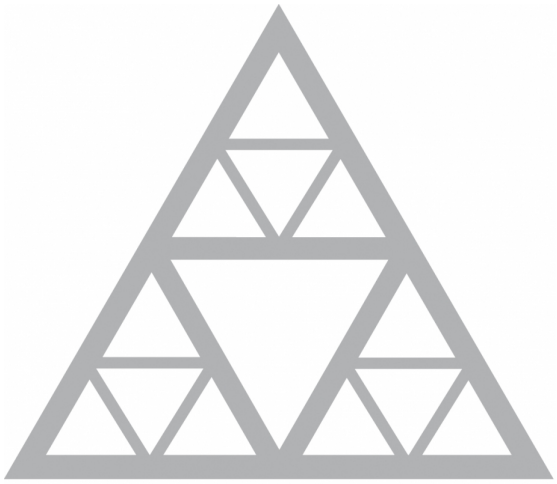
Machine Learning

Projet : extraction des sentiments des tweets



École des Ponts

ParisTech

Adel Chakir
Jean Landais
Sébastin Morel
Aurélien Pion

Mai 2020

Table des matières

1 Introduction

Le language est complexe. À l'aide de diverses parties de notre corps, nous pouvons communiquer les uns avec les autres. Une partie du language est composé de transmition de fait que l'on interprète objectivement comme tels. Mais, la transmition des émotions par le language est bien pus délicat car plus subjective. Lors d'une conversation de vives voix, les gestes, intonations et le vocabulaire utilisés nous aident à reperer l'emotion que notre interlocuteur souhaite nous transmettre. Mais, cela est bien plus délicat à l'écrit. En effet, il ne faut nous fier qu'au mot en eux même pour deceler des émotions. Or, sans contexte et les éléments corporels qui vont avec, il est plus difficile de saisir l'émotion, le sarcasme ou l'ironie etc... En revanche, on a appris à sélectionner des morceaux de phrase porteurs d'émotions. Cela nous permet donc d'avoir notre disposition un champ lexical de mot permettant d'exprimer verbalement des sentiments.
C'est cette capacité que nous allons tenter d'apprendre à nos algorithmes, pour pouvoir extraire des bouts de texte porteur de sentiment. Pour cela, nous utiliserons des tweets exprimant ou non des sentiments.

2 Pré-traitement des données

2.1 Bag of Words

Les algorithmes de machine learning sont construits de telle manière que les input sont des vecteurs de nombres, préferablement de taille fixe. Or, ici, nous disposons de tweets, qui sont des chaînes de caracteres, de taille variable. Ainsi, on ne peut pas utiliser les données brutes, il faut les traiter pour pouvoir s'en sevir.
Pour cela, il a fallu réfléchir à la façon de traduire chaque tweet en un vecteur de taille fixe.
Une méthode qui existe est qui est très utilisée est la représentation des vecteurs en "bag of words ". Cette méthode est relativement simple, on constitue un dictionnaire de tous les mots utilisés. Puis chaque chaîne de caractères est représentée par un vecteur de taille le nombre de mots, chaque indice représente un mot et il comptabilise le nombre de fois qu'il apparaît dans la chaîne de caractère.

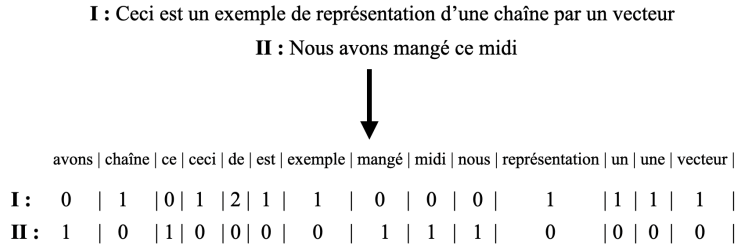


FIGURE 1 – Reprséentation des chaînes en bag of words

Or, ce processus est très lourd. En effet, on remarque que la taille de notre diccionnaire est démentielle pour le jeu de données d'entrainement. Il y a 28983 mots distincts présents dans notre corpus de tweets.
Nous avons intuité alors que c'est pour cela que nous devons d'abord selectionner du texte, pour pouvoir ensuite appliquer des algorithmes de classification, sur les morceaux de tweets séléctionnés. Cela ressemble à la manière que fonctionne la détéction d'émotion dans une phrase : nous utilisons un champs lexical propre à chaque émotion. Ainsi, nous avons vu que justement, les corpus de mots selectionnés étaient plus pauvres : en effet, celui associé aux sentiments positifs ne possède "que" 3962 mots, celui négatif en possède 4646 et l'union des deux en possède 7157 en tout. Cela diminue donc l'ensemble

