TP analyse des opinions dans les Tweets

Elaboré par : Mohamed DHAOUI - MS Big Data

L'objectif de ce TP est d'analyser un corpus de tweets en fonction des opinions exprimées (positif/- négatif). Le langage à utiliser est Python.

La base des tweets à analyser contient 498 tweets annotés manuellement. La base propose 6 champs correspondant aux informations suivantes :

1. la polarité du tweet : Chaque tweet est accompagné d'un score pouvant être égal à 0 (négatif), 2 (neutre) ou 4 (positif).
2. l'identifiant du tweet (2087)
3. la date du tweet (Sat May 16 23 :58 :44 UTC 2009)
4. la requête associée (lyx). Si pas de requête la valeur est NO_ QUERY.
5. l'utilisateur qui a tweeté (robotickilldozr)
6. le texte du tweet(Lyx is cool)

Importation des packages

In [552]:

```
# coding: utf8
import pandas as pd
import numpy as np
import re
import nltk as nltk
import re
from nltk.corpus import wordnet as wn
import csv



from nltk.corpus import wordnet as wn
from nltk.corpus import sentiwordnet as swn

import pandas as pd
from sklearn.decomposition.pca import PCA
from sklearn.externals import joblib
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from gensim.models import Word2Vec
import html.entities
```

1- Prétraitements

Dans cette partie , on va essayer de nettoyer les tweets et supprimer les caractères spéciaux susceptibles de nuire à la mise en place des méthodes d'analyse d'opinions. Pour ce faire , on va commencer par récuperer le fichier de tweet , ajouter les noms de colonnes et ensuite faire les étapes suivantes :

- récupérer le texte associé
- segmenter en tokens
- supprimer les urls
- nettoyer les caractères inhérents à la structure d'un tweet
- corriger les abréviations et les spécificités langagières des tweets à l'aide du dictionnaire DicoSlang (fichier SlangLookupTable.txt d

On commence par lire le fichier de tweets :

In [451]:

```
df = pd.read_csv('testdata.csv', sep = ',', header=None,encoding='latin1')
```

In [452]:

```
df.columns=["score","id","date","req","user","content"]
```

In [453]:

```
df.head()
```

Out[453]:

	score	id	date	req	user	content
0	4	3	Mon May 11 03:17:40 UTC 2009	kindle2	tpryan	@stellargirl I loooooooovvvvvveee my Kindle2. ...
1	4	4	Mon May 11 03:18:03 UTC 2009	kindle2	vcu451	Reading my kindle2... Love it... Lee childs i...
2	4	5	Mon May 11 03:18:54 UTC 2009	kindle2	chadfu	Ok, first assesment of the #kindle2 ...it fuck...
3	4	6	Mon May 11 03:19:04 UTC 2009	kindle2	SIX15	@kenburbary You'll love your Kindle2. I've had...
4	4	7	Mon May 11 03:21:41 UTC 2009	kindle2	yamarama	@mikefish Fair enough. But i have the Kindle2...

On crée ensuite des copies de dataset pour une utilisation future

In [454]:

```
df_init=df.copy()
```

In [455]:

```
df_temp=df.copy()

df_temp['content'] = df_temp['content'].apply(lambda x: re.sub('https?://[A-Za-z0-9./]
+','',x))
#df_temp['content'] = df_temp['content'].apply(lambda x: re.sub('([#])|([^a-zA-Z])','
 ',x))
```

In [461]:

```
df_temp.content[1]
```

Out[461]:

```
'Reading my kindle2...  Love it... Lee childs is good read.'
```

Prétraitement : '@'mention

Maintenant , on va essayer de traiter les mentions @ . Bien que ces mentions contiennent des informations (la personne qui a tweeté ou qui concerné par le tweet ou ...) , cela n'est pas pertinent dans le modèle d'analyse de sentiment . De ce fait ,on va les supprimer en utilisant le regex

Avant de supprimer les "@" , essayant de récuperer le nombre de mentions "@..." dans tout le corpus

In [465]:

```
nb_att=sum(df['content'].apply(lambda x: len(re.findall(r'@[A-Za-z0-9]+', x))))
print( "Nombre de mention '@' est %s" %nb_att)
```

```
Nombre de mention '@' est 124
```

C'est un peu étrange car on s'attendait à que le nombre de mentions de @ soit comparable au nombre de tweet ... Supprimons maintenant ces mentions

In [458]:

```
df.content[0]
```

Out[458]:

```
'@stellargirl I loooooooovvvvvveee my Kindle2. Not that the DX is cool, bu
t the 2 is fantastic in its own right.'
```

In [466]:

```
df['content'] = df['content'].apply(lambda x: re.sub(r'@[A-Za-z0-9]+','',x))
#[^a-zA-Z]", " "
```

In [467]:

```
df.content[0]
```

Out[467]:

```
' I loooooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is f
antastic in its own right.'
```

Prétraitement: URL links

Ensuite , on va supprimer les url links , ces url contiennent bien evidemment des informations mais qui ne sont pas utiles à l'analyse de sentiment

In [468]:

```
df['content'] = df['content'].apply(lambda x: re.sub('https?://[A-Za-z0-9./]+','',x))
```

In [470]:

```
df.content[17]
```

Out[470]:

```
'i love lebron. '
```

Prétraitement : Les hashtags

Commençons par calculer le nombre de hashtag dans le corpus

In [472]:

```
nb_hash=sum(df['content'].apply(lambda x: len(re.findall(r'([#])|([^a-zA-Z])', x))))
print( "Nombre de hashtag  est %s" %nb_hash)
```

```
Nombre de hashtag  est 8477
```

On s'attendait à que le nombre de hashtag soit aussi élévé car il fait partie des entités les plus trouvées dans les tweets . Malgré qu'ils peuvent apporter une information utile au sentiment , on préfère en premier lieu supprimer les hashtags pour facilier le traitement.Supprimons maintenant les Hashtags

In [473]:

```
df['content'] = df['content'].apply(lambda x: re.sub('([#])|([^a-zA-Z])',' ',x))
```

Prétraitement : Double espace

Ensuite , on va supprimer les double space des tweets

In [295]:

```
df['content'] = df['content'].apply(lambda x: re.sub(' +', ' ',x))
df.content[2]
```

Out[295]:

```
'Ok first assesment of the kindle it fucking rocks '
```

Calcul du nombre de caractère inhérents à la structure de tweet

Ici , on va supposer que le nombre d'occurrences des caractères inhérents à la structure de tweet est le nombre de hashtag plus nombre de mentions @ . De ce fait ce nombre est 8477+124 = **8601** caractères

Calcul du nombre hashtag

Le nombre de hashtag a été déjà calculer et il est égal à **8477**

Prétraitement : correction des abréviations et des spécificités langagières des tweets

Comme on a mentionné au début de cette partie , pour améliorer le nettoyage du fichier , on va utiliser un dictionnaire contenant les corrections des abréviations des tweets

