de chaque mot dans son contexte, ce que ne permettent pas en général les word embeddings.

Par exemple, dans les phrases "Jane plays basketball" ou "Hamlet is a play of Shakespeare", le mot *play* n'a pas le même sens pourtant le word embedding à lui seul ne permet pas de faire une différence de contexte.

L'idée du mécanisme self attention est d'avoir une nouvelle représentation de chaque mot une phrase, en fonction des autres mots. Ainsi on captera au maximum le contexte.

Dans une phrase composée de 5 mots $x^{<1>}$, $x^{<2>}$, $x^{<3>}$, $x^{<4>}$ et $x^{<5>}$, on veut une nouvelle représentation des mots donnée respectivement par $A^{<1>}$, $A^{<2>}$, $A^{<3>}$, $A^{<4>}$ et $A^{<5>}$.

A chaque mot $x^{<l>}$, on, associe des valeurs $q^{<l>}$ $(query)$, $k^{<l>}$ (key) et $v^{<l>}(value)$ avec :

$$q^{<l>} = W^q * x^{<l>},\ k^{<l>} = W^k * x^{<l>} \text{ et } v^{<l>} = W^v * x^{<l>}$$

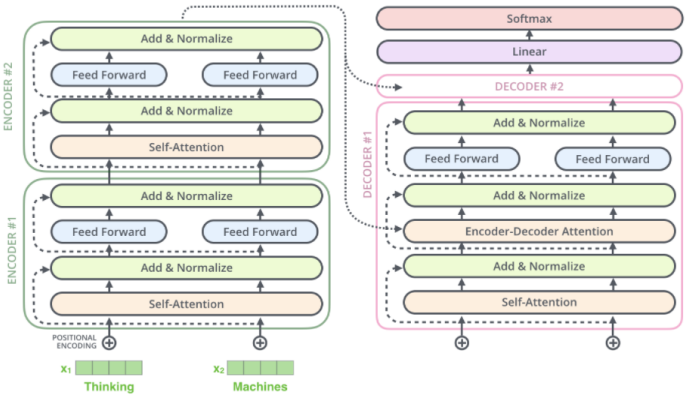
Les matrices W seront apprises par le modèle.

La valeur de la nouvelle représentation correspondant à $x^{<l>}$ est donnée par:

$$A^{<l>} = \sum_{i=1}^{5} \frac{exp(q^{<l>}.k^{<i>})}{\sum_{1 \le j \le 5} exp(q^{<l>}.k^{<j>})} v^{<i>}$$

Intuitivement, on cherche à représenter un lien question/réponse entre le mot l et les autres mots.

Architecture d'un transformer network



BERT

BERT (Bidirectional Encoder Representations from Transformers) est un modèle transformer pré-entraîné développé par google en 2018. Plusieurs modèles comme BERT-base et BERT-large sont disponibles, ils diffèrent par leur nombre de paramètres.
L'architecture de BERT est constituée de plusieurs couches de transformer encoder afin de réussir à contextualiser les entrées. Pour entraîner ce modèle, on s'est donné un problème de ML supervisé qui constituait à prédire chaque mot caché dans une phrase.

D'autres modèles transformers pré-entrainés existent comme ALBERT, XLnet ou GPT-2.

Dans cette première partie, nous avons pu voir les techniques les plus récentes utilisées en NLP. Dans la partie suivante, nous allons mettre en œuvre ces techniques afin de classifier des émotions.

