Ingénierie des données textuelles

De nombreuses applications utilisent des données textuelles pour faire de la prédiction : détection d'opinions, classification automatique de documents en fonction du contenu : spam - no spam, article sport vs article économie, etc...

La classification se fait de manière tout à fait classique par contre il est indispensable de traiter les documents pour pouvoir les faire interpréter par un classifieur. Le traitement des données textuelles est particulièrement difficile car il dépend des données disponibles et tout traitement n'est pas forcément justifié. Par exemple le fait de convertir tout le texte en minuscule peut faire perdre de l'information (Mr Play indique une personne et play un verbe), la suppression des ponctuations peut avoir des conséquences (! est très souvent utilisé pour la détection d'opinions), etc. En outre chaque langue possède aussi ses particularités et les librairies disponibles considèrent souvent l'anglais.

Il existe de nombreuses librairies qui aident à effectuer les prétraitements. Nous en présentons ici quelques unes en mettant en avant la librairie NLTK (Natural Language Toolkit) : http://www.nltk.org (http://www.nltk.org)
Il est conseillé de faire un :

import nltk
nltk.download('all')

pour avoir toutes les librairies.

Encodage des données

Les données textuelles sont souvent sujettes à des problèmes d'encodage (“Latin”, “UTF8” etc). Le plus simple est de les convertir dans un format classique (UTF8).

In [1]:

```
import unicodedata
chaine = u"Klüft skräms inför på fédéral électoral große"
chaine=unicodedata.normalize('NFKD', chaine).encode('ascii','ignore')
print (chaine)

```

```
b'Kluft skrams infor pa federal electoral groe'
```

Suppression des tags XML/HTML

Les données textuelles peuvent être issues de pages web, contenir des entêtes, etc..
L'une des premières étapes consistent à les nettoyer pour ne retenir que le texte.

La librairie BeautifulSoup permet de récupérer directement le texte en supprimant les tags :
https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

In [2]:

```
page = """
<!DOCTYPE html>
<html> <head> <title>Data-Driven Science: A New Paradigm?</title> </head>
  <body>
<h1>Michael L. Nelson</h1> (<a href=mln@cs.odu.edu>) is [here an example]
</body> </html>"""
```

In [3]:

```
print (page)
```

```
<!DOCTYPE html>
<html> <head> <title>Data-Driven Science: A New Paradigm?</title> </
head>
  <body>
<h1>Michael L. Nelson</h1> (<a href=mln@cs.odu.edu>) is [here an exa
mple]
</body> </html>
```

In [4]:

```
from bs4 import BeautifulSoup

def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

page=strip_html (page)
print (page)
```

```
  Data-Driven Science: A New Paradigm?

Michael L. Nelson () is [here an example]
```

De nombreuses modifications peuvent être réalisées à ce niveau notamment en utilisant des expressions régulières. Par exemple la fonction suivante permet de supprimer les textes entre crochets [].

In [5]:

```
import re
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)

page=remove_between_square_brackets(page)
print (page)
```

```
  Data-Driven Science: A New Paradigm?

Michael L. Nelson () is
```

Découpage en phrase

La tokenisation consiste à découper un document en phrases ou une phrase en mots (tokens). Dans un premier temps nous découpons notre document en phrase à l'aide de sent_tokenize.

In [6]:

```
document = """ Data Science is proving to be of paramount importance to the

This is due to the increased need more than 100000 for understanding the in

I'm not sure that we couldn't have a great success. """
```

In [7]:

```
import nltk
from nltk import sent_tokenize

phrases = sent_tokenize(document)
print("Phrase 0 : ",phrases[0])
print("Phrase 1 : ",phrases[1])
print("Phrase 2 : ",phrases[2])
```

```
Phrase 0 :   Data Science is proving to be of paramount importance t
o the "IT industry".
Phrase 1 :  This is due to the increased need more than 100000 for u
nderstanding the insurmountable amount of data being produced.
Phrase 2 :  I'm not sure that we couldn't have a great success.
```

