Analyse des sentiments des Tweets

March 17, 2019

1 Introduction

Dans ce projet, nous construirons **Logistic Regression Classifier** de polarité à trois sens (positif, négatif, neutre) pour les tweets, sans utiliser le moteur d'analyse de sentiment intégré à NLTK.

Nous utiliserons **Logistic Regression Classifier**, des fonctions **bag-of-words** et des lexiques de polarité (à la fois intégrés et externes). Nous créerons également notre propre module de prétraitement pour traiter les tweets bruts.

2 Données utilisées

- training.json : Ce fichier contient ~15k tweets bruts, avec leurs étiquettes de polarité (1 = positif, 0 = neutre, -1 = négatif). Nous utiliserons ce fichier pour former nos classificateurs.
- develop.json : Dans le même format que training.json, le fichier contient un plus petit ensemble de tweets. Nous l'utiliserons pour tester les prédictions de nos classificateurs qui ont été entraînés sur le plateau d'entraînement.

3 Prétraitement

La première chose que nous allons faire est de prétraiter les tweets pour qu'ils soient plus faciles à gérer, et prêts pour l'extraction de fonctionnalités, et la formation par les classificateurs.

Pour commencer, nous allons extraire les tweets du fichier json, lire chaque ligne et stocker les tweets, les étiquettes dans des listes séparées.

Ensuite, pour le prétraitement, nous allons :

- segmenter les tweets en phrases à l'aide d'un segmenteur NTLK
- extraire les phrases à l'aide d'un tokenizer NLTK
- tous les mots en minuscules
- supprimer les noms d'utilisateur twitter commençant par @ en utilisant regex
- supprimer les URL commençant par http en utilisant regex
- nous allons extraire les hashtags, et essayer de les décomposer en multi-mots en utilisant un algorithme MaxMatch, et le dictionnaire de mots anglais fourni avec NLTK.

Construisons quelques fonctions pour accomplir tout cela.

```
In [1]: import json
        import re
        import nltk

        lemmatizer = nltk.stem.wordnet.WordNetLemmatizer()
        dictionary = set(nltk.corpus.words.words()) #To be used for MaxMatch

        #Fonction pour lemmatiser le mot | Utilisé pendant maxmatch
        def lemmatize(word):
            lemma = lemmatizer.lemmatize(word,'v')
            if lemma == word:
                lemma = lemmatizer.lemmatize(word,'n')
            return lemma

        #Fonction pour implémenter l'algorithme maxmatch pour les hashtags multi-mots
        def maxmatch(word,dictionary):
            if not word:
                return []
            for i in range(len(word),1,-1):
                first = word[0:i]
                rem = word[i:]
                if lemmatize(first).lower() in dictionary: #Il est important de lemmatiser les
                    return [first] + maxmatch(rem,dictionary)
            first = word[0:1]
            rem = word[1:]
            return [first] + maxmatch(rem,dictionary)

        #Fonction pour prétraiter un seul tweet dans le dictionnaire.
        def preprocess(tweet):

            tweet = re.sub("@\w+","",tweet).strip()
            tweet = re.sub("http\S+","",tweet).strip()
            hashtags = re.findall("#\w+",tweet)

            tweet = tweet.lower()
            tweet = re.sub("#\w+","",tweet).strip()

            hashtag_tokens = [] #Liste séparée pour les hashtags

            for hashtag in hashtags:
                hashtag_tokens.append(maxmatch(hashtag[1:],dictionary))

            segmenter = nltk.data.load('tokenizers/punkt/english.pickle')
            segmented_sentences = segmenter.tokenize(tweet)

            #La symbolisation générale
            processed_tweet = []
```

```
        word_tokenizer = nltk.tokenize.regexp.WordPunctTokenizer()
        for sentence in segmented_sentences:
            tokenized_sentence = word_tokenizer.tokenize(sentence.strip())
            processed_tweet.append(tokenized_sentence)

        #Traitement des hashtags seulement quand ils existent dans un tweet
        if hashtag_tokens:
            for tag_token in hashtag_tokens:
                processed_tweet.append(tag_token)

        return processed_tweet

    #Fonction personnalisée qui prend un fichier, et passe chaque tweet au préprocesseur
    def preprocess_file(filename):
        tweets = []
        labels = []
        f = open(filename)
        for line in f:
            tweet_dict = json.loads(line)
            tweets.append(preprocess(tweet_dict["text"]))
            labels.append(int(tweet_dict["label"]))
        return tweets,labels
```

