

Classification de données textuelles

Lors de l'étape d'ingénierie de données textuelles nous avons vu que diverses opérations pouvaient être appliquées sur les textes et qu'au final il est possible d'obtenir des textes simplifiés. Nous allons, à présent, étudier comment faire de la classification à partir de données textuelles et comment convertir les textes en vecteurs pour pouvoir faire de la classification.

Vectorisation

Maintenant qu'un document a été transformé en une séquence de mot, il est nécessaire de la transformer en vecteur. C'est le rôle de la vectorisation.

Bag of Words

La manière la plus simple de vectorisation est d'utiliser les Bag of Words (BOW). Il s'agit, à partir d'une liste de mots (vocabulaire) de compter le nombre d'apparition du mot du vocabulaire dans le document.

Cette opération se fait par :

1. Création d'une instance de la classe CountVectorizer
2. Appel de la fonction fit() pour apprendre le vocabulaire à partir de document
3. Appel de la fonction transform() sur un ou plusieurs documents afin de les encoder dans le vecteur.

Attention, par défaut, CountVectorizer effectue un certain nombre de pré-traitements comme par exemple mise en minuscule. Voir [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

[learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

In [1]:

```
1  from sklearn.feature_extraction.text import CountVectorizer
2
3  texte = ["This is an example of CountVectorizer for creating a vector"]
4
5  vectorizer = CountVectorizer()
6  # creation du vocabulaire
7  vectorizer.fit(texte)
8  # Contenu du vocabulaire
9  print(vectorizer.vocabulary_)
10 # encodage du document
11 vector = vectorizer.transform(texte)
12
13 print ("Taille du vecteur :\n",vector.shape)
14
```

```
{'this': 7, 'is': 5, 'an': 0, 'example': 3, 'of': 6, 'countvectorize
r': 1, 'for': 4, 'creating': 2, 'vector': 8}
```

Taille du vecteur :

```
(1, 9)
```

Il est donc à présent possible de traiter un ensemble de documents comme le montre l'exemple suivant. Nous créons également un dataframe.

In [2]:

```
1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer
3 corpus = [
4     'This is my first document.',
5     'This is the document 2 !',
6     'Maybe this is the third document?',
7     'Anything else? may be 40',
8     'Yes !! this is the last one'
9 ]
10 vectorizer = CountVectorizer()
11 X = vectorizer.fit_transform(corpus)
12
13 # vectorizer.get_feature_names()
14 # contient le vocabulaire
15
16
17 df = pd.DataFrame(
18     data=vectorizer.transform(corpus).toarray(),
19     columns=vectorizer.get_feature_names()
20 )
21
22 display(df)
```

	40	anything	be	document	else	first	is	last	may	maybe	my	one	the	third	this
0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1
2	0	0	0	1	0	0	1	0	0	1	0	0	1	1	1
3	1	1	1	0	1	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	0	0	1	1	0	0	0	1	1	0	1

Prise en compte des prétraitements

Considérons les prétraitements suivants qui permettent de supprimer les caractères non Ascii, de mettre en minuscule, d'enlever les ponctuations, de remplacer les nombres et d'enlever les stopwords.

In [3]:

```
1 from nltk.tokenize import word_tokenize
2 import unicodedata
3 import re
4 import inflect
5 from nltk.corpus import stopwords
6
7 def remove_non_ascii(words):
8     new_words = []
9     for word in words:
```

```

10         new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'ignore')
11         new_words.append(new_word)
12     return new_words
13
14     def to_lowercase(words):
15         new_words = []
16         for word in words:
17             new_word = word.lower()
18             new_words.append(new_word)
19         return new_words
20
21     def remove_punctuation(words):
22         new_words = []
23         for word in words:
24             new_word = re.sub(r'^\w\s', '', word)
25             if new_word != '':
26                 new_words.append(new_word)
27         return new_words
28
29     def replace_numbers(words):
30         p = inflect.engine()
31         new_words = []
32         for word in words:
33             if word.isdigit():
34                 new_word = p.number_to_words(word)
35                 new_words.append(new_word)
36             else:
37                 new_words.append(word)
38         return new_words
39
40     def remove_stopwords(words):
41         new_words = []
42         for word in words:
43             if word not in stopwords.words('english'):
44                 new_words.append(word)
45         return new_words
46
47     def normalize(words):
48         words = remove_non_ascii(words)
49         words = to_lowercase(words)
50         words = replace_numbers(words)
51         words = remove_punctuation(words)
52         words = remove_stopwords(words)
53         return words
54
55     def clean_text(text):
56         tokens = word_tokenize(text)
57         tokens=normalize(tokens)
58         text=" ".join([" "+i for i in tokens]).strip()
59         return text
60
61