Découpage en mot

La phrase est découpée en tokens à l'aide de word_tokenize

In [8]:

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize(phrases[0])
print(tokens)
```

```
['Data', 'Science', 'is', 'proving', 'to', 'be', 'of', 'paramount',
'importance', 'to', 'the', '``', 'IT', 'industry', "''", '.']
```

Nous pouvons constater que les ponctuations sont considérées comme des tokens.

Mise en minuscule

In [9]:

```
tokens = [w.lower() for w in tokens]
print (tokens)
```

```
['data', 'science', 'is', 'proving', 'to', 'be', 'of', 'paramount',
'importance', 'to', 'the', '``', 'it', 'industry', "''", '.']
```

Transformation des numériques en mots

In [10]:

```
#inflect est une librairie qui permet de convertir les nombres en mots
import inflect

tokens = word_tokenize(phrases[1])

print ("Nombre à convertir \n")
words = [word for word in tokens if word.isdigit()]
print(words)
p = inflect.engine()
numbertransf = [p.number_to_words(word) for word in tokens if word.isdigit

print ("Nombre après conversion \n")
print(numbertransf)


```

```
Nombre à convertir

['100000']
Nombre après conversion

['one hundred thousand']
```

Suppression des ponctuations

In [11]:

```
# Suppression de tous les termes qui ne sont pas alphanumériques
words = [word for word in tokens if word.isalpha()]
print(words)
```

```
['This', 'is', 'due', 'to', 'the', 'increased', 'need', 'more', 'tha
n', 'for', 'understanding', 'the', 'insurmountable', 'amount', 'of',
'data', 'being', 'produced']
```

Traitement des contractions

La dernière phrase contient des contractions. Voici ce que cela donne lorsque l'on obtient des tokens et que l'on supprime les caractères non alphabétiques. Des parties de négation disparaissent.

In [12]:

```
tokens = word_tokenize(phrases[2])
print(tokens)
words = [word for word in tokens if word.isalpha()]
print(words)
```

```
['I', "'m", 'not', 'sure', 'that', 'we', 'could', "n't", 'have', 'a'
, 'great', 'success', '.']
['I', 'not', 'sure', 'that', 'we', 'could', 'have', 'a', 'great', 's
uccess']
```

In [13]:

```
import contractions


def replace_contractions(text):
    return contractions.fix(text)

print ("Avant remplacement\n")
print (phrases[2])
print ("\nAprès remplacement\n")
laphrase=replace_contractions(phrases[2])
print (laphrase)
tokens = word_tokenize(laphrase)
print(tokens)
```

```
Avant remplacement

I'm not sure that we couldn't have a great success.

Après remplacement

I am not sure that we could not have a great success.
['I', 'am', 'not', 'sure', 'that', 'we', 'could', 'not', 'have', 'a'
, 'great', 'success', '.']
```

Suppression des stop words

Les stopwords sont des mots qui n'ont pas beaucoup d'intérêt dans la classification. Il s'agit des mots comme the, as etc. Attention il faut toujours faire attention à la liste des stopwords. Certains d'entre eux sont peut être utiles pour l'analyse. Le fait de supprimer is peut manquer par la suite. De plus il est parfois utile de mettre dans la liste des stopwords les mots très courant du domaine. Si les données parlent toutes de cinéma il faut mettre cinéma dans la liste des stopwords.

NLTK propose une liste de stopwords pour différentes langues. Ils sont en minuscule et avec des contractions. Pour les utiliser il faut donc avoir fait le même traitement sur nos données.
Il existe de nombreux sites qui proposent des listes de stopwords en différentes langues.

In [14]:

```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
print(stop_words)
stop_words = stopwords.words('french')
print(stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her',
'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 't
heir', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this
', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'w
ere', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', '
does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', '
because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', '
about', 'against', 'between', 'into', 'through', 'during', 'before',
'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'h
ere', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor',
'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't',
'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'coul
dn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn
't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
ghtn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "
shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't
", 'won', "won't", 'wouldn', "wouldn't"]
['au', 'aux', 'avec', 'ce', 'ces', 'dans', 'de', 'des', 'du', 'elle'
, 'en', 'et', 'eux', 'il', 'je', 'la', 'le', 'leur', 'lui', 'ma', 'm
ais', 'me', 'même', 'mes', 'moi', 'mon', 'ne', 'nos', 'notre', 'nous
', 'on', 'ou', 'par', 'pas', 'pour', 'qu', 'que', 'qui', 'sa', 'se',
'ses', 'son', 'sur', 'ta', 'te', 'tes', 'toi', 'ton', 'tu', 'un', 'u
ne', 'vos', 'votre', 'vous', 'c', 'd', 'j', 'l', 'à', 'm', 'n', 's',
't', 'y', 'été', 'étée', 'étées', 'étés', 'étant', 'étante', 'étants
', 'étantes', 'suis', 'es', 'est', 'sommes', 'êtes', 'sont', 'serai'
, 'seras', 'sera', 'serons', 'serez', 'seront', 'serais', 'serait',
'serions', 'seriez', 'seraient', 'étais', 'était', 'étions', 'étiez'
, 'étaient', 'fus', 'fut', 'fûmes', 'fûtes', 'furent', 'sois', 'soit
', 'soyons', 'soyez', 'soient', 'fusse', 'fusses', 'fût', 'fussions'
, 'fussiez', 'fussent', 'ayant', 'ayante', 'ayantes', 'ayants', 'eu'
, 'eue', 'eues', 'eus', 'ai', 'as', 'avons', 'avez', 'ont', 'aurai',
'auras', 'aura', 'aurons', 'aurez', 'auront', 'aurais', 'aurait', 'a
urions', 'auriez', 'auraient', 'avais', 'avait', 'avions', 'aviez',
'avaient', 'eut', 'eûmes', 'eûtes', 'eurent', 'aie', 'aies', 'ait',
'ayons', 'ayez', 'aient', 'eusse', 'eusses', 'eût', 'eussions', 'eus
siez', 'eussent']
```

In [15]:

```
from nltk.corpus import stopwords
print ("Exemple d'application des stopwords sur la première phrase\n")
tokens = word_tokenize(phrases[0])
tokens = [w.lower() for w in tokens]
words = [word for word in tokens if word.isalpha()]
print ("Avant transformation \n")
print (words)

stop_words = set(stopwords.words('english'))
words = [w for w in words if not w in stop_words]
print ("\nAprès transformation \n")
print(words)
```

```
Exemple d'application des stopwords sur la première phrase

Avant transformation

['data', 'science', 'is', 'proving', 'to', 'be', 'of', 'paramount',
'importance', 'to', 'the', 'it', 'industry']

Après transformation

['data', 'science', 'proving', 'paramount', 'importance', 'industry'
]
```

Stemmatisation

La stemmatisation (stemming ou racinisation) est l'opération qui consiste à réduire chaque mot à sa racine. Par exemple malad, maladie, maladive en maladive. Les approches utilisent l'algorithme de Porter qui se base sur le suffixe des mots. NTLK propose la librairie SnowballStemmer pour le français.

In [16]:

```
from nltk.stem.porter import PorterStemmer

tokens = word_tokenize(phrases[0])
tokens = [w.lower() for w in tokens]

porter = PorterStemmer()
stemmed = [porter.stem(word) for word in tokens]
print(stemmed)
```

```
['data', 'scienc', 'is', 'prove', 'to', 'be', 'of', 'paramount', 'im
port', 'to', 'the', '``', 'it', 'industri', "''", '.']
```

In [17]:

```
# un autre stemmatiser
from nltk.stem.lancaster import LancasterStemmer

lancaster_stemmer = LancasterStemmer()
lstemmed = [lancaster_stemmer.stem(word) for word in tokens]
print(lstemmed)
```