Lecture des fichiers des abréviations

In [477]:

```
dictfile="SlangLookupTable.txt"
with open(dictfile) as f:
    reader = csv.reader(f, delimiter="\t")
    d = list(reader)
```

Transformation du fichier en dataframe

In [478]:

```
df_dict = pd.DataFrame(d)
df_dict.columns=['abv','mean']
```

In [479]:

```
df_dict.head(5)
```

Out[479]:

	abv	mean
0	121	one to one
1	a/s/l	age, sex, location
2	adn	any day now
3	afaik	as far as I know
4	afk	away from keyboard

Maintenant , on va créer une fonction `correcttabv` qui permet de reperer les abréviations dans les tweets (soit à la fin , soit , au début , soit au milieu) et les corriger

In [480]:

```
testtext="adn i do not know afk 121"
```

In [481]:

```
def correctabv(text) :
    for i in df_dict.abv :
        text=text.replace(" "+ i+ " ", " ")
        text=text.replace(" "+ i+ ",", " ")
        text=text.replace(" "+ i+ ".", " ")
        text=text.replace(" "+ i+ "!", " ")

        if testtext.find(i)+len(i) ==len(text):
            text=text.replace(i, " ")
        if testtext.find(i)==0:
            text=text.replace(i, " ")

    return text
```

In [483]:

```
correctabv(testtext)
```

Out[483]:

```
'  i do not know  '
```

Appliquons maintenant cette fonction au dataset

In [484]:

```
df['content'] =df['content'].apply(lambda x: correctabv(x))
```

Prétraitment : tokenize

La tokenization est la tache de splitter les une chaines de caractères en plusieurs mots . Cela peut se faire via la fonction split classique , mais afin de tenir compte de nature grammaticale des mots et le contexte dans lequel il a été employé , il vaut mieux utiliser la fonction `word_tokenize` de nltk

In [491]:

```
preProcessedTweet = df['content'].apply(lambda x: nltk.word_tokenize(x))
```

In [492]:

```
preProcessedTweet[0:5]
```

Out[492]:

```
0    [I, looooooooovvvvvveee, my, Kindle, Not, that,...
1    [Reading, my, kindle, Love, it, Lee, childs, i...
2    [Ok, first, assesment, of, the, kindle, it, fu...
3    [You, ll, love, your, Kindle, I, ve, had, mine...
4    [Fair, enough, But, i, have, the, Kindle, and,...
Name: content, dtype: object
```

2- Etiquetage grammatical

Maintenant on va s'interesser à la nature grammaticale des mots . Cela va nous servira après pour filtrer les mots . Pour ce faire, on va developper une fonction capable de déterminer la catégorie grammaticale (POS : Part Of Speech) de chaque mot du tweet

In [83]:

```
taggedTweet=preProcessedTweet.apply(lambda x: nltk.pos_tag(x))
taggedTweet[0:5]
```

Out[83]:

```
0    [(I, PRP), (looooooooovvvvvveee, VBP), (my, PRP...
1    [(Reading, VBG), (my, PRP$), (kindle, NN), (Lo...
2    [(Ok, NNP), (first, JJ), (assesment, NN), (of,...
3    [(You, PRP), (ll, VBP), (love, VB), (your, PRP...
4    [(Fair, NNP), (enough, RB), (But, CC), (i, NNS...
Name: content, dtype: object
```

On développe ensuite une fonction permettant de compter le nombre de verbes dans un tweet :

In [86]:

```
listverb_ind=["VB","VBD","VBG","VBN","VBP","VBZ"]

def countverb(listtoken):
    return np.sum([1 for i,j in listtoken if j in listverb_ind])
```

On affiche ensuite le nombre de verbes dans le corpus

In [494]:

```
nb_verbs=sum(taggedTweet.apply(lambda x: countverb(x)))
print( "Nombre de verbes est %s" %nb_verbs)
```

```
Nombre de verbes est 1119.0
```

3 . Algorithme de détection v1 : appel au dictionnaire Sentiwordnet

Dans cette partie , on a exploiter quelques fonctionalités intéressantes de NLTK : On utilisera la base de données wordnet qui permet d'accéder à l'ensemble des synsets qui sont liés à un mot donné à l'aide d'une commande simple sous Python.

Dans cette étape , on fera les taches suivantes :

- Récupération uniquement des mots correspondant à des adjectifs, noms, adverbes et verbes
- Calculer pour chaque mot du tweet les scores associés à leur premier synset
- Calculer pour chaque tweet la somme des scores positifs et négatifs des SentiSynsets du tweet, — Comparer la somme des scores positifs et des scores négatifs de chaque tweet pour décider de la classe à associer au tweet.

Pour commencer , on définit une liste contenant les tag à garder dans le tweet

In [312]:

```
listkeep=["JJ","FW","JJR","JJS","NN","NNS","NNP","NNPS","RB","RBR","RBS","VB","VBD","VB
G","VBN","VBP","VBZ"]
```

Ensuite , on définit une fonction `extractscoremot` qui renvoie les scores d'un mot donnée . Après on définit la fonction `scoretweet` qui prend comme input un tagwords de tweet , filtre les elements correspondant à {adj,noms,adv,verbes} , calcule le score de chaque mot et enfin renvoyer la somme des scores positifs et négatifs

In [496]:

```
def extractscoremot (mot):
    tt=wn.synsets(mot)
    if len(wn.synsets(mot)) > 0 :
        tt=tt[0].name() # à exploiter les autre options
        s=swn.senti_synset(tt)
        return s.pos_score(),s.neg_score()
    else :
        return 0,0

def scoretweet (tweettag):

    listnouns=[ x for (x,y) in tweettag if y in listkeep]
    return {'pos':np.sum([ extractscoremot(mot)[0] for mot in listnouns]),'neg':np.sum
([ extractscoremot(mot)[1] for mot in listnouns])}
```

Un exemple d'output correspondant à un tweettags :

In [497]:

```
scoretweet(taggedTweet[0])
```

Out[497]:

```
{'pos': 1.125, 'neg': 1.25}
```

On applique ensuite cette fonction à notre corpus prétraité

In [316]:

```
taggedTweetscore=taggedTweet.apply(lambda x: scoretweet(x))
```

On définit une fonction qui permettent de classer les tweet en {0,2,4}

In [498]:

```
def classscore(dictscore):
    if dictscore['pos'] > dictscore['neg'] :
        return [dictscore['pos'],dictscore['neg'],4]
    elif dictscore['pos'] < dictscore['neg'] : return [dictscore['pos'],dictscore['neg'
],0]
    else : return  [dictscore['pos'],dictscore['neg'],2]
```

In [499]:

```
taggedTweetClass=taggedTweetscore.apply( lambda x: classscore(x))
```

L'output de cette fonction est le suivant :

In [501]:

```
taggedTweetClass[0:5]
```

Out[501]:

```
0     [8.125, 1.25, 4]
1    [6.625, 0.125, 4]
2        [4.0, 7.0, 0]
3       [14.0, 3.0, 4]
4        [7.0, 2.0, 4]
Name: content, dtype: object
```