```

In [4]:

```
1  texte="we have to think that is text is 1000 *#better than. the "  
2  print ("Texte avant le nettoyage \n")  
3  print (texte)  
4  texte=clean_text(texte)  
5  print ("Texte après le nettoyage \n")  
6  print (texte)
```

Texte avant le nettoyage

we have to think that is text is 1000 *#better than. the

Texte après le nettoyage

think text one thousand better

L'appel aux fonctions de prétraitements peut se faire directement dans CountVectorizer. Attention cependant il est préférable de ne pas le faire. Par exemple dans le cas d'un pipeline et d'un gridsearch le prétraitement sera effectué à chaque fois ! Il est par contre utile de le faire lors de la dernière étape et que le modèle est sauvegardé pour permettre qu'un nouveau document puisse être transformé avant d'être mis sous la forme d'un vecteur (voir plus bas).

In [5]:

```
1  print (len(corpus))  
2  ▼  for i in range(len(corpus)):  
3      corpus[i]=clean_text(corpus[i])
```

5

In [6]:

```
1
2  #ici le preprocessor ne sert à rien
3  # car les données ont été nettoyées avant.
4  ▼ vectorizer = CountVectorizer(
5      preprocessor=clean_text
6  )
7
8  X = vectorizer.fit_transform(corpus)
9
10
11 ▼ df = pd.DataFrame(
12     data=vectorizer.transform(corpus).toarray(),
13     columns=vectorizer.get_feature_names()
14 )
15
16 display(df)
```

	anything	document	else	first	forty	last	may	maybe	one	third	two	yes
0	0	1	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	1	0	1	0	0
3	1	0	1	0	1	0	1	0	0	0	0	0
4	0	0	0	0	0	1	0	0	1	0	0	1

Appel dans CountVectorizer

In [7]:

```
1
2  ▼ corpus = [
3      'This is my first document.',
4      'This is the document 2 !',
5      'Maybe this is the third document?',
6      'Anything else? may be 40',
7      'Yes !! this is the last one'
8  ]
9  #Rappel ce n'est pas efficace
10 #il vaut mieux traiter les données avant
11 #attention aux pipelines
12  ▼ vectorizer = CountVectorizer(
13      preprocessor=clean_text
14  )
15
16  X = vectorizer.fit_transform(corpus)
17
18
19  ▼ df = pd.DataFrame(
20      data=vectorizer.transform(corpus).toarray(),
21      columns=vectorizer.get_feature_names()
22  )
23
24  display(df)
```

	anything	document	else	first	forty	last	may	maybe	one	third	two	yes
0	0	1	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	1	0	1	0	0
3	1	0	1	0	1	0	1	0	0	0	0	0
4	0	0	0	0	0	1	0	0	1	0	0	1

TfidfVectorizer

CountVectorizer en prenant en compte l'occurrence des mots est souvent trop limité. Une alternative est d'utiliser la mesure TF-IDF (Term Frequency – Inverse Document) via une instance de la classe TfidfVectorizer. Le principe est le même que pour CountVectorizer.

Remarque : Il est possible si CountVectorizer a déjà été utilisé de le faire suivre par TfidfTransformer pour simplement mettre à jour les valeurs.