Avant d'exécuter le prétraitement de nos données d'entraînement, voyons comment fonctionne l'algorithme maxmatch.

```
In [2]: maxmatch('wecan',dictionary)

Out[2]: ['we', 'can']
```

Essayons de lui donner quelque chose de plus dur que ça.

```
In [3]: maxmatch('casestudy',dictionary)

Out[3]: ['cases', 'tu', 'd', 'y']
```

Comme nous pouvons le voir dans l'exemple ci-dessus, il décompose incorrectement le mot'casestudy', en renvoyant'cases', au lieu de'case' pour la première itération, ce qui aurait été un meilleur résultat. C'est parce qu'il extrait avec gourmandise les'cases' en premier.

Pour une amélioration, nous pouvons implémenter un algorithme qui compte mieux le nombre total de correspondances réussies dans le résultat du processus maxmatch, et retourner celui qui a le plus grand nombre de correspondances réussies.

Exécutons notre module de prétraitement sur les données brutes de formation.

```
In [7]: #Exécuter le module de prétraitement de base et capturer les données (peut-être passer
        train_data = preprocess_file(r'C:\Users\acer\Desktop\training.json')
        train_tweets = train_data[0]
        train_labels = train_data[1]

In [8]: print (train_tweets[:2])
```

```
[[['dear', 'the', 'newooffice', 'for', 'mac', 'is', 'great', 'and', 'all', ',', 'but', 'no', '
```

Hmm... on peut faire mieux que ça pour comprendre ce qui se passe. Ecrivons un script simple qui lancera le module de prétraitement sur quelques tweets, et imprimera les résultats originaux et traités, côte à côte, s'il détecte un hashtag multi-mot.

```
In [38]: #printing des exemples de hashtags multi-mots (ne fonctionne pas pour les tweets mult
         f = open(r'C:\Users\acer\Desktop\training.json')
         count = 1
         for index,line in enumerate(f):
             if count >5:
                 break
             original_tweet = json.loads(line)["text"]
             hashtags = re.findall("#\w+",original_tweet)
             if hashtags:
                 for hashtag in hashtags:
                     if len(maxmatch(hashtag[1:],dictionary)) > 1:
                         #Si la longueur du tableau renvoyé par la fonction maxmatch est supér
                         #ca signifie que l'algorithme a détecté un hashtag avec plus d'un mot
                         print (str(count) + ". Tweet original : " + original_tweet + "\ntweet
                         count += 1
                         break
```

```
1. Tweet original : If I make a game as a #windows10 Universal App. Will #xboxone owners be abl
tweet traité: [['if', 'i', 'make', 'a', 'game', 'as', 'a', 'universal', 'app', '.'], ['will',

2. Tweet original : Microsoft, I may not prefer your gaming branch of business. But, you do mal
tweet traité: [['microsoft', ',', 'i', 'may', 'not', 'prefer', 'your', 'gaming', 'branch', 'of

3. Tweet original : @MikeWolf1980 @Microsoft I will be downgrading and let #Windows10 be out fo
tweet traité: [['i', 'will', 'be', 'downgrading', 'and', 'let', 'be', 'out', 'for', 'almost',

4. Tweet original : @Microsoft 2nd computer with same error!!! #Windows10fail Guess we will sho
tweet traité: [['2nd', 'computer', 'with', 'same', 'error', '!!!'], ['guess', 'we', 'will', 'sh

5. Tweet original : Sunday morning, quiet day so time to welcome in #Windows10 @Microsoft @Winc
tweet traité: [['sunday', 'morning', ',', 'quiet', 'day', 'so', 'time', 'to', 'welcome', 'in']
```

C'est mieux comme ça ! Notre module de prétraitement fonctionne comme prévu.