des mots considérés. En revanche, pour les tweets neutres, on trouve un ensemble de mot conséquent : 15650 mots au total pour 18792 dans les 3 corpus confondus (de la colonne selected_text). Ainsi, l'ensemble des tweets neutres conservent une forte richesse de mot. On peut donc intuiter qu'on n'épure peu les tweets neutres. Cela est confirmé lorsqu'on l'on regarde les tweets classés neutres : on ne selectionne alors rien.
On va confirmer notre intuition en comparant l'histogramme des tailles des tweets séléctionnés par rapport à leur taille d'origine en fonction de leur classe :

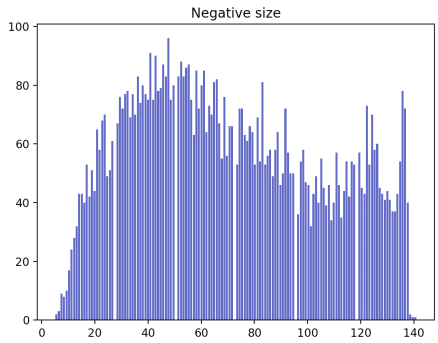


FIGURE 2 – Tailles des tweets négatifs avant

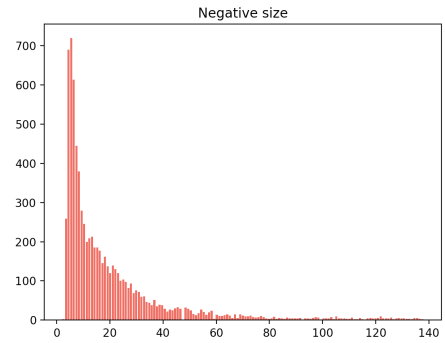


FIGURE 3 – Tailles des tweets négatifs après

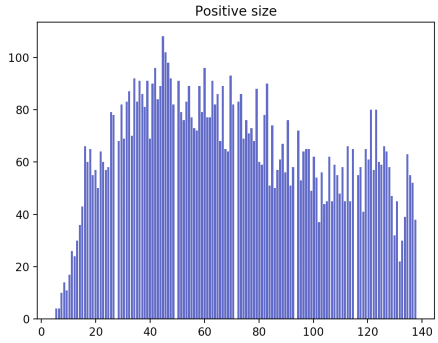


FIGURE 4 – Tailles des tweets positifs avant

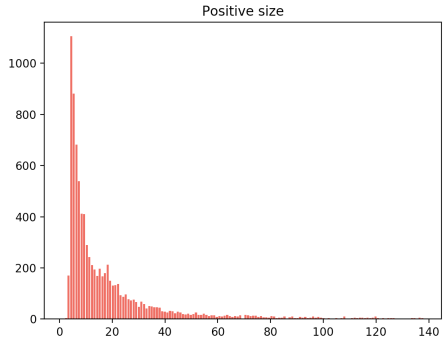


FIGURE 5 – Tailles des tweets positifs après

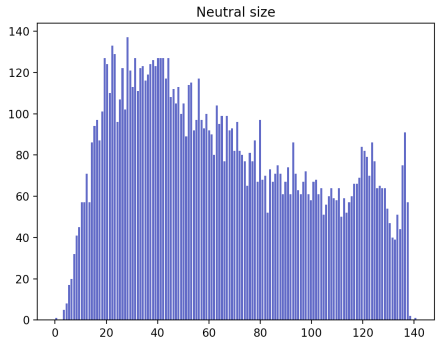


FIGURE 6 – Tailles des tweets neutres avant

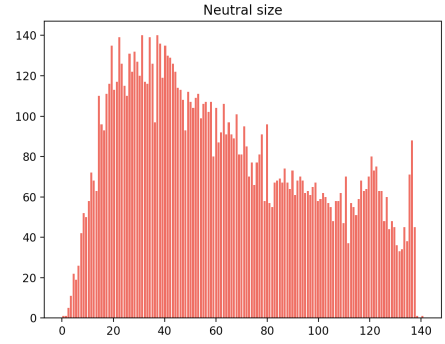


FIGURE 7 – Tailles des tweets neutres après

On remarque alors que les tweets neutres ne sont pas impactés par l'épuration contrairement aux deux autres classes. Ainsi, dans tous nos algorithmes, nous considerons que les tweets classés neutres ne sont pas épurés donc on conserve tout le tweet lors de la selection.