In [8]:

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
2  corpus = [
3      'This is my first document.',
4      'This is the document 2 !',
5      'Maybe this is the third document?',
6      'Anything else? may be 40',
7      'Yes !! this is the last one'
8  ]
9  vectorizer = TfidfVectorizer()
10 X = vectorizer.fit_transform(corpus)
11
12 # vectorizer.get_feature_names()
13 # contient le vocabulaire
14
15
16 df = pd.DataFrame(
17     data=vectorizer.transform(corpus).toarray(),
18     columns=vectorizer.get_feature_names()
19 )
20
21 display(df)
```

	40	anything	be	document	else	first	is	last	may
0	0.000000	0.000000	0.000000	0.381399	0.000000	0.569497	0.320844	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.541107	0.000000	0.000000	0.455196	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.356359	0.000000	0.000000	0.299781	0.000000	0.000000
3	0.447214	0.447214	0.447214	0.000000	0.447214	0.000000	0.000000	0.000000	0.447214
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.278803	0.494873	0.000000

In [9]:

```
1  ▼ # Appel d'un pré-prétraitement
2  ▼ vectorizer = TfidfVectorizer(
3      preprocessor=clean_text
4  )
5
6  X = vectorizer.fit_transform(corpus)
7
8
9  ▼ df = pd.DataFrame(
10     data=vectorizer.transform(corpus).toarray(),
11     columns=vectorizer.get_feature_names()
12 )
13
14 display(df)
```

	anything	document	else	first	forty	last	may	maybe	one	third	1
0	0.0	0.556451	0.0	0.830881	0.0	0.00000	0.0	0.00000	0.00000	0.00000	0.00000
1	0.0	0.556451	0.0	0.000000	0.0	0.00000	0.0	0.00000	0.00000	0.00000	0.830881
2	0.0	0.427993	0.0	0.000000	0.0	0.00000	0.0	0.63907	0.00000	0.63907	0.00000
3	0.5	0.000000	0.5	0.000000	0.5	0.00000	0.5	0.00000	0.00000	0.00000	0.00000
4	0.0	0.000000	0.0	0.000000	0.0	0.57735	0.0	0.00000	0.57735	0.00000	0.00000

Appel de fonction de prétraitement dans TfidfVectorizer

In [10]:

```
1  ▼ # Appel d'un pré-prétraitement
2  ▼ vectorizer = TfidfVectorizer(
3      preprocessor=clean_text
4  )
5
6  X = vectorizer.fit_transform(corpus)
7
8
9  ▼ df = pd.DataFrame(
10     data=vectorizer.transform(corpus).toarray(),
11     columns=vectorizer.get_feature_names()
12 )
13
14 display(df)
15
```

	anything	document	else	first	forty	last	may	maybe	one	third	1
0	0.0	0.556451	0.0	0.830881	0.0	0.00000	0.0	0.00000	0.00000	0.00000	0.000
1	0.0	0.556451	0.0	0.000000	0.0	0.00000	0.0	0.00000	0.00000	0.00000	0.830
2	0.0	0.427993	0.0	0.000000	0.0	0.00000	0.0	0.63907	0.00000	0.63907	0.000
3	0.5	0.000000	0.5	0.000000	0.5	0.00000	0.5	0.00000	0.00000	0.00000	0.000
4	0.0	0.000000	0.0	0.000000	0.0	0.57735	0.0	0.00000	0.57735	0.00000	0.000

Les ngrams

Très souvent il est utile de prendre en compte les n-grammes, i.e. la suite de n mots consécutifs car ils peuvent être important pour la classification. Il est tout à fait possible de les obtenir soit pendant l'étape de prétraitement, soit lors de l'étape de vectorisation en le passant en paramètre.

In [11]:

```
1  ▼ # Appel d'un pré-prétraitement
2  ▼ vectorizer = TfidfVectorizer(
3      preprocessor=clean_text,
4      ngram_range=(1, 2)
5  )
6
7  X = vectorizer.fit_transform(corpus)
8
9
10 ▼ df = pd.DataFrame(
11     data=vectorizer.transform(corpus).toarray(),
12     columns=vectorizer.get_feature_names()
13 )
14
15 display(df)
```

	anything	anything else	document	document two	else	else may	first	first document	forl
0	0.000000	0.000000	0.427993	0.000000	0.000000	0.000000	0.63907	0.63907	0.000000
1	0.000000	0.000000	0.427993	0.63907	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.317527	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.377964	0.377964	0.000000	0.000000	0.377964	0.377964	0.000000	0.000000	0.37796
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

5 rows × 21 columns

Exemple de classification

A partir du moment où nous disposons d'une matrice, nous pouvons appliquer toutes les approches que nous avons vu précédemment.

Nous illustrons au travers d'un exemple de classification multiclasse. Ce dernier est tiré de "The 20 newsgroups text dataset". Il s'agit d'un jeu de données de 20 newsgroups comprenant à peu près 18000 news sur 20 sujets différents.

Ce jeu de données est disponible sous scikit learn qui propose des fonctions pour le manipuler : <https://scikit-learn.org/stable/datasets/index.html#newsgroups-dataset> (<https://scikit-learn.org/stable/datasets/index.html#newsgroups-dataset>)

fetch_20newsgroups permet de charger le fichier. Il est possible de récupérer un jeu d'entrainement, de test ou l'ensemble.

Présentation du jeu de données

In [12]:

```
1 from sklearn.datasets import fetch_20newsgroups
2 news = fetch_20newsgroups(subset='all')
3 from pprint import pprint
4 print ("liste des topics \n")
5 pprint(list(news.target_names))
```

liste des topics

```
['alt.atheism',
'comp.graphics',
'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware',
'comp.sys.mac.hardware',
'comp.windows.x',
'misc.forsale',
'rec.autos',
'rec.motorcycles',
'rec.sport.baseball',
'rec.sport.hockey',
'sci.crypt',
'sci.electronics',
'sci.med',
'sci.space',
'soc.religion.christian',
'talk.politics.guns',
'talk.politics.mideast',
'talk.politics.misc',
'talk.religion.misc']
```

Téléchargement d'une partie des topics

In [13]:

```
1 categories = ['alt.atheism', 'talk.religion.misc',
2               'rec.sport.hockey', 'comp.graphics', 'sci.space']
3
4 news = fetch_20newsgroups(subset='all',
5                           categories=categories)
6
7 print ("Taille du jeu de données\n")
8 print (news filenames.shape)
9
10 print ("Un exemple de données\n")
11 print (news.data[5], '\n TOPIC : ', news.target[5],
12         '\n*****')
```

Taille du jeu de données

(4386,)

Un exemple de données

From: LMARSHA@cms.cc.wayne.edu (Laurie Marshall)

Subject: Re: WHERE ARE THE DOUBTERS NOW? HMM?
Article-I.D.: cms.16BA79DBA.LMARSHA
Organization: Wayne State University, Detroit MI U.S.A.
Lines: 22
NNTP-Posting-Host: cms.cc.wayne.edu

In article <1993Apr4.051942.27095@ramsey.cs.laurentian.ca>
maynard@ramsey.cs.laurentian.ca (Roger Maynard) writes:

>
>For those of you who can only decide which team is best after you have
>seen the standings:
>
>TOR 42 25 11 95 .609
>CHI 42 25 11 95 .609
>DET 44 28 9 97 .599
>VAN 41 28 9 91 .583
>
>No team in the Campbell Conference has a better record than Toronto
>.

That's true, but according to your stats, Chicago has just as good a
record as Toronto. It's interesting that you should list Toronto ahead
of Chicago.

Laurie Marshall
Wayne State University
Detroit, Michigan
Go Wings!!

TOPIC : 2

Nettoyage des données

In [14]:

```
1  
2 ▼ def clean_news (data):  
3 ▼     for i in range(len(data)):  
4         #print (i)  
5         data[i]=clean_text(data[i])  
6     return data  
7  
8
```

In [15]:

```
1 news.data=clean_news(news.data)
```

Vectorisation

In [16]:

```
1 vectorizer = TfidfVectorizer()  
2 vectors = vectorizer.fit_transform(news.data)  
3  
4
```

X et y

In [17]:

```
1 ▼ #Specification des variables à prédire et de la classe X et y  
2 X = vectors.toarray()  
3  
4 y = news.target  
5
```

Création du jeu d'apprentissage et de test

In [18]:

```
1 from sklearn.model_selection import train_test_split  
2  
3 validation_size=0.3 #30% du jeu de données pour le test  
4  
5 testsize= 1-validation_size  
6 seed=30  
7 ▼ X_train,X_test,y_train,y_test=train_test_split(X,  
8                                                  y,  
9                                                  train_size=validation_size,  
10                                                 random_state=seed,  
11                                                 test_size=testsize)  
12  
13
```

Utilisation du classifieur

In [19]:

```
1 from sklearn.naive_bayes import GaussianNB
2 from sklearn.model_selection import KFold
3 from sklearn.model_selection import cross_val_score
4 from time import time
5
6 seed=7
7 k_fold = KFold(n_splits=10, shuffle=True, random_state=seed)
8 clf = GaussianNB()
9
10 scoring = 'accuracy'
11 t0 = time()
12 score = cross_val_score(clf, X, y, cv=k_fold, scoring=scoring)
13 print("Réalisé en %0.3fs" % (time() - t0))
14
15 ▼ print('Les différentes accuracy pour les 10 évaluations sont : \n',
16       score, '\n')
17 ▼ print('Accuracy moyenne : ', score.mean(),
18       ' standard deviation', score.std())
```

Réalisé en 166.380s

Les différentes accuracy pour les 10 évaluations sont :

```
[0.94305239 0.9498861 0.94305239 0.94077449 0.93394077 0.93621868
0.93835616 0.94292237 0.91552511 0.95205479]
```

Accuracy moyenne : 0.939578327664576 standard deviation 0.009600959296048284

Mise en place d'un pipeline

In [20]:

```
1 from sklearn.linear_model import SGDClassifier
2 from sklearn.pipeline import Pipeline
3 from sklearn.feature_extraction.text import TfidfTransformer
4 from sklearn.metrics import accuracy_score, confusion_matrix
5 from time import time
6 from sklearn.metrics import classification_report
7
8 ▼ pipeline = Pipeline([('vect', TfidfVectorizer()),
9 ▼                      ('clf', SGDClassifier(loss='hinge',
10                                           penalty='l2',
11                                           alpha=1e-3,
12                                           random_state=42,
13                                           max_iter=5, tol=None)),
14                      ])
15
16
17
18
19 X=news.data
20 y=news.target
21
22
23 ▼ X_train,X_test,y_train,y_test=train_test_split(X,
```

```

24                                     y,
25                                     train_size=validation_size,
26                                     random_state=seed,
27                                     test_size=testsize)
28
29
30 t0 = time()
31 pipeline.fit(X_train, y_train)
32 print("Fit réalisé en %0.3fs" % (time() - t0))
33
34 t0 = time()
35 result = pipeline.predict(X_test)
36 print("Prédiction réalisée en %0.3fs" % (time() - t0))
37
38 print('\n accuracy:',accuracy_score(result, y_test),'\n')
39
40
41
42
43 conf = confusion_matrix(y_test, result)
44 print ('\n matrice de confusion \n',conf)
45
46
47
48 print ('\n',classification_report(y_test, result))
49
50

```

/Users/pascalponcelet/Desktop/Sicki-learn/Tools/tools/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:183: FutureWarning: max_iter and tol parameters have been added in SGDClassifier in 0.19. If max_iter is set but tol is left unset, the default value for tol in 0.19 and 0.20 will be None (which is equivalent to -infinity, so it has no effect) but will change in 0.21 to 1e-3. Specify tol to silence this warning.

FutureWarning)

Fit réalisé en 0.532s
Prédiction réalisée en 0.603s

accuracy: 0.9397590361445783

matrice de confusion

```
[[510   6   1   5  24]
 [  1 665   4   4   1]
 [  0   9 722   0   1]
 [  1  21   4 653   1]
 [ 82   9   3   8 336]]
```

	precision	recall	f1-score	support
0	0.86	0.93	0.89	546
1	0.94	0.99	0.96	675
2	0.98	0.99	0.98	732
3	0.97	0.96	0.97	680
4	0.93	0.77	0.84	438
micro avg	0.94	0.94	0.94	3071
macro avg	0.94	0.93	0.93	3071
weighted avg	0.94	0.94	0.94	3071

Mise en place d'un gridsearch avec pipeline pour rechercher le meilleur classifieur

Dans cette section nous intégrons un pipeline complet : lancement du TfidfVectorizer , utilisation de deux classifieurs (DecisionTreeClassifier et SGDClassifier) avec les hyperparamètres pour évaluer le meilleur via GridSearchCV. Attention le processus de gridsearch est très long.

In [21]:

```
1  from sklearn.model_selection import train_test_split
2  from sklearn.model_selection import GridSearchCV
3  from sklearn.pipeline import Pipeline
4  from sklearn.tree import DecisionTreeClassifier
5  from sklearn.linear_model import SGDClassifier
6  from time import time
7  from sklearn.svm import SVC
8  import pickle
9
10
11  # Specification des pipelines
12  # programmation à optimiser par une fonction :)
13  ▼ pipeline_SGDC = Pipeline([('tfidf', TfidfVectorizer()),
14                             ('clf', SGDClassifier())])
15
16
17  ▼ parameters_SGDC = [
18      {'clf__max_iter': (5,),
19       'clf__alpha': (0.00001, 0.000001),
20       'clf__penalty': ('l2', 'elasticnet')}
21  ]
22
```



```

23 ▼ pipeline_DT = Pipeline([('tfidf', TfidfVectorizer()),
24                             ('clf', DecisionTreeClassifier())])
25
26
27 #param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
28 param_range = [1, 5, 8, 10]
29 ▼ parameters_DT = [
30 ▼     {'clf__min_samples_leaf': param_range,
31         'clf__criterion': ['gini', 'entropy'],
32         'clf__max_depth': param_range,
33         'clf__min_samples_split': param_range[1:]}
34 ]
35
36
37
38
39 X=news.data
40 y=news.target
41
42
43 ▼ X_train,X_test,y_train,y_test=train_test_split(X,
44                                                     y,
45                                                     train_size=validation_size,
46                                                     random_state=seed,
47                                                     test_size=testsize)
48
49 # Creation des GridSearchCV avec les pipelines spécifiques
50
51 ▼ gs_SGDC = GridSearchCV(pipeline_SGDC,
52                           parameters_SGDC,
53                           cv=3,
54                           n_jobs=-1,
55                           scoring='accuracy')
56
57
58 ▼ gs_DT = GridSearchCV(pipeline_DT,
59                         parameters_DT,
60                         cv=3,
61                         n_jobs=-1,
62                         scoring='accuracy')
63
64
65
66 grids = [gs_SGDC, gs_DT]
67 grid_dict={0:'Linear classifiers', 1:'Decision Tree'}
68
69 best_acc = 0.0
70 best_clf = 0.0
71 best_gs = ''
72
73 ▼ for idx,gs in enumerate(grids):
74     print('\nClassifier: %s' % grid_dict[idx])
75     t0 = time()
76     gs.fit(X_train, y_train)
77     print("Fit réalisé en %0.3fs" % (time() - t0))
78
79     print('Meilleurs paramètres : %s' % gs.best_params_)

```

```

80
81     print("Meilleur score d'accuracy sur l'entrainement: %.3f" % gs.best_s
82     # Prediction sur le jeu de test avec les meilleurs paramètres
83     t0 = time()
84     result = gs.predict(X_test)
85     print("Prédiction réalisée en %0.3fs" % (time() - t0))
86
87     print("Score d'accuracy pour les meilleurs paramètres sur jeu de test
88
89     print ('\n matrice de confusion \n',confusion_matrix(y_test, result))
90
91     print ('\n',classification_report(y_test, result))
92
93     #Modele avec la meilleure accuracy sur le jeu de test
94     if accuracy_score(y_test, result) > best_acc:
95         best_acc = accuracy_score(y_test, result)
96         best_gs = gs
97         best_clf = idx
98
99
100
101     print('\nClassifieur avec la meilleur accuracy sur le jeu de test\n',
102           grid_dict[best_clf])
103
104

```

Classifieur: Linear classifiers

/Users/pascalponcelet/Desktop/Sicki-learn/Tools/tools/lib/python3.6/site-packages/sklearn/linear_model/stochastic_gradient.py:183: FutureWarning: max_iter and tol parameters have been added in SGDClassifier in 0.19. If max_iter is set but tol is left unset, the default value for tol in 0.19 and 0.20 will be None (which is equivalent to -infinity, so it has no effect) but will change in 0.21 to 1e-3. Specify tol to silence this warning.

FutureWarning)

Fit réalisé en 8.424s

Meilleurs paramètres : {'clf__alpha': 1e-06, 'clf__max_iter': 5, 'clf__penalty': 'l2'}

Meilleur score d'accuracy sur l'entrainement: 0.913

Prédiction réalisée en 0.682s

Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.9

33

matrice de confusion

