

## Représentation des données textuelles

Il existe principalement deux méthodes qui permettent de transformer et d'extraire de l'information grâce à la tokenization. On parle de représentation sparse des données qui sont réalisés grâce à deux méthodes : bags-of-words & TF-IDF.

### bags-of-words(Sacs-de-mots):

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 texte = ["La vie est douce","La vie est tranquille, est belle, est douce"]
3 vect = CountVectorizer()
4 T= vect.fit_transform(texte)
5 dictionnaire_des_mots=vect.vocabulary_
6 print("dictionnaire_des_mots :", dictionnaire_des_mots)
7 liste_des_mots=list(dictionnaire_des_mots.keys())
8 print("liste_des_mots :", liste_des_mots)
9 Matrice_sparse_correspondante=T.toarray()
10 print("Matrice_sparse_correspondante:\n",Matrice_sparse_correspondante)
```

dictionnaire\_des\_mots : {'la': 3, 'vie': 5, 'est': 2, 'douce': 1, 'tranquille': 4, 'belle': 0}  
 liste\_des\_mots : ['la', 'vie', 'est', 'douce', 'tranquille', 'belle']  
 Matrice\_sparse\_correspondante:  
 [[0 1 1 1 0 1]  
 [1 1 3 1 1 1]]

## Représentation des données textuelles

### Matrice sparse binaire: **CountVectorizer(binary=True)**

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 texte = ["La vie est douce","La vie est tranquille et est belle"]
3 vect = CountVectorizer()
4 T= vect.fit_transform(texte)
5 dictionnaire_des_mots=vect.vocabulary_
6 print("dictionnaire_des_mots :", dictionnaire_des_mots)
7 liste_des_mots=list(dictionnaire_des_mots.keys())
8 print("liste_des_mots :", liste_des_mots)
9 Matrice_sparse_correspondante=T.toarray()
10 print("Matrice_sparse_correspondante:\n",Matrice_sparse_correspondante)
```

dictionnaire\_des\_mots : {'la': 4, 'vie': 6, 'est': 2, 'douce': 1, 'tranquille': 5, 'et': 3, 'belle': 0}  
 liste\_des\_mots : ['la', 'vie', 'est', 'douce', 'tranquille', 'et', 'belle']  
 Matrice\_sparse\_correspondante:  
 [[0 1 1 0 1 0 1]  
 [1 0 2 1 1 1 1]]

## Représentation des données textuelles

### TF-IDF (term frequency-inverse document frequency)

Les distances basées sur des tokens

- Une mesure qui est largement employée dans le domaine de la recherche d'informations, semble convenable ici. Il s'agit du TF/IDF .
- La fréquence de terme, TF, dans un document donné montre l'importance de ce terme dans le document en question.
- La fréquence inverse de document, IDF, est une mesure de l'importance générale du terme dans l'ensemble des documents.
- Les mesures de TF et TF/IDF sont définies comme suit :

Soit D un corpus des documents et t un terme à considérer:

le nombre  
d'occurrences du  
terme t dans un  
document

le nombre des  
termes dans ce  
document

$$TF = \frac{n(t)}{N} \text{ et } TF/IDF = TF \times \log\left(\frac{|D|}{d(t)}\right)$$

dénote le nombre des  
documents dans le  
corpus D

le nombre des  
documents qui  
contiennent au moins  
une fois le terme t.

## Représentation des données textuelles

```
import string
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
def netoyage(corpus_ensemble_documents):
    for i in range(len(corpus_ensemble_documents)):
        corpus_ensemble_documents[i]=corpus_ensemble_documents[i].lower()
    for i in range(len(corpus_ensemble_documents)):
        for c in string.punctuation:
            x=corpus_ensemble_documents[i].replace(c, ' ')
            corpus_ensemble_documents[i]=x
    stopwords_anglais=stopwords.words('english') #ou french
    for i in range(len(corpus_ensemble_documents)):
        L=corpus_ensemble_documents[i].split()
        for mot in L:
            if mot in stopwords_anglais:
                while mot in L:
                    L.remove(mot)
        corpus_ensemble_documents[i]=" ".join(L)
    return(corpus_ensemble_documents)
```

## Représentation des données textuelles

```

from numpy import *
def TF(terme,corpus,numero_document):
    x=corpus[numero_document].count(terme)
    y=len(corpus[numero_document].split())
    return x/y

def IDF(terme,corpus,numero_document):
    D=len(corpus)
    d_t=0
    for document in corpus:
        if terme in document:
            d_t+=1
    TF_val=TF(terme,corpus,numero_document)
    return TF_val*log(1+(D/d_t))

def cles_correspondante_a_valeur(valeur,dictionnaire):
    for cle in dictionnaire.keys():
        if dictionnaire[cle]==valeur:
            return cle

```

83

## Représentation des données textuelles

```

def matrice_sparse(dictionnaire,corpus_ensemble_documents):
    M=numpy.zeros((len(corpus_ensemble_documents),len(dictionnaire.values()))
    for i in range(len(corpus_ensemble_documents)):
        for j in dictionnaire.values():
            x=cles_correspondante_a_valeur(j,dictionnaire)
            M[i,j]=IDF(x,corpus_ensemble_documents,i)
    return M

def affiche(M):
    (n,p)=M.shape
    for i in range(n):
        for j in range(p):
            M[i,j]=round(M[i,j],2)
    print(M)