2.2 Affiner notre ensemble de mots

2.2.1 Enlever les mots considerés comme courant

On peut afficher les mots les plus influents du dictionnaire, en fonction de leur label :

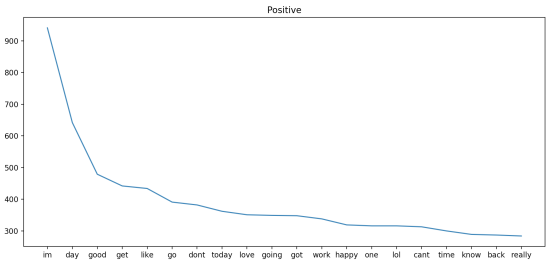


FIGURE 8 – Mots les plus influents des tweets positifs

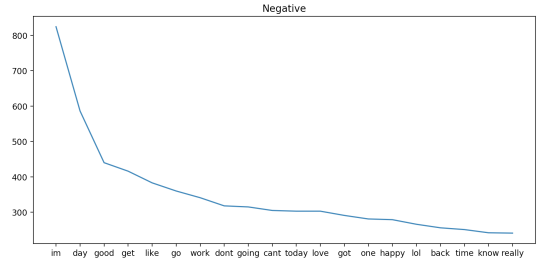


FIGURE 9 – Mots les plus influents des tweets negatifs

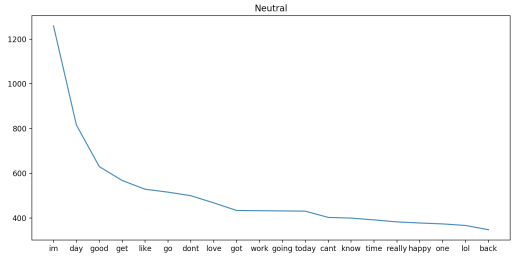
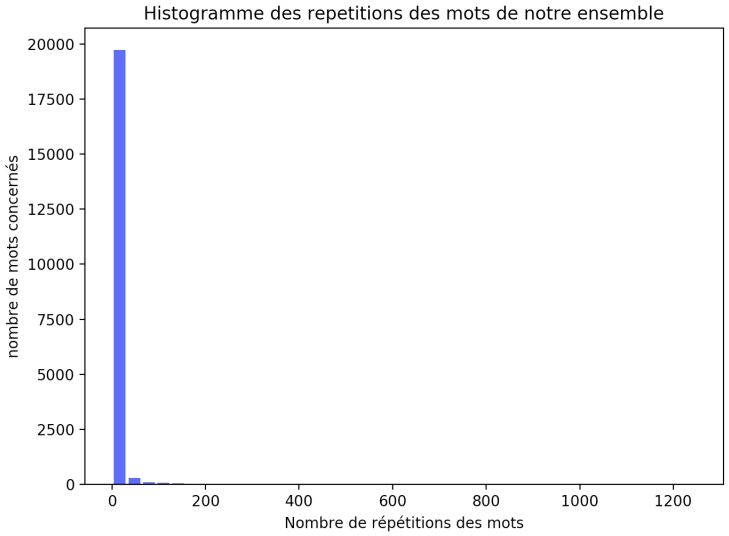


FIGURE 10 – Mots les plus influents des tweets neutres

Avec cela, on peut trouver les mots importants qu'on pourrait peut être ommetre de traiter : comme le mot I'm par exemple qui est omnis present.
Pour pouvoir réduire la dimension de notre ensemble de mot, il est important de pouvoir réduire le nombre de mot de cet ensemble. On va donc supprimer tout les mots que l'on utilise beaucoup sans qu'ils aient de sens direct avec les émotions. Nous avons décidé de supprimer les mots suivants : im, day, get, go, dont, today, going, got, work, one, cant, time, know, back, really, see, mothers, want, home, night, still, new, think, much, well, thanks, last, morning, need, tomorrow. On aura verifier au préalable que l'emploi de ces mots ne dependent pas des emotions perçues. Par exemple, on conserve le mot love, qui n'est pas autant présent dans le jeu de données négatif que dans celui positif.