L'étape suivante consiste à convertir chaque tweet traité en un dictionnaire de fonctionnalités pour un sac de mots. Nous allons permettre des options pour supprimer les mots vides pendant le processus, et aussi pour supprimer les mots rares, c'est-à-dire les mots qui apparaissent moins de n fois dans l'ensemble de la formation.

```
In [11]: from nltk.corpus import stopwords
```

```
stopwords = set(stopwords.words('english'))

#Pour identifier les mots apparaissant moins de n fois, nous sommes en train de créer

total_train_bow = {}

for tweet in train_tweets:
    for segment in tweet:
        for token in segment:
            total_train_bow[token] = total_train_bow.get(token,0) + 1

#Fonction pour convertir les tweets pré_traités en dictionnaires de fonctionnalité de
#Permet de supprimer des options pour supprimer des mots vides, et aussi pour supprim
def convert_to_feature_dicts(tweets,remove_stop_words,n):
    feature_dicts = []
    for tweet in tweets:
        # Dictionnaire des fonctionnalités de build pour tweet
        feature_dict = {}
        if remove_stop_words:
            for segment in tweet:
                for token in segment:
                    if token not in stopwords and (n<=0 or total_train_bow[token]>=n)
                        feature_dict[token] = feature_dict.get(token,0) + 1
        else:
            for segment in tweet:
                for token in segment:
                    if n<=0 or total_train_bow[token]>=n:
                        feature_dict[token] = feature_dict.get(token,0) + 1
        feature_dicts.append(feature_dict)
    return feature_dicts
```

Maintenant que nous avons notre fonction de conversion de tweets bruts en dictionnaires de fonctionnalités, nous allons l'exécuter sur training and development data. Nous convertirons également le dictionnaire des fonctionnalités en une représentation éparses, afin qu'il puisse être utilisé par les algorithmes ML de scikit.

In [13]:
```
from sklearn.feature_extraction import DictVectorizer
vectorizer = DictVectorizer()

#Conversion en dictionnaires de fonctionnalités
train_set = convert_to_feature_dicts(train_tweets,True,2)

dev_data = preprocess_file(r'C:\Users\acer\Desktop\training.json')

dev_set = convert_to_feature_dicts(dev_data[0],False,0)

#Conversion en représentations éparses
training_data = vectorizer.fit_transform(train_set)
```

```
development_data = vectorizer.transform(dev_set)
```

4 Classification

Maintenant, nous allons passer nos données dans decision tree classifier, et essayer d'ajuster les paramètres en utilisant la recherche par grille sur les combinaisons de paramètres.

```
In [47]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.model_selection import cross_validate
         from sklearn.metrics import accuracy_score, classification_report
         from sklearn.model_selection import GridSearchCV

         #Grille utilisée pour tester les combinaisons de paramètres
         tree_param_grid = [
             {'criterion':['gini','entropy'], 'min_samples_leaf': [75,100,125,150,175], 'max_fe
             }
         ]

         tree_clf = GridSearchCV(DecisionTreeClassifier(),tree_param_grid,cv=10,scoring='accura

         tree_clf.fit(training_data,train_data[1])

         print ("Paramètres optimaux pour DT: " + str(tree_clf.best_params_)) #Pour print la m

         tree_predictions = tree_clf.predict(development_data)

         print ("\nPrécision de la Decision Tree : " + str(accuracy_score(dev_data[1],tree_prec
```

```
Paramètres optimaux pour DT: {'criterion': 'entropy', 'max_features': None, 'min_samples_leaf'

Précision de la Decision Tree : 0.5729842308836656
```

Le **decision tree classifier** ne semble pas fonctionner très bien, mais nous n'avons toujours pas de point de repère avec lequel le comparer.

Exécutons nos données dans **dummy classifier** qui choisira la classe la plus fréquente comme sortie, à chaque fois.

```
In [40]: from sklearn.dummy import DummyClassifier

         #Le classificateur fictif ci-dessous prédit toujours la classe la plus fréquente, com
         dummy_clf = DummyClassifier(strategy='most_frequent')
         dummy_clf.fit(development_data,dev_data[1])
         dummy_predictions = dummy_clf.predict(development_data)

         print ("\nPrécision de référence de la classe la plus courante: " + str(accuracy_score
```

```
Précision de référence de la classe la plus courante: 0.42862243379946446
```

Nous pouvons voir que notre classificateur DT est au moins plus performant que le classificateur factice.