```

84

## Représentation des données textuelles

```
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
texte = ["La vie est douce","La vie est tranquille, est belle, est douce",
        "le corona-virus est méchant"]
texte=netoyage(texte)
vect = TfidfVectorizer()
T= vect.fit_transform(texte)
dictionnaire_des_mots=vect.vocabulary_
print("dictionnaire_des_mots :", dictionnaire_des_mots)
liste_des_mots=list(dictionnaire_des_mots.keys())
print("liste_des_mots :", liste_des_mots)
Matrice_sparse_correspondante=T.toarray()
print("Matrice_sparse_methode predefinie:\n")
affiche(Matrice_sparse_correspondante)
#on donne un poids important aux tokens qui apparaissent souvent dans un
#document en particulier mais pas dans tous les documents du corpus
print("Matrice_sparse obtenue par notre methode:\n")
affiche(matrice_sparse(dictionnaire_des_mots,texte))
```

dictionnaire\_des\_mots : {'vie': 5, 'douce': 2, 'tranquille': 4, 'belle': 0, 'corona': 1, 'virus': 6, 'méchant': 3}

liste\_des\_mots : ['vie', 'douce', 'tranquille', 'belle', 'corona', 'virus', 'méchant']

Matrice\_sparse\_methode predefinie:

```
[[0.  0.  0.71 0.  0.  0.71 0. ]
 [0.56 0.  0.43 0.  0.56 0.43 0. ]
 [0.  0.58 0.  0.58 0.  0.  0.58]]
```

Matrice\_sparse obtenue par notre methode:

```
[[0.  0.  0.46 0.  0.  0.46 0. ]
 [0.35 0.  0.23 0.  0.35 0.23 0. ]
 [0.  0.46 0.  0.46 0.  0.  0.46]]
```

85

## Application de la régression Logistique sur les Spams

```
import numpy as np
import pandas
spams = pandas.read_table("D:\\SMSSpamCollection.txt",sep="\t",header=0)
spamsTrain, spamsTest = train_test_split(spams,train_size=0.7,random_state=1)
from sklearn.feature_extraction.text import CountVectorizer
parseur = CountVectorizer()
XTrain = parseur.fit_transform(spamsTrain['message'])
mdtTrain = XTrain.toarray()
from sklearn.linear_model import LogisticRegression
modelFirst= LogisticRegression()
#essayer cette version, pourquoi ça ne marche pas ?
#modelFirst.fit(spamsTrain['message'],spamsTrain['classe'])
modelFirst.fit(mdtTrain,spamsTrain['classe'])
score1=modelFirst.score(mdtTrain, spamsTrain['classe'])
print("score sur data train:"+str(score1))
mdtTest = parseur.transform(spamsTest['message'])
#predTest = modelFirst.predict(mdtTest)
score2=modelFirst.score(mdtTest, spamsTest['classe'])
print("score sur data test:"+str(score2))
```

score sur data train:0.9976923076923077  
score sur data test:0.9838516746411483

Sans binary=True

86



## Application de la régression Logistique sur les Spams

```
import numpy as np
import pandas
spams = pandas.read_table("D:\\SMSSpamCollection.txt", sep="\t", header=0)
spamsTrain, spamsTest = train_test_split(spams, train_size=0.7, random_state=1)
from sklearn.feature_extraction.text import CountVectorizer
parseur = CountVectorizer(binary=True)
XTrain = parseur.fit_transform(spamsTrain['message'])
mdtTrain = XTrain.toarray()
from sklearn.linear_model import LogisticRegression
modelFirst = LogisticRegression()
#essayer cette version, pourquoi ça ne marche pas ?
#modelFirst.fit(spamsTrain['message'], spamsTrain['classe'])
modelFirst.fit(mdtTrain, spamsTrain['classe'])
score1 = modelFirst.score(mdtTrain, spamsTrain['classe'])
print("score sur data train:" + str(score1))
mdtTest = parseur.transform(spamsTest['message'])
#predTest = modelFirst.predict(mdtTest)
score2 = modelFirst.score(mdtTest, spamsTest['classe'])
print("score sur data test:" + str(score2))
```

score sur data train:0.9976923076923077  
score sur data test:0.9838516746411483

Avec binary=True

87

## Application de la régression Logistique sur les Spams

```
from sklearn import metrics
parseurBis = CountVectorizer(stop_words='english')
XTrainBis = parseurBis.fit_transform(spamsTrain['message'])
mdtTrainBis = XTrainBis.toarray()
modelBis = LogisticRegression()
modelBis.fit(mdtTrainBis, spamsTrain['classe'])
mdtTestBis = parseurBis.transform(spamsTest['message'])
score3 = modelBis.score(mdtTrainBis, spamsTrain['classe'])
print("score sur data train:" + str(score3))
score4 = modelBis.score(mdtTestBis, spamsTest['classe'])
print("score sur data test:" + str(score4))
```

score sur data train:0.9956410256410256  
score sur data test:0.9760765550239234

Sans stop\_words

88

## Application de la régression Logistique sur les Spams

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn import metrics
3 parseur3 = TfidfVectorizer()
4 XTrain3 = parseur3.fit_transform(spamsTrain['message'])
5 mdtTrain3 = XTrain3.toarray()
6 model3 = LogisticRegression()
7 model3.fit(mdtTrain3, spamsTrain['classe'])
8 mdtTest3 = parseur3.transform(spamsTest['message'])
9 score5=model3.score(mdtTrain3, spamsTrain['classe'])
10 print("score sur data train:"+str(score5))
11 score6=model3.score(mdtTest3, spamsTest['classe'])
12 print("score sur data test:"+str(score6))
```

```
score sur data train:0.9743589743589743
score sur data test:0.9712918660287081
```

Avec TfidfVectorizer