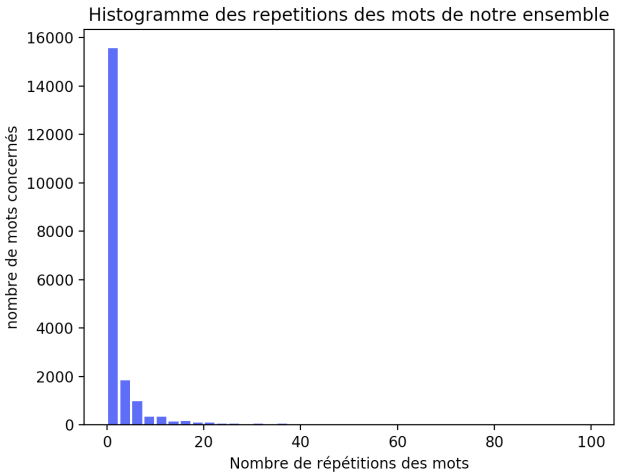
2.2.2 Enlever les mots trop peu utilisés

Bien que nous ne concentrons que sur les tweets possédant des émotions et que nous avons supprimés les mots considérés comme commun, notre ensemble de mot contient quand même 20296 mots distincts ce qui est énorme.
En effet, si nous voulons faire une PCA pour essayer de réduire la dimension en maximisant la variance entre les données d'entrainement, nous aurons une matrice à environ 400 millions de coefficients ce qui est énorme. Nous allons donc devoir trouver un moyen de se débarrasser de certains mots.
On pense alors que bien que les langues soient riches en vocabulaire, on utilise que très peu de mots en réalités en moyenne. De plus, les fautes de frappes, d'orthographe et les abréviations déclienent en beaucoup d'exemplaires des mots (par exemple, le mot "aw" pour imiter l'expression orale d'émerveillement, est décliné sous la forme aww, awww, awwwww etc...). On a donc décidé de regarder l'histogramme des mots présents dans tout notre diccionnaire en fonction de leur nombre de parution dans les tweets.



On voit alors qu'à cette échelle, on voit uniquement que la plus grande partie des mots ne se répetent que très peu ! Ainsi, il fait sens de se permettre d'ommettre les mots anodins de notre corpus de texte.

Même en zoomant, on voit que les mots ne se repetent que très rarement plus que 20 fois. On va donc choisir de ne pas prendre les mots ayant moins de 5 occurences pour pemettre de diminuer drastiquement notre ensemble de mot pris en compte.
Ceci n'est pas optimal, puisqu'on perd des mots inutiles mais aussi des mots importants tel que "insomnia" qui est rare mais négatif pour autant.
En revanche , cela permet de pouvoir effectuer une PCA sur nos données puisqu'on obtient alors un ensemble à 20296 mots à un ensemble à 2487 mots : on diminue de 10 le nombres de mots de notre ensemble.



2.3 Gérer les matrices sparses

Une fois qu'on a vectorialisé nos tweets, on se retrouve avec de gros vecteurs pour chaque tweet. Or, notre tweet est sparse : cela signifie que très peu de ses composantes sont nulles. On peut alors le représenter par une liste de couple notant la position des coordonnées non nulles ainsi que leur valeur. Cela permet de diminuer drastiquement les coefficients de notre matrice de texte.
En pratique, pour le jeu de données d'entrainement (en ne prenant pas les tweets neutres), si on vectorialise l'ensemble "text", on obtient une matrice de taille 20295 (le nombre de mot total) par 16363 (le nombre de tweet) soit 332087085 coefficients en tout. Or, en utilisant une matrice sparse, on trouve que seulement 101905 coefficients de notre matrice sont non nuls, soit 0.3% des coefficients!
Nous n'avons pas pu utiliser ces matrices pour nos algorithmes car elles n'apportaient que peu d'avantage mais elles peuvent être très importantes pour permettre justement de diminuer drastiquement le temps de calcul pour eviter des boucles "for" très couteuses en temps.

3 Algorithme naïf de selection de mot

Cette première méthode est une méthode tentant de copier le fonctionnement de l'apprentissage humain : On va considerer que c'est par l'apprentissage qu'on remarque l'aspect sentimentale de certains mots. Ainsi, on va se servir du jeu d'entrainement pour construire l'ensemble de mot que nous reconnaitrons comme étant l'ensemble des mots d'émotions.