Nous allons faire la même chose pour le classificateur **logisitc regression maintenant**.

```
In [48]: from sklearn.linear_model import LogisticRegression

         log_param_grid = [
             {'C':[0.012,0.0125,0.130,0.135,0.14],
              'solver':['lbfgs'],'multi_class':['multinomial']
             }
         ]

         log_clf = GridSearchCV(LogisticRegression(),log_param_grid,cv=10,scoring='accuracy')

         log_clf.fit(training_data,train_data[1])

         log_predictions = log_clf.predict(development_data)

         print ("Paramètres optimaux pour LR: " + str(log_clf.best_params_))

         print ("Précision de la Logistic Regression  : " + str(accuracy_score(dev_data[1],log_
```

```
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWar
  "of iterations.", ConvergenceWarning)
```

```
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)
C:\Users\acer\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarn
  "of iterations.", ConvergenceWarning)


Paramètres optimaux pour LR: {'C': 0.012, 'multi_class': 'multinomial', 'solver': 'lbfgs'}
Précision de la Logistic Regression  : 0.663373995834573
```

Pour récapituler ce qui vient de se passer, nous avons créé un classificateur **logistic regression** en effectuant **grid search** des meilleurs paramètres pour C (paramètre de régularisation), le type de solveur et la gestion multi_class, tout comme nous l'avons fait pour **decision tree classifier**.

5 Lexiques de polarité

Maintenant, nous allons essayer d'intégrer des informations externes dans l'ensemble de formation, sous la forme des scores de polarité pour les tweets.

Nous allons construire deux lexiques automatiques, les comparer avec l'ensemble annoté manuellement de NLTK, puis ajouter cette information à nos données de formation.

ce lexique sera construit via SentiWordNet. Ceci a pré-calculé des scores positifs, négatifs et neutres pour certains mots dans WordNet. Comme cette information est arrangée sous forme de synapses, nous prendrons simplement la polarité la plus commune à travers ses sens (et prendrons neutre en cas d'égalité).

```
In [37]: from nltk.corpus import sentiwordnet as swn
         from nltk.corpus import wordnet as wn
         import random

         swn_positive = []

         swn_negative = []

         #Fonction fournie avec l'affectation, non décrite ci-dessous.
         def get_polarity_type(synset_name):
             swn_synset =  swn.senti_synset(synset_name)
             if not swn_synset:
                 return None
             elif swn_synset.pos_score() > swn_synset.neg_score() and swn_synset.pos_score() >
                 return 1
             elif swn_synset.neg_score() > swn_synset.pos_score() and swn_synset.neg_score() >
                 return -1
             else:
                 return 0


         for synset in wn.all_synsets():

             # Nombre de synset polarité pour chaque lemme
             pos_count = 0
             neg_count = 0
             neutral_count = 0

             for lemma in synset.lemma_names():
                 for syns in wn.synsets(lemma):
                     if get_polarity_type(syns.name())==1:
                         pos_count+=1
                     elif get_polarity_type(syns.name())==-1:
                         neg_count+=1
                     else:
                         neutral_count+=1
```

```
            if pos_count > neg_count and pos_count >= neutral_count: #>=neutre comme des mots
                                                                     #même si elle est tou
                swn_positive.append(synset.lemma_names()[0])
            elif neg_count > pos_count and neg_count >= neutral_count:
                swn_negative.append(synset.lemma_names()[0])

        swn_positive = list(set(swn_positive))
        swn_negative = list(set(swn_negative))


        print  (random.sample(swn_positive,5))

        print  (random.sample(swn_negative,5))
```

```
['doubtful', 'unclogged', 'niceness', 'Mama', 'ad_hominem']
['unargumentative', 'ductless', 'dyspeptic', 'nonadhesive', 'countermand']
```

Cela ressemble à un excellent ensemble de deux mots négatifs positifs, en regardant les échantillons. Mais voyons comment il se compare au jeu annoté manuellement de NLTK.

Lexiques pour la classification Et si on utilisait les lexiques pour le problème principal de classification ?