Classification des émotions

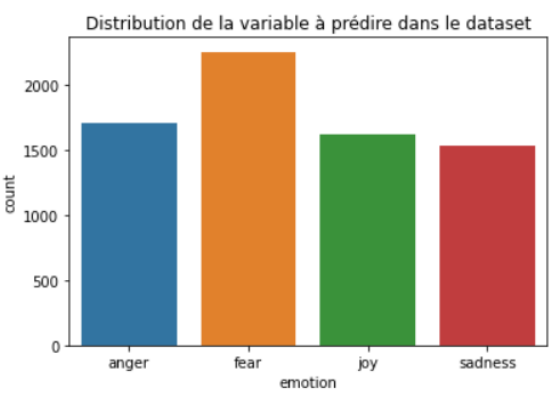
Présentation

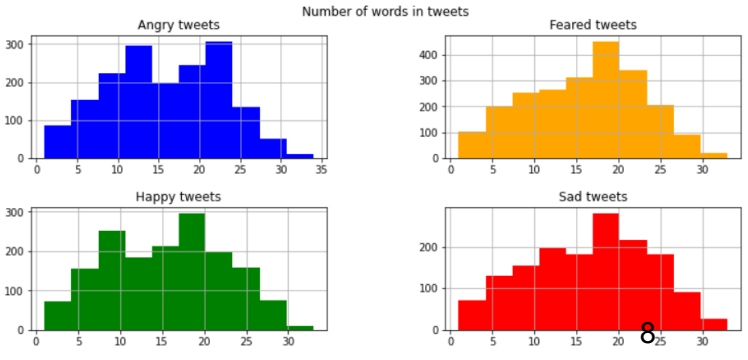
Nous nous penchons sur une étude de mars 2021 dans cet article de recherche initulé *Emotion and sentiment analysis of tweets using BERT*
Les auteurs exposent leur méthode avec BERT pour détecter des sentiments (positifs/négatif/neutre) et émotions (colère/peur/joie/tristesse) sur des posts twitter. Dans le cadre de notre étude, on se limitera à la détection des émotions.

Les données sont récupérées à cette adresse :
https://saifmohammad.com/WebPages/TweetEmotionIntensity-dataviz.html

Nettoyage et analyses des données

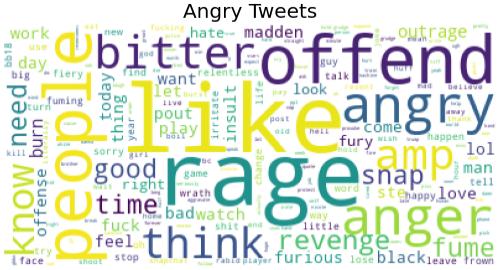
Nous avons un dataset de 7102 lignes
On effectue un nettoyage "de base" suivant pour faire nos analyses, ainsi que pour le modèle baseline et l'application de réseau de neurones récurrents.
On supprime les tweets qui contiennent uniquement des caractères "non ASCII" , les tweets très courts (avec un nombre de caractères <=2), les mentions, les URLS, les retweets et on lemmatize.
On ajoutera quelques traitements selon les modèles.
Pour les analyses et le modèle baseline à suivre, on enlève en plus la ponctuation et on passe le texte en minuscule.





Grâce aux graphiques ci-dessus, on peut voir que l'étiquette "fear" est un peu plus présente. Les distributions pour chaque étiquette du nombre de mots par tweet sont similaires, les paires colère/joie et peur/tristesse ont une distribution particulièrement similaire. Réalisons des word cloud afin d'observer les mots les plus présents.

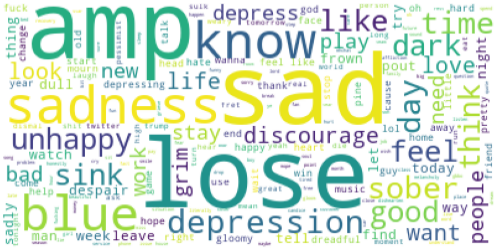
Angry Tweets



Feared Tweets

Happy Tweets

Sad Tweets



Avant de passer à la création des modèles, on réalise un undersampling afin d'équilibrer les proportions de chaque étiquette.

Modèles baseline

On pre-process notre texte avec TF-IDF vectorizer puis on applique plusieurs classifieurs dont les résultats sont résumés dans le tableau suivant.

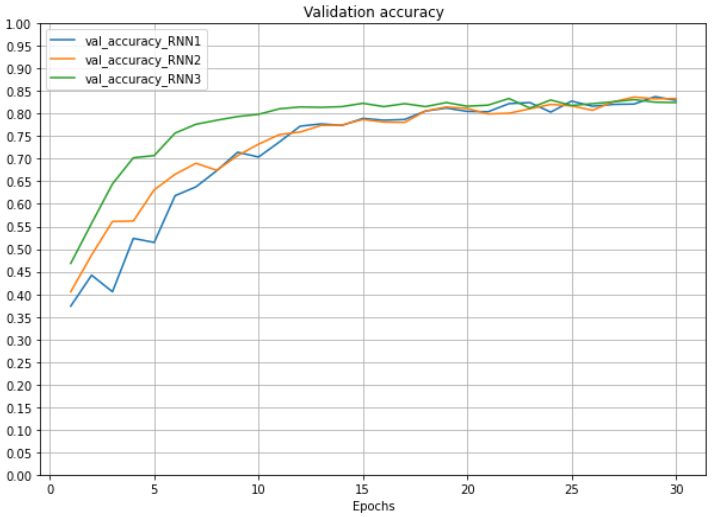
	accuracy	f1-score
KNN	0.5677	0.5688
LinearSVC	0.7153	0.7155
LogisticRegression	0.7096	0.71

Le f1-score correspond ici au macro f1-score c'est à dire à la moyenne des f1 scores de chaque target.