3.1 Construction du dictionnaire

On va alors construire un ensemble de mots contenant l'ensemble des mots présents dans les séléctions des tweets. On ne prendra que les tweets labélisés positif ou négatif car on a remarqué que les tweets neutres ne sont pas modifiés. Ainsi, les morceaux des tweets neutres séléctionnés ne sont pas pertinents.

3.2 Traitement du jeu de test

On va alors vectorialiser notre ensemble de tweets de test à l'aide de notre diccionnaire calculé précedement. Cela va donc nous donner la repartition des mots d'émotions dans notre jeu de données de test. On va alors considerer que nous selec-

tionnerons les même mots.

3.3 Séléction

On va alors séléctionner uniquement les parties de tweets que nous avons déjà vu dans le jeu d'entrainement. On va remettre les morceaux dans l'ordre de la phrase et on obtient alors ce que l'on veut.
Il faut aussi vérifier que les tweets déjà labélisés de neutre ne selectionnent pas de boût de tweet, donc on les copie tel quel.
Enfin, il se peut que dans les tweets labélisés positif ou negatif, on ne trouve pas de mot associé à une émotion. On va alors recopier le tweet de base pour ne pas avoir une séléction vide pour ce genre de tweet.

3.4 Résultat

Exemples de résultats		
Émotion	Tweet de base	Séléction
Negative	So hot today =_= don‘t like it and i hate my new timetable, having such a bad week	hot like hate bad week
Negative	I‘m so very tired...and have insomnia.	insomnia
Positive	you‘re the absoute best	youre best
Negative	Now I have a sunburn	sunburn
Neutre	_celeste finally some sleep in silence	_celeste finally some sleep in silence

Vous trouverez ci-joint à ce rapport, le résultat global de cette méthode dans un fichier csv.

3.5 Problèmes liés à cette méthode

Tout d'abord, on a supprimé la ponctuation, donc on ne la retrouve pas en sortie. Cela pose donc un problème puisqu'il faudrait retrouver la ponctuation aussi, tel que les "..." qui permette aussi d'exprimer des expressions.
Ensuite, on ne capte pas les expressions puisqu'on raisonne mot par mot.
De plus, on supprime un grand nombre de mot dans notre épuration, beaucoup sont anodins mais quelques exceptions font qu'on perd de la diversité dans notre séléction de texte.
Enfin, on ne selectionne que des mots selectionnés dans la méthode d'entrainement. L'algorithme ne permet pas d'apprendre par lui même en reconnaissant des patterns plus subtils que les mots en eux-même.

4 Sélection de mots après traitement par un algorithme d'inspiration k-NN

Dans un second temps, nous avons de nous inspirer de l'algorithme de k-NN pour produire un programme nous permettant d'identifier la partie d'un tweet la plus susceptible de véhiculer une émotion. On dispose d'un ensemble de tweets d'entrainement classés en 3 catégories : neutre, positifs et négatifs.

4.1 Partition en classes et métrique employée

Nous sommes donc partis du fait que nous disposions de 3 classes bien définies, à savoir "neutre", "positif" et "négatif", comprenant quelques milliers de tweets chacune pour essayer de classer toute chaîne de caractère dans l'une de ces 3 classes en définissant une métrique sur l'espace des chaînes de caractères. La compétition Kaggle originale nous propose déjà une métrique, la similitude de Jaccard, définie comme suit :

```
def jaccard(str1, str2):
    a = set(str1.lower().split())
    b = set(str2.lower().split())
    c = a.intersection(b)
    return float(len(c)) / (len(a) + len(b) - len(c))
```

Il faut tout d'abord noter que cette quantité n'est pas une distance mais bien, comme son nom l'indique, une métrique, c'est à dire une quantité positive qui se rapproche de 1 si les deux éléments comparés sont jugés proches.
La similarité de Jaccard fonctionne comme suit : pour deux chaînes de caractères prises en argument, on regarde le nombre de mots qu'elles ont en commun et on renvoie la fraction de ce nombre sur le nombre total de mots distincts présent dans les deux chaînes.
Cette métrique est celle que nous avons utilisé, mais il nous a néanmoins fallu l'adapter : Dans le cas où on lui rentre deux chaînes de caractères ne comprenant qu'un mot chacune, elle renvoie un si ces deux mots sont le même et zéro sinon. Des mots de la même famille ayant la même racine mais n'étant pas exactement pareils seront donc considérés comme étant très éloigné car le Jaccard renverra zéro. Pour pallier à ça, on a défini un Jaccard sur les lettres que l'on applique dans le cas de chaînes ne comprenant qu'un mot :

```
def jaccard_lettres(str1, str2):
    l1 = []
    l2 = []
    for i in range(len(str1)):
        l1.append(str1[i])
    for i in range(len(str2)):
        l2.append(str2[i])
    a = set(l1)
    b = set(l2)
    c = a.intersection(b)
    return float(len(c)) / (len(a) + len(b) - len(c))
```