On récupérer ensuite la classe prédite pour chaque tweet et on calcule enfin la matrice de confusion :

In [502]:

```
predctedscore=taggedTweetClass.apply(lambda x:x[2])
```

In [503]:

```
df_confusion = pd.crosstab(df.score, predctedscore, rownames=['Actual'], colnames=['Predicted'], margins=True)
```

In [504]:

```
df_confusion
```

Out[504]:

Predicted	0	2	4	All
Actual				
0	65	7	105	177
2	15	12	112	139
4	10	5	167	182
All	90	24	384	498

Ici on voit bien que le nombre de tweet positif correctement prédits est **167** , ce qui est très moyen comme taux car on prédit beaucoup de positifs par rapports au réel (105 tweets neutres & 112 tweets négatifs sont classé positifs via notre algorithme) . On constate aussi que le nombre de tweet de classe 0 et dont la prédiction donne 4 est 105 , de ce fait , il y a un problème de distinction entre la classe neutre et la classe positive .On voit également qu'on a des erreurs de prédictions élévés sur les autres classes. Essayons de claculer la précision globale de l'algorithme .

In [508]:

```
score=(df_confusion[0:1][0].values[0] +df_confusion[1:2].values[0][1] +df_confusion[2:3
].values[0][2])/df_confusion[3:4].values[0][3]
print( "Le score de la première méthode est %.2f" %score)
```

```
Le score de la première méthode est 0.49
```

Le score est très faible et il reste encore du travail à effectuer ...

4- Algorithme de détection v2 : gestion de la négation et des modifieurs

Dans cette partie , on va essayer de tenir compte des négations et des modifieurs dans le scoring des tweets
On aura besoin de :

- La liste des mots en anglais correspondant à des négations
- La liste des mots correspondant aux modifieurs

On implémentera une fonctionqui ,pour chaque mot, effectue les opérations suivantes :

- multiplie par 2 le score négatif et le score positif associés au mot si le mot précédent est un modifieur ;
- utilise uniquement le score négatif du mot pour le score positif global du tweet et le score positif du mot pour le score négatif global du tweet si le mot précédent est une négation.

On commence par lire le dictionnaire de modifiers et des négations et les transformer en dataframe

In [513]:

```
dictfile="BoosterWordList.txt"

with open(dictfile) as f:
    reader = csv.reader(f, delimiter="\t")
    d = list(reader)
modifiers=pd.DataFrame(d)
modifiers.columns=["modifier","att"]
modifierslist=modifiers.modifier.tolist()
modifiers.head()
```

Out[513]:

	modifier	att
0	absolutely	1
1	definitely	1
2	extremely	2
3	fuckin	2
4	fucking	2

Pour les négations , on a pris les mots du dictionnaire fourni avec le TP et on les a mis dans une liste

In [105]:

```
negatives=["aren't","arent","can't","cannot","cant","don't","dont","isn't","isnt","neve
r","not","won't","wont","wouldn't","wouldnt"]
```

On développe ensuite une fonction qui prend un mot et son prédecesseur et renvoie les scores positif et négatif de tweet en tenant compte des régles expliquées au début de la partie

In [329]:

```
def extractscoremotmodif (mot,prec):
    tt=wn.synsets(mot)
    coef=1
    if len(wn.synsets(mot)) > 0 :
        if prec in (modifierslist):
            coef=2
        tt=tt[0].name() # à exploiter les autre options
        s=swn.senti_synset(tt)
        if prec in negatives :
            return coef*s.neg_score(),coef*s.pos_score(),1

        else :
            return coef*s.pos_score(),coef*s.neg_score(),0

    else :
        return 0,0,0
```

On implémente ensuite une fonction `scoretweetmodif` qui permet de scorer un tweettags en appelant la fonction précedement définie

In [111]:

```
def scoretweetmodif (tweettag):

    scorepos=[]
    listnouns=[ x for (x,y) in tweettag if y in listkeep]

    for ind,mot in enumerate(listnouns) :
        if ind > 0 :
            score = extractscoremotmodif (mot,listnouns[ind-1])
        else :
            score =extractscoremot(mot)
            score=(score[0],score[1],0)
        scorepos.append(score)
    return {'pos':np.sum([ mot[0] for mot in scorepos]),'neg':np.sum([mot[1] for mot in
scorepos]),
 'flag':np.sum([mot[2] for mot in scorepos])}
   # return scorepos
```

On calcule ensuite le nouveau vecteur de score

In [514]:

```
taggedTweetscore=taggedTweet.apply(lambda x: scoretweetmodif(x))
```

On récupère les termes négatifs contenus dans des tweets positifs dans une liste

In [ ]:

```
listneg=[ x for x in taggedTweetscore if (x['flag'] > 0 and x['pos'] > x ['neg']) ]
```

On calcule enfin le score et la matrice de confusion

In [515]:

```
taggedTweetscore=taggedTweet.apply(lambda x: scoretweetmodif(x))
taggedTweetClass=taggedTweetscore.apply( lambda x: classscore(x))
predctedscore=taggedTweetClass.apply(lambda x:x[2])
df_confusion = pd.crosstab(df.score, predctedscore, rownames=['Actual'], colnames=['Predicted'], margins=True)
```

In [516]:

```
df_confusion
```

Out[516]:

Predicted	0	2	4	All
Actual				
0	80	29	68	177
2	19	58	62	139
4	26	29	127	182
All	125	116	257	498

Le nombre de tweets negatifs ,neutres et positifs correctement détectés avec cette version de l'algorithme sont respectivement **80** , **58** et **127**

In [519]:

```
score=(df_confusion[0:1][0].values[0] +df_confusion[1:2].values[0][1] +df_confusion[2:3].values[0][2])/df_confusion[3:4].values[0][3]
print("Le score de cette version est %.2f" %score)
```

```
Le score de cette version est 0.53
```

On est arrivé à améliorer le score de 4% , ce qui n'est pas mal . On constate , via la matrice de confusion, que le traitement des négations et des modifieurs a permis de réduire l'erreur qu'on commet sur la classe 4 : L'algorithme a tendance au début de classifier les tweets en positifs , mais après ce traitement , il arrive à distinguer mieux le neutre et le négatif du positif . Essayons maintenant de creuser d'autres pistes

In [523]:

```
listneg=[ x for x in taggedTweetscore if (x['flag'] > 0 and x['pos'] > x ['neg']) ]
print( "Le nombre de termes négatifs contenus dans les tweets positifs est %s" %(len(listneg)))
```

```
Le nombre de termes négatifs contenus dans les tweets positifs est 4
```

5. Algorithme de détection v3 : gestion des emoticons

Maintenant , on va essayer de gérer les emoticons dans les tweets en utilisant le dictionnaire d'émoticons `EmoticonLookupTable.txt`

On va garder le vecteur de score obtenu avec la partie précédente et on contruisera également un autre score uniquement basé sur les emoticons , sur la base initiale de tweets , mais en supprimant les URL car elles contiennent `:/` .