```

[[495  5  0  5 41]
 [ 3 653 13  4  2]
 [ 1 10 716  3  2]
 [ 7 17  8 643  5]
 [ 58  8  8  7 357]]

```

	precision	recall	f1-score	support
0	0.88	0.91	0.89	546
1	0.94	0.97	0.95	675
2	0.96	0.98	0.97	732
3	0.97	0.95	0.96	680

	4	0.88	0.82	0.84	438
micro avg		0.93	0.93	0.93	3071
macro avg		0.93	0.92	0.92	3071
weighted avg		0.93	0.93	0.93	3071

Classifier: Decision Tree

Fit réalisé en 108.210s

Meilleurs paramètres : {'clf__criterion': 'entropy', 'clf__max_depth': 10, 'clf__min_samples_leaf': 8, 'clf__min_samples_split': 10}

Meilleur score d'accuracy sur l'entraînement: 0.634

Prédiction réalisée en 0.712s

Score d'accuracy pour les meilleurs paramètres sur jeu de test : 0.631

matrice de confusion

```
[[293  44  16 141  52]
 [  3 518  14 140   0]
 [  3 108 473 148   0]
 [  1 121   7 550   1]
 [142  52  13 128 103]]
```

	precision	recall	f1-score	support
0	0.66	0.54	0.59	546
1	0.61	0.77	0.68	675
2	0.90	0.65	0.75	732
3	0.50	0.81	0.62	680
4	0.66	0.24	0.35	438
micro avg	0.63	0.63	0.63	3071
macro avg	0.67	0.60	0.60	3071
weighted avg	0.67	0.63	0.62	3071

Classifier avec la meilleur accuracy sur le jeu de test
Linear classifiers

Recupération du meilleur classifieur avec ses paramètres

Une fois le résultat obtenu, il est possible de récupérer le meilleur classifieur et ses paramètres. Ici il s'agit de Linear Classifier (SGDClassifier). Dans la suite nous montrons comment relancer une classification via un pipeline et une sauvegarde. Attention ici dans le pipeline nous ajoutons le clean_text (cf remarques précédentes) et nous sauvegardons le modèle pour pouvoir l'utiliser après avec d'autres données.

Le fait de mettre le clean_text dans le pipeline permet lors de la sauvegarde de celui-ci de pouvoir lorsqu'il y a de nouvelles données de les relancer dans le pipeline (les données récupéreront donc la matrice du TfidfVectorizer et les pré-traitements associés).

In [23]:

```
1 from sklearn.linear_model import SGDClassifier
2 from sklearn.pipeline import Pipeline
3 from sklearn.feature_extraction.text import TfidfTransformer
```

```

1  from sklearn.feature_extraction.text import TfidfTransformer
2  from sklearn.metrics import accuracy_score, confusion_matrix
3  from time import time
4  from sklearn.metrics import classification_report
5
6  #Recupération des données pour l'exemple
7  #et partir proprement
8
9  categories = ['alt.atheism', 'talk.religion.misc',
10               'rec.sport.hockey', 'comp.graphics', 'sci.space']
11
12  news = fetch_20newsgroups(subset='all',
13                           categories=categories)
14
15
16
17
18
19
20
21  pipeline = Pipeline([('vect', TfidfVectorizer(preprocessor=clean_text)),
22                       ('clf', SGDClassifier(loss='hinge',
23                                             penalty='l2',
24                                             alpha=1e-05,
25                                             random_state=42,
26                                             max_iter=5,
27                                             tol=None)),
28                      ])
29
30
31
32
33  X=news.data
34  y=news.target
35
36
37  X_train,X_test,y_train,y_test=train_test_split(X,
38                                                  y,
39                                                  train_size=validation_size,
40                                                  random_state=seed,
41                                                  test_size=testsize)
42
43
44  t0 = time()
45  print ("Lancement du fit \n")
46  pipeline.fit(X_train, y_train)
47  print("Fit réalisé en %0.3fs" % (time() - t0))
48
49  t0 = time()
50  print ("Lancement de la prédiction \n")
51  result = pipeline.predict(X_test)
52  print("Prédiction réalisée en %0.3fs" % (time() - t0))
53
54  print('\n accuracy:',accuracy_score(result, y_test),'\n')
55
56  conf = confusion_matrix(y_test, result)
57  print ('\n matrice de confusion \n',conf)
58
59
60

```