Cette nouvelle métrique fonctionne exactement comme le Jaccard donné par Kaggle, mais sur les lettres. C'est à dire que la quantité renvoyée est d'autant plus proche de 1 que le nombre de lettres en commun entre les deux mots est grand. Les lettres jouent pour le mot le rôle que les mots jouaient pour la phrase dans le Jaccard initial.

Une fois ces métriques définies, nous les avons utilisée pour pouvoir attribuer à une chaîne de caractère la catégorie qui lui correspond, selon la métrique, parmi "neutre", "positif" et "negatif".

4.2 Algorithme récursif de sélection

Le premier algorithme, récursif, que nous avons cherché à appliquer est le suivant :

Entrée : Un tweet, c'est à dire une chaine de caractères dont le nombre de caractères est limité à 260, et une classe (0 pour "neutre", 1 pour "positif", 2 pour "négatif")

- Les 3 classes de notre partition sont déjà définies et invariantes. En utilisant la métrique de Jaccard fournie sur Kaggle, on calcule la similitude du tweet à chacune des trois classes. C'est bien une similitude et pas une distance, c'est à dire une quantité positive telle que plus les deux chaînes de caractères comparées sont proches, plus leur similitude est proche de 1. On affecte ensuite le tweet à la catégorie avec laquelle sa similitude est la plus élevée. C'est la partie de l'algorithme inspirée du k-NN.
- Si le tweet a été affecté à la catégorie "neutre", on s'arrête là. Sinon, on le coupe en deux et on applique le point précédent à chacune des deux moitiés, en gardant en mémoire la catégorie du tweet complet. Si aucune des deux moitiés n'est affectée à cette catégorie, on renvoie le tweet entier. Si les deux moitiés sont affectées à la catégorie d'origine, on renvoie le tweet entier. Si seule l'une des deux moitiés est affectée à la catégorie d'origine, on la découpe en deux et on procède récursivement
- Si la chaîne de caractères prise en entrée ne contient qu'un mot, on la renvoie telle quelle. Cela assura la terminaison de l'algorithme.

Ce premier algorithme nous a permis d'obtenir les résultats suivant :

Exemples de résultats		
Émotion	Tweet de base	Séléction
Negative	So hot today =_= don‘t like it and i hate my new timetable, having such a bad week	newtimetable,
Negative	I‘m so very tired...and have insomnia.	tired...
Positive	this is good	good
Negative	This is bad	bad
Neutre	_celeste finally some sleep in silence	” (chaine vide)

Cet algorithme s'est empiriquement avéré efficace pour détecter les phrases ayant un mot fortement connoté positif ou négatif, par exemple "good", "great" ou "bad". Néanmoins, ses performances sur des tweets moins fortement connotés sont moindres. De plus, cet algorithme de sélection présente deux inconvénients : Tout d'abord, il ne renvoie que des chaines de caractères ne contenant qu'un seul mot. Ensuite, et c'est le principal problème, il est très long (environ 10 secondes pour traiter un tweet de 15 mots), ce qui nous a empêcher de le faire tourner sur la totalité de nos tweets.

4.3 Algorithme "simple" de sélection

Pour pallier à la lenteur du précédent algorithme et au fait qu'il ne renvoie q'un seul mot, nous avons développé l'algorithme suivant :

Entrée : Un tweet, c'est à dire une chaine de caractères dont le nombre de caractères est limité à 260, un entier "classe_originale" qui correspond à la classe originale du tweet (égal à 0 pour "neutre", 1 pour "positif", 2 pour "négatif"), un entier "nb_parties" égal au nombre de parties qu'on souhaite découper le tweet.

— On découpe le tweet en un nombre de parties égal à nb_parties

— On applique notre algorithme de classification à chacune des parties du tweet.

— On initialise une chaine de caractères vide "resultat"

— Pour chaque partie du tweet, si sa classe est égale à la classe du tweet originale, on la concatène avec la chaîne de caractères "résultat".

— On renvoie "résultat".