- Les émoticons positifs rencontrés augmentent de 1 la valeur du score positif du tweet
- Les émoticons négatifs augmentent de 1 la valeur du score négatif du tweet.

On commence par lire le fichier des emoticons et le transformer en dataframe

In [117]:

```
dictfile="EmoticonLookupTable.txt"
with open(dictfile) as f:
    reader = csv.reader(f, delimiter="\t")
    d = list(reader)
```

In [118]:

```
emoticon=pd.DataFrame(d)
emoticon.columns=["symbol","score"]
```

In [119]:

```
emoticon.head()
```

Out[119]:

	symbol	score
0	%-(	-1
1	%-)	1
2	(-:	1
3	(:	1
4	(^ ^)	1

On transforme la colonne score en numérique

In [521]:

```
emoticon.score = pd.to_numeric(emoticon.score, errors='coerce')
```

On reparte de la base df_temp qui contient les tweets initiaux après avoir supprimer les urls

In [522]:

```
df_temp.head(3)
```

Out[522]:

	score	id	date	req	user	content
0	4	3	Mon May 11 03:17:40 UTC 2009	kindle2	tpryan	@stellargirl I loooooooovvvvvveee my Kindle2. ...
1	4	4	Mon May 11 03:18:03 UTC 2009	kindle2	vcu451	Reading my kindle2... Love it... Lee childs i...
2	4	5	Mon May 11 03:18:54 UTC 2009	kindle2	chadfu	Ok, first assesment of the #kindle2 ...it fuck...

On développe ensuite une fonction qui renvoie le score de chaque tweet en se basant uniquement sur les emoticons

In [122]:

```
def getsmileyscore ( tweet) :
    scorep,scoren=0,0
    for i,emo in enumerate(emoticon.symbol.tolist()) :
        if emo in tweet :
            if int(emoticon.score[i]) > 0 :
                scorep +=1
            else :
                scoren +=1
    return {'emop':scorep ,'emon':scoren}
```

Ci-desous un exemple d'output de la fonction

In [524]:

```
tweet="Reading my kindle2... Love it... Lee childs i... :/"
getsmileyscore(tweet)
```

Out[524]:

```
{'emop': 0, 'emon': 1}
```

On calcule ensuite le emoticon_score de tous les tweets :

In [525]:

```
smiley_score=df_temp.content.apply(lambda x: getsmileyscore(x))
```

On combine à la fin le score de v2 et le score d'emoticon pour calcule rle score final :

In [528]:

```
taggedTweetscoresmiley=taggedTweetscore.copy()
for  i,el in enumerate(taggedTweetscore) :
    c=taggedTweetscore[i].copy()
    taggedTweetscoresmiley[i]['pos']=taggedTweetscoresmiley[i]['pos']+smiley_score[i][
'emop']
    taggedTweetscoresmiley[i]['neg']=taggedTweetscoresmiley[i]['neg']+smiley_score[i][
'emon']
   # if c['pos']> c['neg'] and  (taggedTweetscoresmiley[i]['pos'] < taggedTweetscoresmi
ley[i]['neg'] or taggedTweetscoresmiley[i]['pos']== taggedTweetscoresmiley[i]['neg']):
    #    print([i])
```

Ci-dessous un extraint du vecteur de score obtenu :

In [529]:

```
taggedTweetscoresmiley[0:5]
```

Out[529]:

```
0     {'pos': 1.125, 'neg': 1.25, 'flag': 0}
1    {'pos': 1.625, 'neg': 0.125, 'flag': 0}
2        {'pos': 0.0, 'neg': 0.0, 'flag': 0}
3        {'pos': 4.0, 'neg': 2.0, 'flag': 1}
4        {'pos': 2.0, 'neg': 0.0, 'flag': 0}
Name: content, dtype: object
```

Calculons maintenant la nouvelle matrice de confusion et le score de l'algorithme

In [531]:

```
taggedTweetClassSmiley=taggedTweetscoresmiley.apply( lambda x: classscore(x))
predctedscoreSmiley=taggedTweetClassSmiley.apply(lambda x:x[2])
df_confusion_smiley = pd.crosstab(df.score, predctedscoreSmiley, rownames=['Actual'], c
olnames=['Predicted'], margins=True)
df_confusion_smiley
```

Out[531]:

Predicted	0	2	4	All
Actual				
0	88	27	62	177
2	19	58	62	139
4	26	24	132	182
All	133	109	256	498

Le nombre de tweets negatifs ,neutres et positifs correctement détectés avec cette version de l'algorithme sont respectivement **88** , **58** et **132**

In [533]:

```
score=(df_confusion_smiley[0:1][0].values[0] +df_confusion_smiley[1:2].values[0][1] +df
_confusion_smiley[2:3].values[0][2])/df_confusion_smiley[3:4].values[0][3]
score
print("Le score de version V3 est %.2f" %score)
```

```
Le score de version V3 est 0.56
```

In [540]:

```
nbre_emoticon=np.sum([smiley['emop'] + smiley['emon'] for smiley in smiley_score])
print("Le nombre d'emoticons  est %s" %nbre_emoticon)
```

```
Le nombre d'emoticons  est 57
```

Le fait de tenir compte des emoticons nous a permis d'améliorer le score de 3 % . Etant donnée le nombre faible d'emoticons dans le corpus , on ne peut pas esperer une amélioration significative . On voit également qu'on commet une erreur importante dans la distinction entre la classe neutre et la classe positive

6 . Algorithme de détection v4

Pour améliorer notre score , on suggère d'envisager une approche sur deux étapes :

- La première consiste à enrichir les dictionnaires de detection de sentiments et le traitmeent de ponctuation
- La deuxième consiste à utiliser un corpus existant de tweet labélisé 'positf' et 'negatif' disponible sur intrernet , entrainer un classifieur la dessus , ensuite prédire le score de chaque tweet de notre dataset (proba d'etre positif) et enfin ajouter la probabilité (si elle est très elevé ou très faible) au score obtenu de l'algorithme v3

Première étape : enrichissement et ajout des dicitionnaires

En premier lieu , on a commencé par enrichir les dictionnaires de négations et de modifieurs utilisés précedemment vu qu'ils comportent un nombre faible d'élements . Pour ce faire :

- on a mis à jour le dictionnaire des termes négatifs
- on a ajouté un autre dictionnaire d'adverbes scrappé du https://en.wiktionary.org/wiki/Category:English_degree_adverbs (https://en.wiktionary.org/wiki/Category:English_degree_adverbs)
- on a ajouté aussi les termes de ponctuations accentués comme '!!!' ou '???' qui pourrait avoir le meme effet que les adverbes