Créons une fonction qui calcule un score de polarité pour une phrase basée sur un lexique donné. Nous allons compter les mots positifs et négatifs qui apparaissent dans le tweet, puis retourner un +1 s'il y a plus de mots positifs, un -1 s'il y a plus de mots négatifs, et un 0 sinon.

Nous comparerons ensuite les résultats des deux lexiques sur l'ensemble de développement.

```
In [45]: from nltk.corpus import opinion_lexicon
         import math

         positive_words = opinion_lexicon.positive()
         negative_words = opinion_lexicon.negative()
         #Tous les lexiques sont convertis en ensembles pour un prétraitement plus rapide.
         manual_pos_set = set(positive_words)
         manual_neg_set = set(negative_words)

         syn_pos_set = set(swn_positive)
         syn_neg_set = set(swn_negative)


         #Fonction permettant de calculer le score de polarité d'une phrase en fonction de la
         def get_polarity_score(sentence,pos_lexicon,neg_lexicon):
             pos_count = 0
             neg_count = 0
             for word in sentence:
                 if word in pos_lexicon:
                     pos_count+=1
                 if word in neg_lexicon:
```

```
                neg_count+=1
        if pos_count>neg_count:
            return 1
        elif neg_count>pos_count:
            return -1
        else:
            return 0


    #Fonction permettant de calculer le score pour chaque tweet, de le comparer aux étiqu
        accuracy_count = 0
        for index,tweet in enumerate(dataset):
            if datalabels[index]==get_polarity_score([word for sentence in tweet for word
                accuracy_count+=1
        return (accuracy_count/len(dataset))*100

    print ("Précision du lexique manuel: "+str(data_polarity_accuracy(dev_data[0],dev_data
    print ("Précision du lexique automatique : "+str(data_polarity_accuracy(dev_data[0],de
```

```
Précision du lexique manuel: 55.001487652484386
Précision du lexique automatique : 50.50877714965784
```

6 Lexique de la polarité avec apprentissage machine

En conclusion, nous étudierons les effets de l'ajout du score de polarité comme caractéristique de notre classificateur statistique.

Nous allons créer une nouvelle version de notre fonction d'extraction de fonctionnalités, pour intégrer les fonctionnalités supplémentaires et recycler notre classificateur de régression logistique pour voir s'il y a une amélioration.

```
In [33]: def convert_to_feature_dicts_v2(tweets,manual,first,remove_stop_words,n):
            feature_dicts = []
            for tweet in tweets:
                # Dictionnaire des fonctionnalités de build pour tweet
                feature_dict = {}
                if remove_stop_words:
                    for segment in tweet:
                        for token in segment:
                            if token not in stopwords and (n<=0 or total_train_bow[token]>=n)
                                feature_dict[token] = feature_dict.get(token,0) + 1
                else:
                    for segment in tweet:
                        for token in segment:
                            if n<=0 or total_train_bow[token]>=n:
                                feature_dict[token] = feature_dict.get(token,0) + 1
                if manual == True:
```

```
                    feature_dict['manual_polarity'] = get_polarity_score([word for sentence in
                if first == True:
                    feature_dict['synset_polarity'] = get_polarity_score([word for sentence in

                feature_dicts.append(feature_dict)
            return feature_dicts

In [35]: training_set_v2 = convert_to_feature_dicts_v2(train_tweets,True,False,True,2)

         training_data_v2 = vectorizer.fit_transform(training_set_v2)

In [49]: dev_set_v2 = convert_to_feature_dicts_v2(dev_data[0],True,False,False,0)

         development_data_v2 = vectorizer.transform(dev_set_v2)

         log_clf_v2 = LogisticRegression(C=0.012,solver='lbfgs',multi_class='multinomial')

         log_clf_v2.fit(training_data_v2,train_data[1])

         log_predictions_v2 = log_clf_v2.predict(development_data_v2)

         print ("Précision de la Logistic Regression V2 (avec les scores de polarité) : " + str

Précision de la Logistic Regression V2 (avec les scores de polarité) : 0.6809878012496281
```

Bien que minime, il y a eu une certaine amélioration dans le classificateur en intégrant les données de polarité.

Ceci conclut notre projet de construction d'un classificateur de polarité de base à 2 voies pour les tweets.