Ce second algorithme est plus rapide que le précédent, mais demeure lent (environ une seconde pour traiter un tweet de 15 mots). Il présente l'avantage d'avoir un output dont la longueur n'est pas bloquée à un mot. Voici quelques résultats obtenus via cet algorithme, pour un nombre de parties de découpe du tweet égal à trois :

Exemples de résultats		
Émotion	Tweet de base	Séléction
Negative	So hot today =_= don‘t like it and i hate my new timetable, having such a bad week	So hot today =_= don‘tlike it and i hate my,
Negative	I‘m so very tired...and have insomnia.	I‘m so
Positive	this is good	good
Negative	This is bad	This is bad
Neutre	_celeste finally some sleep in silence	some sleep in silence

5 Prédiction des selection de textes avec des méthodes de classifications

On va considérer que les mots n'aparraissent qu'une fois dans un tweet. Si un mot apparaît plus d'une fois dans un tweet, dans sa représentation de bag of words, on mettra seulement 1 à l'indice en question.
On ne peut pas utiliser les algorithmes de classifications tel quel dans notre problème ni les algorithmes de regression. En effet ils prédisent uniquement des scalaires et la sortie que l'on a est un vecteur. Ainsi nous avons décidé d'utiliser ces méthodes sur chaque composante des vecteurs de sorties, soit de réaliser environ 2000 classifications.
Plus concrètement soit $x_i = [0,0,1,...,0,1,0,0]$ les vecteurs représentant les text et soit $y_i = [0,0,0,0,...,1,0,1]$ les vecteurs selected text. Soit P_k l'application qui transforme x_i en la coordonnée $(y_i)_k$, $P_k(x_i) = (y_i)_k$. On cherche k P_k applications qui vont ainsi à partir du vecteur x_i de retrouver le vecteur y_i.
En fait ici P_k représente un algorithme de classification. Pour notre projet on a choisi d'utiliser une regression linéaire et une méthode de type SVM pour classifier chaque vecteur sur chaque composante.
Pour appliquer les algorithmes nous avons réduit la taille des vecteurs x_i avec une pca (on a projeté sur un espace de dimension 25). Puis nous avons réalisé les 2000 algorithmes de classification ce qui est la partie la plus longue du processus.

Exemples de résultats		
Émotion	Tweet de base	Séléction
Negative	http ://twitpic.com/4wp8s - My ear hurts, and THIS is my medicine. GUM	ear hurts
Positive	Happy Mothers Day!!!	Happy
Positive	i always have those for my Champions League parties Tis awesome	always Tis awesome
Negative	bahah sadly I am not	sadly
Neutre	_celeste finally some sleep in silence	" (chaine vide)

6 Problèmes rencontrés et amélioration

On a rencontré de nombreux problèmes mais le principal étant la dimension très importante des données que l'on manipule. Ainsi une première piste d'amélioration est de réduire encore la taille du dictionnaire de mot de manière intelligente. En effet on peut supprimer du corpus tous les nombres, les numéros de téléphone, les adressses mails, les adresses de sites internet etc... Mais aussi on peut supprimer de nombreux mots en mettant tous les verbes à l'infinitif par exemples, ou alors les mots du type "allll", "alll", "alllllllll" se représente par simplement "all" et on peut supprimer les mots mal orthographiés. Mais il ne faut non plus trop supprimer des mots de notre dictionnaire car sinon on perd du sens. Donc on doit garder un dictionnaire de base assez important malgrés tout. Ainsi avec de grands dictionnaire on a des meilleurs représentation mais les algorithmes vont mettre très longtemps à tourner pour avoir une solution acceptable.
En problème important que nous n'avons pas considéré ici est l'ordre des mots qui peut être important pour la signification de la phrase (par exemple "not loved" n'a pas du tout le même sens que "love). Et aussi on peut prendre en compte le fais que certain apparaissent plusieurs fois dans une même phrase lors de l'utilisation des algorithmes de classification.
Enfin pour améliorer davantage notre approche, il faudrait également optimiser d'avantages nos calculs, pour pouvoir mettre en place peut être des réseaux de neurones pouvant effectuer de la séléction de texte.

7 Bibliographie

- Overview of Text Similarity Metrics in Python - Sanket Gupta - https ://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50

- Jaccard index - Wikipedia - https ://en.wikipedia.org/wiki/Jaccard_index.
- A Gentle Introduction to Sparse Matrices for Machine Learning - Jason Brownlee