In [541]:

```
negatives = ["aint", "arent", "cannot", "cant", "couldnt", "darent", "didnt", "doesnt",
     "ain't", "aren't", "can't", "couldn't", "daren't", "didn't", "doesn't",
     "dont", "hadnt", "hasnt", "havent", "isnt", "mightnt", "mustnt", "neither",
     "don't", "hadn't", "hasn't", "haven't", "isn't", "mightn't", "mustn't",
     "neednt", "needn't", "never", "none", "nope", "nor", "not", "nothing", "nowhere",
     "oughtnt", "shant", "shouldnt", "uhuh", "wasnt", "werent",
     "oughtn't", "shan't", "shouldn't", "uh-uh", "wasn't", "weren't",
     "without", "wont", "wouldnt", "won't", "wouldn't", "rarely", "seldom", "despite",
"aren't","arent","can't","cannot","cant","don't","dont","isn't","isnt","never","not","won't",
           "wont","wouldn't","wouldnt"]
```

In [542]:

```
BOOSTER_DICT = ["absolutely", "amazingly","awfully", "completely", "considerably",
     "decidedly", "deeply", "effing", "enormously",
     "entirely", "especially", "exceptionally", "extremely",
     "fabulously", "flipping", "flippin",
     "fricking", "frickin", "frigging", "friggin", "fully", "fucking",
     "greatly", "hella", "highly" , "hugely", "incredibly",
     "intensely", "majorly", "more", "most", "particularly",
     "purely", "quite", "really", "remarkably",
     "so", "substantially",
     "thoroughly", "totally", "tremendously",
     "uber", "unbelievably", "unusually", "utterly",
     "very","too","quite",
     "almost", "barely", "hardly" , "just enough",
     "kind of", "kinda", "kindof", "kind-of",
     "less", "little", "marginally", "occasionally", "partly",
     "scarcely", "slightly", "somewhat",
     "sort of", "sorta", "sortof", "sort-of"]
PUNC_LIST = ["!!", "!!!", "??", "???", "?!?", "!?!", "?!?!", "!?!?"]
```

On a adapté ensuite nos fonctions de scoring pour qu'ils tiennent compte des nouveaux dictionnaires

In [543]:

```
def extractscoremotmodif (mot,prec):
    tt=wn.synsets(mot)
    coef=1
    if len(wn.synsets(mot)) > 0 :
        if prec in (modifierslist+BOOSTER_DICT+PUNC_LIST):
            coef=2
        tt=tt[0].name() # à exploiter les autre options
        s=swn.senti_synset(tt)
        if prec in negatives :
            return coef*s.neg_score(),coef*s.pos_score(),1

        else :
            return coef*s.pos_score(),coef*s.neg_score(),0

    else :
        return 0,0,0

def scoretweetmodif (tweettag):

    scorepos=[]
    listnouns=[ x for (x,y) in tweettag if y in listkeep]

    for ind,mot in enumerate(listnouns) :
        if ind > 0 :
            score = extractscoremotmodif (mot,listnouns[ind-1])
        else :
            score =extractscoremot(mot)
            score=(score[0],score[1],0)
        scorepos.append(score)
    return {'pos':np.sum([ mot[0] for mot in scorepos]),'neg':np.sum([mot[1] for mot in
scorepos]),
 'flag':np.sum([mot[2] for mot in scorepos])}
   # return scorepos
```

In [547]:

```
####### une fonction qui calcule le score à partir d'une matrice de confusion  ########
####
def getscore ( dfconfusion) :
    score=(dfconfusion[0:1][0].values[0] +dfconfusion[1:2].values[0][1] +dfconfusion[2:
3].values[0][2])/dfconfusion[3:4].values[0][3]
    return score
#taggedTweetscoremodif1=taggedTweet_beforv3.apply(lambda x: scoretweetmodif(x))
```

Deuxième étape : Modèle sur un corpus labélisé + et -

Comme on l'a précisé au début de la section , notre idée consiste à utiliser un jeux de données disponible sur internet qui contient des tweets labélisés { positif , négatif } , générer un embedding des tweets et ensuite entrainer un classifieur permettant d'estimer la probabilité qu'un tweet soit positif. Ensuite on utilise ce classifieur pour prédire les classe de tweet de notre jeux de données (en probabilité) :

- si la probabilité d'etre positif est supérieur à 0.7 , on ajoute un +1 au score positif du tweet obtenu de l'algorithme v3
- si la probabilité d'etre positif est inférieur à 0.3 , on ajoute un +1 au score négatif obtenu de l'algorithme v3

Le dataset 'Sentiment Analysis Dataset.csv' peut etre téléchargé via le lien suivant : http://thinknook.com/wp-content/uploads/2012/09/Sentiment-Analysis-Dataset.zip (http://thinknook.com/wp-content/uploads/2012/09/Sentiment-Analysis-Dataset.zip)

On commence par lire les données :

In [11]:

```
data = pd.read_csv('Sentiment Analysis Dataset.csv',
                       usecols=['Sentiment', 'SentimentText'], error_bad_lines=False)
```

In [145]:

```
data.head(25)
```

Out[145]:

	Sentiment	SentimentText
0	0	is so sad for my APL frie...
1	0	I missed the New Moon trail...
2	1	omg its already 7:30 :O
3	0	.. Omgaga. Im sooo im gunna CRy. I'...
4	0	i think mi bf is cheating on me!!! ...
5	0	or i just worry too much?
6	1	Juuuuuuuuuuuuuuuussssst Chillin!!
7	0	Sunny Again Work Tomorrow :-| ...
8	1	handed in my uniform today . i miss you ...
9	1	hmmmm.... i wonder how she my number @-)
10	0	I must think about positive..
11	1	thanks to all the haters up in my face a...
12	0	this weekend has sucked so far
13	0	jb isnt showing in australia any more!
14	0	ok thats it you win.
15	0	<-------- This is the way i feel right ...
16	0	awhhe man.... I'm completely useless rt no...
17	1	Feeling strangely fine. Now I'm gonna go l...
18	0	HUGE roll of thunder just now...SO scary!!!!
19	0	I just cut my beard off. It's only been gr...
20	0	Very sad about Iran.
21	0	wompppp wompp
22	1	You're the only one who can see this cause...
23	0	<---Sad level is 3. I was writing a mass...
24	0	... Headed to Hospitol : Had to pull out o...