```
['dat', 'sci', 'is', 'prov', 'to', 'be', 'of', 'paramount', 'import'
, 'to', 'the', '``', 'it', 'industry', "''", '.']
```

In [18]:

```
# un autre stemmatiseur qui accepte le français
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("french")
phrase = "malade malades maladie maladies maladive"
tokens = word_tokenize(phrase)
print ("Avant transformation \n")
print (tokens)
stemmed = [stemmer.stem(word) for word in tokens]
print ("\n Après transformation\n")
print (stemmed)

```

```
Avant transformation
['malade', 'malades', 'maladie', 'maladies', 'maladive']

 Après transformation

['malad', 'malad', 'malad', 'malad', 'malad']
```

La lemmatisation

La lemmatisation consiste aussi à réduire chaque mot à sa racine mais par contre elle ne va pas s'intéresser au suffixe du mot. Elle effectue une première analyse pour mettre les verbes à l'infinitif, supprimer les s pour les pluriels.

Le choix de l'une ou de l'autre méthode dépend bien entendu de la tâche de classification que l'on souhaite faire.

In [19]:

```

from nltk.stem import WordNetLemmatizer

tokens = word_tokenize(phrases[0])
tokens = [w.lower() for w in tokens]

print ("Lemmatisation \n")
wordnet_lemmatizer = WordNetLemmatizer()
lstemmed = [wordnet_lemmatizer.lemmatize(word,pos='v') for word in tokens]
print("Lemmatisation : \n",lstemmed)

```

```
print ("\n A comparer avec la Stemmatisation\n")
lancaster_stemmer = LancasterStemmer()
lstemmed = [lancaster_stemmer.stem(word) for word in tokens]
print(lstemmed)


print("Lemmatisation : \n",lstemmed)
print ("\n autre exemple avec la phrase\n")
sentence = "have had having dogs crying"
print (sentence)
tokens = word_tokenize(sentence)


porter = PorterStemmer()
stemmed = [porter.stem(word) for word in tokens]
print("Stemmatisation : \n",stemmed)


wordnet_lemmatizer = WordNetLemmatizer()
lstemmed = [wordnet_lemmatizer.lemmatize(word,pos='v') for word in tokens]
print("Lemmatisation : \n",lstemmed)


```

```
Lemmatisation

Lemmatisation :
 ['data', 'science', 'be', 'prove', 'to', 'be', 'of', 'paramount', '
importance', 'to', 'the', '``', 'it', 'industry', "''", '.']
 A comparer avec la Stemmatisation

['dat', 'sci', 'is', 'prov', 'to', 'be', 'of', 'paramount', 'import'
, 'to', 'the', '``', 'it', 'industry', "''", '.']
Lemmatisation :
 ['dat', 'sci', 'is', 'prov', 'to', 'be', 'of', 'paramount', 'import
', 'to', 'the', '``', 'it', 'industry', "''", '.']

 autre exemple avec la phrase

have had having dogs crying
Stemmatisation :
 ['have', 'had', 'have', 'dog', 'cri']
Lemmatisation :
 ['have', 'have', 'have', 'dog', 'cry']
```

Part of speech tagging

Cette étape consiste à appliquer un analyseur morpho-syntaxique pour déterminer le genre du mot dans la phrase. Elle peut être très utile pour, par exemple, ne considérer que les adjectifs, les verbes ou les adverbes dans le cas de la détection de l'opinion.

In [20]:

```
1
2   tokens = word_tokenize(phrases[0])
3   tokens = [w.lower() for w in tokens]
4
5   nltk.pos_tag(tokens)
```

Out[20]:

```
[('data', 'NNS'),
 ('science', 'NN'),
 ('is', 'VBZ'),
 ('proving', 'VBG'),
 ('to', 'TO'),
 ('be', 'VB'),
 ('of', 'IN'),
 ('paramount', 'NN'),
 ('importance', 'NN'),
 ('to', 'TO'),
 ('the', 'DT'),
 ('``', '``'),
 ('it', 'PRP'),
 ('industry', 'NN'),
 ("''", "''"),
 ('.', '.')]
```

Il est possible de connaître l'intitulé de chaque Tag par la fonction nltk.help.upenn_tagset('XX') où XX représente le Tag recherché.

In [21]:

```
1   print (nltk.help.upenn_tagset('RB'))
2   print (nltk.help.upenn_tagset('VB'))
3
```

```
RB: adverb
    occasionally unabatingly maddeningly adventurously professedly
    stirringly prominently technologically magisterially predominate
ly
    swiftly fiscally pitilessly ...
None
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake bal
kanize
    bank begin behold believe bend benefit bevel beware bless boil b
omb
    boost brace break bring broil brush build ...
None
```

Il est possible d'intégrer le genre directement avec le mot à l'aide de la fonction pos_tag

In [22]:

```
from nltk import pos_tag

tokens = word_tokenize(phrases[0])

tokens=pos_tag(tokens)
print (tokens)
```

```
[('Data', 'NNP'), ('Science', 'NNP'), ('is', 'VBZ'), ('proving', 'VB
G'), ('to', 'TO'), ('be', 'VB'), ('of', 'IN'), ('paramount', 'NN'),
('importance', 'NN'), ('to', 'TO'), ('the', 'DT'), ('``', '``'), ('I
T', 'NNP'), ('industry', 'NN'), ("''", "''"), ('.', '.')]
```

Données de type tweet

Les tweets ont une syntaxe très particulière et généralement les traitements se font à l'aide d'expressions régulières.

In [23]:

```
tweet = '#NLP is thus an example :D ;) RT @theUser: see http://example.com
```

In [24]:

```
import re
#traitement des émoticones
emoticons_str = r"""
    (?:
        [:=;] # Eyes
        [oO\-]? # Nose (optional)
        [D\)\]\(\]/\\OpP] # Mouth
    )"""

#Prise en compte des éléments qui doivent être regroupés
regex_str = [
    emoticons_str,
    r'<[^>]+>', # HTML tags
    r'(?:@[\w_]+)', # @-mentions
    r"(?:\#+[\w_]+[\w\'_\-]*[\w_]+)", # hash-tags
    r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&amp;+]|[!*\(\),]|(?:%[0-9a-f][0-9a-

    r'(?:(?:\d+,?)+(?:\.?\d+)?)', # nombres
    r"(?:[a-z][a-z'\-_]+[a-z])", # mots avec - et '
    r'(?:[\w_]+)', # autres mots
    r'(?:\S)' # le reste
]

tokens_re = re.compile(r'('+'|'.join(regex_str)+')', re.VERBOSE | re.IGNORI
emoticon_re = re.compile(r'^'+emoticons_str+'$', re.VERBOSE | re.IGNORECASI

def tokenize(s):
    return tokens_re.findall(s)

def preprocess(s, lowercase=False):
    tokens = tokenize(s)
```

```
    if lowercase:
        tokens = [token if emoticon_re.search(token) else token.lower() for
    return tokens

# un example de tweet
tweet = '#NLP is thus an example :D RT @theUser: see http://example.com'
print ("Un exemple de tweet : \n",tweet)

print ("\nLe tweet avec un processus normal de transformation\n")
print (word_tokenize(tweet))
print ("\nLe tweet avec des expressions régulières\n")
words=preprocess(tweet)
print(words)
```

```
Un exemple de tweet :
 #NLP is thus an example :D RT @theUser: see http://example.com
(http://example.com)

Le tweet avec un processus normal de transformation

['#', 'NLP', 'is', 'thus', 'an', 'example', ':', 'D', 'RT', '@', 'th
eUser', ':', 'see', 'http', ':', '//example.com']

Le tweet avec des expressions régulières

['#NLP', 'is', 'thus', 'an', 'example', ':D', 'RT', '@theUser', ':',
'see', 'http://example.com']
```