Avec les RNN

On utilise le word embedding GloVe pré-entraîné qu'on récupère à l'adresse https://nlp.stanford.edu/projects/glove/
Ensuite on a testé trois RNN :
RNN 1 : LSTM avec un "nettoyage de base" des données (cf III.b)
RNN 2 : LSTM avec en plus du nettoyage de base suppression de la ponctuation et des emojis
RNN 3 : LSTM bidirectionnel avec un "nettoyage de base" des données (cf III.b)

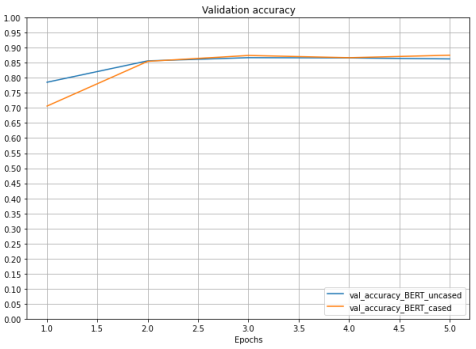
La fonction d'activation de sortie est la fonction softmax et on 4 unités de sortie (une pour chaque émotion).
L'accuracy sur le jeu de test est présente dans le graphique suivant.



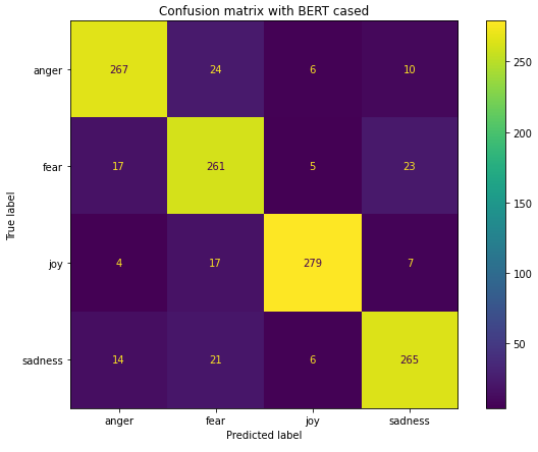
Les accuracy au bout de 30 epochs sont environ de 0.83. On observe que le modèle bidirectionnel donne cette performance beaucoup plus rapidement que les deux autres (15 epochs).

BERT

On applique le nettoyage de base et on essaye les modèles BERT cased et BERT uncased.
On exporte le modèle pré-entraîné à l'aide de la librairie Transformers.
La fonction d'activation de sortie est la fonction softmax et on 4 unités de sortie (une pour chaque émotion).
Les résultats sont présentés dans le graphique ci-contre.
Accuracy avec BERT cased : 0.875 et avec BERT uncased : 0.863.



Observons la matrice de confusion avec BERT cased.



L’émotion la moins bien reconnue est la peur, et la mieux reconnue est la joie.

Enfin, effectuons quelques textes. On constate que les prédictions ne sont pas très bonnes si on ne met pas certains mots clés. Il faudrait davantage de données pour obtenir un modèle plus robuste.

tweet	Prédiction BERT cased
'I feel happy for the gift! Thank you'	joy
'This play was wonderful!amazing! I wanna watch it again !’	joy
I passed my driving test !! 😀😀😀 #positive	fear
I passed my driving test !! 😀😀😀 #smile	joy
'The tornado caused a lot of material and human damage. My thoughts are with the victims'	fear
'The tornado caused a lot of material and human damage. I\'ve got so much pain for the victims.'	fear
'The tornado caused a lot of material and human damage. I feel so sad for the victims.'	sadness

Sources :

Emotion and sentiment analysis of tweets using BERT
http://ceur-ws.org/Vol-2841/DARLI-AP_17.pdf (mars 2021)

Guide to using pretrained word embeddings
https://blog.paperspace.com/pre-trained-word-embeddings-natural-language-processing/#final-thoughts

Cheatsheet NLP
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#word-representation

Multiclass classification using transformers for beginners
https://www.analyticsvidhya.com/blog/2021/12/multiclass-classification-using-transformers/

Cours : Openclassroom “introduction to NLP”; Andrew NG “sequence models”

MERCI