Ici on définit quelques fonctions permettant de traiter les tweets de ce dataset

In [553]:

```
def html2unicode(s):
    # These are for regularizing HTML entities to Unicode:
    html_entity_digit_re = re.compile(r"&#\d+;")
    html_entity_alpha_re = re.compile(r"&\w+;")

    # First the digits:
    ents = set(html_entity_digit_re.findall(s))
    if len(ents) > 0:
        for ent in ents:
            entnum = ent[2:-1]
            try:
                entnum = int(entnum)
                s = s.replace(ent, unichr(entnum))
            except:
                pass

    # Now the alpha versions:
    ents = set(html_entity_alpha_re.findall(s))
    ents = filter((lambda x : x != "&amp;"), ents)

    for ent in ents:
        entname = ent[1:-1]
        try:
            s = s.replace(ent, unichr(html.entities.name2codepoint[entname]))
        except:
            pass
        s = s.replace("&amp;", " and ")

    return s
```

In [554]:

```
# The code below was adapted from Chris Potts' implementation at
# http://sentiment.christopherpotts.net/code-data/happyfuntokenizing.py

import re
#import htmlentitydefs
import html.entities

def get_regex_strings():
    return (
        # Phone numbers:
        r"""
        (?:
          (?:            # (international)
            \+?[01]
            [\-\s.]*
          )?
          (?:            # (area code)
            [\(]?
            \d{3}
            [\-\s.\)]*
          )?
          \d{3}          # exchange
          [\-\s.]*
          \d{4}          # base
        )"""
        ,
        # Emoticons:
        r"""
        (?:
          [<>]?
          [:;=8]                     # eyes
          [\-o\*\']?                 # optional nose
          [\)\]\(\[dDpP/\:\}\{@\|\\] # mouth
          |
          [\)\]\(\[dDpP/\:\}\{@\|\\] # mouth
          [\-o\*\']?                 # optional nose
          [:;=8]                     # eyes
          [<>]?
        )"""
        ,
        # HTML tags:
         r"""<[^>]+>"""
        ,
        # Twitter username:
        r"""(?:@[\w_]+)"""
        ,
        # Twitter hashtags:
        r"""(?:\#+[\w_]+[\w\'_\-]*[\w_]+)"""
        ,
        # Remaining word types:
        r"""
        (?:[a-z][a-z'\-_]+[a-z])       # Words with apostrophes or dashes.
        |
        (?:[+\-]?\d+[,/.:-]\d+[+\-]?)  # Numbers, including fractions, decimals.
        |
        (?:[\w_]+)                     # Words without apostrophes or dashes.
        |
        (?:\.(?:\s*\.){1,})            # Ellipsis dots.
        |
```

```
        (?:\S)                  # Everything else that isn't whitespace.
        """
        )


def html2unicode(s):
    # These are for regularizing HTML entities to Unicode:
    html_entity_digit_re = re.compile(r"&#\d+;")
    html_entity_alpha_re = re.compile(r"&\w+;")

    # First the digits:
    ents = set(html_entity_digit_re.findall(s))
    if len(ents) > 0:
        for ent in ents:
            entnum = ent[2:-1]
            try:
                entnum = int(entnum)
                s = s.replace(ent, unichr(entnum))
            except:
                pass

    # Now the alpha versions:
    ents = set(html_entity_alpha_re.findall(s))
    ents = filter((lambda x : x != "&amp;"), ents)

    for ent in ents:
        entname = ent[1:-1]
        try:
            s = s.replace(ent, unichr(html.entities.name2codepoint[entname]))
        except:
            pass
        s = s.replace("&amp;", " and ")

    return s


def tokenize(s):
    # This is the core tokenizing regex:
    word_re = re.compile(r"""(%s)""" % "|".join(get_regex_strings()), re.VERBOSE | re.U
NICODE)

    # The emoticon string gets its own regex so that we can preserve case for them as n
eeded:
    emoticon_re = re.compile(get_regex_strings()[1], re.VERBOSE | re.I | re.UNICODE)

    # Try to ensure unicode:


    # Fix HTML character entitites:
    s = html2unicode(s)

    # Tokenize:
    words = word_re.findall(s)

    # Possible alter the case, but avoid changing emoticons like :D into :d:
    words = map((lambda x : x if emoticon_re.search(x) else x.lower()), words)

    return words
```

On génère ensuite un embedding tf-idf des tweet prétraités via la commande `TfidfVectorizer`

In [555]:

```
print('Pre-processing tweet text...')
corpus = data['SentimentText']
vectorizer = TfidfVectorizer(decode_error='replace', strip_accents='unicode',
                                stop_words='english', tokenizer=tokenize)
X = vectorizer.fit_transform(corpus.values)
y = data['Sentiment'].values
```

```
Pre-processing tweet text...
```

On entraine ensuite un modèle **NaiveBayes** sur l'embedding des tweets (c'est le modèle basique , on pourra également essayer des approches plus sophistiquées comme les CNN et les RNN)

In [556]:

```
print('Training sentiment classification model...')
classifier = MultinomialNB()
classifier.fit(X, y)
```

```
Training sentiment classification model...
```

Out[556]:

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

Le score du classifieur sur le train est 83% , on pourra certainement l'améliorer mais , par faute de temps , on ne va pas creuser ce point .

In [557]:

```
classifier.score(X,y)
```

Out[557]:

```
0.8324175510922873
```

Maintenant , on recupère les données de notre dataset et on applique notre vectorizer pour générer l'embedding correspondant :