```

61 print ('\n',classification_report(y_test, result))
62
63 print("\nSauvegarde du pipeline grid search")
64 filename = 'thebestone.pkl'
65 pickle.dump(pipeline, open(filename, 'wb'))
66
67

```

Lancement du fit

```

/Users/pascalponcelet/Desktop/Sicki-learn/Tools/tools/lib/python3.6/
site-packages/sklearn/linear_model/stochastic_gradient.py:183: Future
Warning: max_iter and tol parameters have been added in SGDClassifi
er in 0.19. If max_iter is set but tol is left unset, the default va
lue for tol in 0.19 and 0.20 will be None (which is equivalent to -i
nfinity, so it has no effect) but will change in 0.21 to 1e-3. Speci
fy tol to silence this warning.
  FutureWarning)

```

Fit réalisé en 92.018s

Lancement de la prédiction

Prédiction réalisée en 211.909s

accuracy: 0.9296646043633996

matrice de confusion

```

[[495   5   7   8  31]
 [  4 647  12   9   3]
 [  0   7 723   1   1]
 [  5  20  13 637   5]
 [ 64   8   9   4 353]]

```

	precision	recall	f1-score	support
0	0.87	0.91	0.89	546
1	0.94	0.96	0.95	675
2	0.95	0.99	0.97	732
3	0.97	0.94	0.95	680
4	0.90	0.81	0.85	438
micro avg	0.93	0.93	0.93	3071
macro avg	0.92	0.92	0.92	3071
weighted avg	0.93	0.93	0.93	3071

Sauvegarde du pipeline grid search

Utilisation de nouvelles données

L'objectif ici est d'utiliser de nouvelles données à partir du modèle appris. Lors de la sauvegarde le pipeline entier a été sauvegardé. Cela implique que lorsque l'on va vouloir prédire les prétraitements du pipeline vont être appliqués.

In [25]:

```
1 import pickle
2 from sklearn.metrics import accuracy_score
3 from sklearn.metrics import confusion_matrix
4 from sklearn.metrics import classification_report
5
6 print ("Chargement du modèle \n")
7 filename = 'thebestone.pkl'
8 clf_loaded = pickle.load(open(filename, 'rb'))
9
10
11 print ("A partir d'un nouveau texte\n")
12 print ("Utilisation d'un texte de 20newsgroup\n")
13 categories = ['alt.atheism', 'talk.religion.misc',
14               'rec.sport.hockey', 'comp.graphics', 'sci.space']
15
16 news = fetch_20newsgroups(subset='all',
17                             categories=categories)
18
19 print ("Sélection aléatoire de 20 documents \n")
20 from random import randint
21 samples=[]
22 samples_result=[]
23 sample_new=[]
24 for i in range(1,20):
25     val=randint(1,4385)
26     sample_new.append(val)
27     samples.append(news.data[val])
28     samples_result.append(news.target[val])
29
30 print ("Prédiction des news sélectionnées\n")
31
32
33
34 result = clf_loaded.predict(samples)
35
36 print ("Valeurs réelles vs. valeurs prédites\n")
37 for i in range(len(result)):
38     print ("News : ",sample_new[i],
39           "\t réelle ",
40           samples_result[i],
41           " prédite ",
42           result [i])
```

Chargement du modèle

A partir d'un nouveau texte

Pour l'exemple utilisation d'un texte de 20newsgroup

Sélection aléatoire de 20 documents

Prédiction des news sélectionnées

Valeurs réelles vs. valeurs prédites

News : 906 réelle 2 prédite 2

News :	3141	réelle	3	prédite	3
News :	2906	réelle	4	prédite	0
News :	693	réelle	2	prédite	2
News :	3454	réelle	1	prédite	1
News :	2993	réelle	3	prédite	3
News :	4080	réelle	3	prédite	3
News :	4042	réelle	1	prédite	1
News :	1520	réelle	2	prédite	2
News :	2277	réelle	2	prédite	2
News :	1769	réelle	3	prédite	3
News :	3359	réelle	2	prédite	2
News :	2642	réelle	1	prédite	1
News :	4247	réelle	1	prédite	1
News :	1871	réelle	0	prédite	0
News :	3921	réelle	1	prédite	1
News :	976	réelle	3	prédite	3
News :	1132	réelle	3	prédite	3
News :	2475	réelle	2	prédite	2