In [559]:

```
X_predict = vectorizer.transform(df_temp.content)
```

On prédit ensuite la probabilité d'avoir un sentiment positif de chaque tweet

In [560]:

```
proba_model=classifier.predict_proba(X_predict)
```

In [561]:

```
score_model_pos=[ i[1] for i in proba_model]
score_model_pos
```

Out[561]:

```
[0.5951628792743565,
 0.6165334225115324,
 0.6304053832021198,
 0.6200921456994368,
 0.6633596009306225,
 0.6843895971415599,
 0.11603542840026963,
 0.7185028645747751,
 0.765744695307406,
 0.6402629234268521,
 0.685392924249912,
 0.4365668207043725,
 0.6640085296999537,
 0.47029888887596905,
 0.2004342063906284,
 0.6183404144764137,
 0.20346297238884145,
 0.47563211725327753,
 0.41368438115792266,
 0.5285190717810705,
 0.6826535724485436,
 0.6518773687300844,
 0.5368696261540217,
 0.45841869516094863,
 0.3187564600924987,
 0.888466570555584,
 0.5043652116971936,
 0.7927869444936917,
 0.7621391251215481,
 0.7256761312331165,
 0.5598433590332862,
 0.7313032057001897,
 0.6155361288957714,
 0.30317204305859086,
 0.43398149971390554,
 0.21932919592317157,
 0.04353630944826192,
 0.2327365586202071,
 0.6244935318782446,
 0.5402898620329991,
 0.6347944149261476,
 0.49715395595128664,
 0.06262348325185872,
 0.17921457385602513,
 0.25737920418891813,
 0.42556571916782704,
 0.5000500885211578,
 0.5299301142153201,
 0.3935807032576889,
 0.2894044798335682,
 0.23357531723396374,
 0.5530415931995838,
 0.552118173634583,
 0.6371628946666724,
 0.46250512794004467,
 0.7759123848469365,
 0.8174662760592479,
 0.7170977242730162,
 0.6760449559620721,
```

```
    0.2884178617106853,
    0.5117126678225017,
    0.3399490168695719,
    0.6606903934421998,
    0.4535577797615171,
    0.27373020100265527,
    0.4191448636514004,
    0.8062699037000289,
    0.7625204447184881,
    0.5150101865196868,
    0.5790034196404823,
    0.6113484400521876,
    0.5678804711808574,
    0.33543070289828697,
    0.7506517175041125,
    0.6545827526001047,
    0.36887185616059504,
    0.6810349488641563,
    0.6427983475805169,
    0.36234169178404346,
    0.25284264122375527,
    0.7731174487192253,
    0.46835689740601627,
    0.8057117329420683,
    0.4322594046792255,
    0.7318470364737665,
    0.4975809813178676,
    0.2693404657078242,
    0.4655550709894883,
    0.17267569963149162,
    0.33653928231987873,
    0.30101923480846043,
    0.2650092893015356,
    0.07425102671095613,
    0.30424601975215654,
    0.1631329934811582,
    0.4180157951518258,
    0.09484827878651994,
    0.07243766360864863,
    0.2782735485664142,
    0.11349079493286622,
    0.5630230371066024,
    0.7849654049128679,
    0.4675287300129609,
    0.466881063203154,
    0.24074224224452648,
    0.12575354701953106,
    0.6738405288691457,
    0.6637590560392538,
    0.746863146002676,
    0.7734077564520687,
    0.8346844932511489,
    0.7107545619702547,
    0.7570172148761154,
    0.615525727246594,
    0.7135424187372947,
    0.6860216765338568,
    0.6557059703893084,
    0.6240635787374716,
    0.6216409911876895,
    0.6978222551455738,
```

```
    0.7480322216091324,
    0.7388716453957522,
    0.6969540240905843,
    0.2137476068428649,
    0.7633407114372281,
    0.7752902002957013,
    0.6892581120974363,
    0.7374424251180098,
    0.6655492092380779,
    0.6623099238802704,
    0.5911944195407057,
    0.7001932478425229,
    0.7317319835129087,
    0.7742540733806127,
    0.7135683131086331,
    0.6829910172931908,
    0.5029378229521777,
    0.4626784197270908,
    0.22735139993677211,
    0.31024066097571873,
    0.2791192247839125,
    0.1357412475161148,
    0.20199552963585174,
    0.05251157122940075,
    0.1413533629138166,
    0.2015388192414805,
    0.1803968292064523,
    0.11762677693703401,
    0.17130469153706665,
    0.12012059772966376,
    0.34548755667130043,
    0.2322031159684297,
    0.051789153895156294,
    0.5453354733401312,
    0.34841888219053874,
    0.4123220280682164,
    0.4490027249622885,
    0.38830973693873666,
    0.4920949389036155,
    0.47160100615545963,
    0.4730042617775219,
    0.1140708844821395,
    0.07344498526063753,
    0.2938055291571723,
    0.07540442143753351,
    0.3948144757620059,
    0.33377034948161494,
    0.19578702357310943,
    0.12519600200650996,
    0.20482436133495655,
    0.20699383265232307,
    0.3847354669107471,
    0.37867986901546324,
    0.5304465753326759,
    0.37269938344791,
    0.4306663721866257,
    0.2806922144298348,
    0.4086981427436494,
    0.699058692766329,
    0.6126995524510293,
    0.407654616014574,
```

```
    0.43621145179423,
    0.4476726194624092,
    0.4743392982293291,
    0.360757792160958,
    0.38700589996243157,
    0.5587629914487007,
    0.6587799149637192,
    0.6822966176319641,
    0.6103206238641253,
    0.6606897040125711,
    0.6873995484240546,
    0.627417210220911,
    0.5544520644299138,
    0.6370910395672079,
    0.5242021888725347,
    0.717966112277888,
    0.7387350951280276,
    0.5095093823212647,
    0.5577822679955095,
    0.5325369859148813,
    0.5957108770833612,
    0.7012221655687372,
    0.6880378768904927,
    0.3897250148246523,
    0.6222460686607539,
    0.7309881012373453,
    0.6759030365784948,
    0.525166114133204,
    0.510269072875708,
    0.22155819737024346,
    0.13684928129763724,
    0.4853132263049519,
    0.15860971115045708,
    0.13394467139368807,
    0.4243792541171926,
    0.632983182430268,
    0.24561281160342688,
    0.3400376404359438,
    0.20725290557658388,
    0.5545179326470231,
    0.08242323902532932,
    0.24330585183099926,
    0.22484272398033958,
    0.2331654065815157,
    0.12735969174531236,
    0.49635998090947925,
    0.2650250254759543,
    0.1965293656790886,
    0.249826402250685,
    0.062236477144318875,
    0.4005793638706687,
    0.7264952023909786,
    0.6707044011175926,
    0.3248268291527369,
    0.3991041910126962,
    0.8455746152862701,
    0.5961985640592965,
    0.7053000216589895,
    0.7700176938891441,
    0.3968118044134541,
    0.43196552748673595,
```

```
    0.6314817149772379,
    0.8271508102591253,
    0.7266879137800711,
    0.7107393923846198,
    0.7805770583703263,
    0.4433365191837099,
    0.8245262525597689,
    0.7822908948024843,
    0.23292254204289592,
    0.4290327697932566,
    0.6042649743048412,
    0.6588231230042351,
    0.441321031333753,
    0.47609477374057363,
    0.5073536603773947,
    0.5910895796486095,
    0.4398300065540129,
    0.553205145741692,
    0.512780743160208,
    0.5525032443665012,
    0.4773770261794203,
    0.5130939241521856,
    0.6741845909854445,
    0.4744404339244405,
    0.5964649895588253,
    0.5366685273291221,
    0.7687456755567351,
    0.3509388239005154,
    0.6657450437766822,
    0.1063924201645818,
    0.1056912901649402,
    0.7038315697922568,
    0.5440344803465789,
    0.7577729691033153,
    0.42220239981700264,
    0.7672802454821952,
    0.7262730247509566,
    0.5463247292701396,
    0.1879437765687972,
    0.18112651208303004,
    0.46265415027440177,
    0.39757861308938436,
    0.5210258074150597,
    0.7080683273248781,
    0.328783560915837,
    0.7757103027329113,
    0.6838289182584071,
    0.7558132740911715,
    0.8100222819534464,
    0.7508630439598019,
    0.5278737202510387,
    0.5067361859594647,
    0.7916199777397952,
    0.5354605847899053,
    0.046750708097724275,
    0.09906348336918724,
    0.42803128752441394,
    0.1839227682524785,
    0.4133985464733102,
    0.17495918717058137,
    0.3291060590170251,
```

```
    0.4866673176966451,
    0.1940069120990123,
    0.37093105363533374,
    0.7039609816470209,
    0.45006220483764064,
    0.5937458903482552,
    0.6086824762390479,
    0.6628057164929153,
    0.6261118686796424,
    0.29727832498021833,
    0.7071687009758137,
    0.19002378159449787,
    0.9224127398086459,
    0.6810245072546369,
    0.798607594044597,
    0.8186904774749,
    0.6781303860677775,
    0.5657466863460818,
    0.5341195602658921,
    0.7700277061815647,
    0.7721723938140591,
    0.6302966644263356,
    0.5482847667706573,
    0.747426360064437,
    0.5178735956715609,
    0.3424801800822271,
    0.5072218899455395,
    0.40170374285365695,
    0.5103769121955484,
    0.2934952968459115,
    0.29368287841624524,
    0.047578490173589345,
    0.6389963850911435,
    0.2820618359037801,
    0.7584582702402058,
    0.5550184623675756,
    0.7363762401258831,
    0.46850938310980145,
    0.5040500242061446,
    0.5246181725527984,
    0.20055788511035766,
    0.6827046588280982,
    0.5875068073642246,
    0.8354095214176629,
    0.9003236755616569,
    0.6964617648132484,
    0.8512393378228118,
    0.5133377353086939,
    0.6237517848139007,
    0.807557714346441,
    0.61188398702675,
    0.615363306456297,
    0.7582701742270852,
    0.8748526879795454,
    0.822997934919841,
    0.68299972614309,
    0.6788715319653305,
    0.6468687012148588,
    0.5083344085537586,
    0.7504629818947074,
    0.8979129517127402,
```

```
    0.6611141018499743,
    0.7496311441892165,
    0.5786174790700616,
    0.8025938200828603,
    0.678690802290243,
    0.7788850839131568,
    0.6385929091610673,
    0.4443076885841105,
    0.8163837829037889,
    0.18481717712364207,
    0.32006422603920276,
    0.5611440799784445,
    0.3165636279154283,
    0.3226507767735259,
    0.4813265798614263,
    0.4001208348992653,
    0.44867229260034625,
    0.1808007554433471,
    0.33824706638622737,
    0.09819868664139687,
    0.08987919066428686,
    0.10871944286669562,
    0.4431720490854325,
    0.30827236546502457,
    0.22135571250656946,
    0.3990112202797444,
    0.17900642419451973,
    0.16429796737821903,
    0.08932487694878402,
    0.23405862891774148,
    0.10172377302687811,
    0.3086413768345874,
    0.36782718401921116,
    0.15986181205265534,
    0.19675170904335382,
    0.5789352649889298,
    0.4722830349740498,
    0.5489669413383168,
    0.3083861672272533,
    0.24998459586194846,
    0.34981624047339227,
    0.3248139273449888,
    0.08543440196380497,
    0.3886434671789644,
    0.2386803958745216,
    0.20158387170550268,
    0.22453522402760406,
    0.4664611295785693,
    0.27481083204403406,
    0.5822454266701909,
    0.6275997189134324,
    0.4547516346781244,
    0.46844291140662603,
    0.11969487749745908,
    0.3906247835874505,
    0.7084208285968192,
    0.2163012384489504,
    0.40263841976354087,
    0.3820750419019757,
    0.6636580734678469,
    0.5871658638548239,
```

```
    0.4102255234302867,
    0.4239389271932645,
    0.5899358120432026,
    0.5963282334130859,
    0.3311779873714889,
    0.7123225072560115,
    0.5958494803020384,
    0.7313241420632404,
    0.5773497619001292,
    0.5817210707817758,
    0.8030657448821404,
    0.43545326813553836,
    0.5739768815435154,
    0.6243746000838994,
    0.5077431751480933,
    0.2745368696988524,
    0.5755071047478064,
    0.5775851055139678,
    0.6300096258624259,
    0.5838674916250357,
    0.5967776974126694,
    0.6613581549100018,
    0.5900042984137857,
    0.5763943182204346,
    0.7277298636827203,
    0.356078412926805,
    0.7804815451880597,
    0.6362906205490358,
    0.15530217115845726,
    0.6978463639554513,
    0.4920224970711249,
    0.566782815572941,
    0.6129609849189132,
    0.33915007702590677,
    0.5829910547715738,
    0.36168130664293724,
    0.8949091533868413,
    0.3795584595656294,
    0.7939273183767853,
    0.3391216738293966,
    0.455790819734106,
    0.6034469421636608,
    0.8825109379691077,
    0.10603601991617813,
    0.4871708338816775,
    0.617873415092733,
    0.8833348644621243,
    0.30916189115180054,
    0.4798442100994261,
    0.282876705697608,
    0.6548721148790241,
    0.7971147126311728,
    0.6047847304887485,
    0.7587767172368242,
    0.46421582317413657,
    0.35075786785032204,
    0.310354765053522,
    0.28709300580529834,
    0.40020716894044867,
    0.1588533327814174,
    0.2769976759024841,
```

```
0.5859503295749806,
0.7278098634806179,
0.7815262950740766,
0.7716822170564785,
0.41232739127367235,
0.7344134172843805,
0.3801833418415178,
0.6409141385838697,
0.2263465557772004,
0.6023047215968794,
0.10584501499550296,
0.10176987054791772]
```

On recupère après les scores des tweets de notre dernier algorithme du TP

In [156]:

```
taggedTweetscoresmileymodel=taggedTweetscoresmiley.copy()
```

On met à jour les scores en tenant compte de la probabilité issu du modèle NaiveBayes :

- si la probabilité d'etre positif est supérieur à 0.7 , on ajoute un +1 au score positif du tweet obtenu de l'algorithme v3
- si la probabilité d'etre positif est inférieur à 0.3 , on ajoute un +1 au score négatif obtenu de l'algorithme v3

In [562]:

```
#taggedTweetClassSmiley=taggedTweetscoresmiley.apply( lambda x: classscore(x))
for i,s in enumerate(taggedTweetscoresmiley) :
    if score_model_pos[i] > 0.7 :
        taggedTweetscoresmileymodel[i]['pos']=taggedTweetscoresmiley[i]['pos']+1

    if score_model_pos[i] < 0.3 :
        taggedTweetscoresmileymodel[i]['neg']=taggedTweetscoresmiley[i]['neg']+1
```

On calcule enfin la matrice de confusion et le score

In [563]:

```
taggedTweetClassSmileyModel=taggedTweetscoresmileymodel.apply( lambda x: classscore(x))
predctedscoreSmileymodel=taggedTweetClassSmileyModel.apply(lambda x:x[2])
df_confusion_smileymodel = pd.crosstab(df.score, predctedscoreSmileymodel, rownames=['Actual'], colnames=['Predicted'], margins=True)
df_confusion_smileymodel
```

Out[563]:

Predicted	0	2	4	All
Actual				
0	132	10	35	177
2	20	52	67	139
4	17	13	152	182
All	169	75	254	498

In [565]:

```
score=(df_confusion_smileymodel[0:1][0].values[0] +df_confusion_smileymodel[1:2].values[0][1] +df_confusion_smileymodel[2:3].values[0][2])/df_confusion_smileymodel[3:4].values[0][3]
score
print( "Le score du dernier algo est %.2f" %score)
```

```
Le score du dernier algo est 0.67
```

On arrive à améliorer le score initial de presque 10% , ce qui n'est pas mal de tout par rapport à l'effort fourni sur la partie modélisation .On arrive surtout à distinguer le neutre du positif via cette approche (dans v3 , il y avait 62 neutres qui sont classés positifs , et maintenant ça devient 32) . Pour continuer à améliorer le score , on pourra enrichir le dictionnaires des emoticons , utiliser des dictionnaires existant des mots +/- , des traducteurs de phrases (parfois on peut observer des mots en langues étrangères) , des termes anglais courants ...