1 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.

1.1 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-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 [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer

texte = ["This is an example of CountVectorizer for creating a vector"]

vectorizer = CountVectorizer()
# creation du vocabulaire
vectorizer.fit(texte)
# Contenu du vocabulaire
print(vectorizer.vocabulary_)
# encodage du document
vector = vectorizer.transform(texte)

print ("Taille du vecteur :\n",vector.shape)

```

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

In [ ]:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
corpus = [
    'This is my first document.',
    'This is the document 2 !',
    'Maybe this is the third document?',
    'Anything else? may be 40',
    'Yes !! this is the last one'
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

# vectorizer.get_feature_names())
# contient le vocabulaire


df = pd.DataFrame(
    data=vectorizer.transform(corpus).toarray(),
    columns=vectorizer.get_feature_names()
)

display(df)
```

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 [ ]:

```
from nltk.tokenize import word_tokenize
import unicodedata
import re
import inflect
from nltk.corpus import stopwords

def remove_non_ascii(words):
    new_words = []
    for word in words:
        new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'ignor
        new_words.append(new_word)
    return new_words

def to_lowercase(words):
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

def remove_punctuation(words):
    new_words = []
    for word in words:
        new_word = re.sub(r'[^\w\s]', '', word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

def replace_numbers(words):
    p = inflect.engine()
    new_words = []
    for word in words:
        if word.isdigit():
            new_word = p.number_to_words(word)
            new_words.append(new_word)
        else:
            new_words.append(word)
    return new_words

def remove_stopwords(words):
    new_words = []
    for word in words:
        if word not in stopwords.words('english'):
            new_words.append(word)
    return new_words

def normalize(words):
    words = remove_non_ascii(words)
    words = to_lowercase(words)
    words = replace_numbers(words)
    words = remove_punctuation(words)
    words = remove_stopwords(words)
    return words

def clean_text(text):
    tokens = word_tokenize(text)
    tokens=normalize(tokens)
    text="".join([" "+i for i in tokens]).strip()
    return text
```

```
60
61
```

In [ ]:

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

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 [ ]:

```
print (len(corpus))
for i in range(len(corpus)):
    corpus[i]=clean_text(corpus[i])
```

In [ ]:

```

#ici le preprocessor ne sert à rien
# car les données ont été nettoyées avant.
vectorizer = CountVectorizer(
    preprocessor=clean_text
)

X = vectorizer.fit_transform(corpus)


df = pd.DataFrame(
    data=vectorizer.transform(corpus).toarray(),
    columns=vectorizer.get_feature_names()
)

display(df)
```

Appel dans CountVectorizer

In [ ]:

```
corpus = [
    'This is my first document.',
    'This is the document 2 !',
    'Maybe this is the third document?',
    'Anything else? may be 40',
    'Yes !! this is the last one'
]
#Rappel ce n'est pas efficace
#il vaut mieux traiter les données avant
#attention aux pipelines
vectorizer = CountVectorizer(
    preprocessor=clean_text
)

X = vectorizer.fit_transform(corpus)


df = pd.DataFrame(
    data=vectorizer.transform(corpus).toarray(),
    columns=vectorizer.get_feature_names()
)

display(df)
```

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 [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = [
    'This is my first document.',
    'This is the document 2 !',
    'Maybe this is the third document?',
    'Anything else? may be 40',
    'Yes !! this is the last one'
]
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)

# vectorizer.get_feature_names())
# contient le vocabulaire


df = pd.DataFrame(
    data=vectorizer.transform(corpus).toarray(),
    columns=vectorizer.get_feature_names()
)

display(df)
```

In [ ]:

```
# Appel d'un pré-prétraiment
vectorizer = TfidfVectorizer(
    preprocessor=clean_text
)

X = vectorizer.fit_transform(corpus)


df = pd.DataFrame(
    data=vectorizer.transform(corpus).toarray(),
    columns=vectorizer.get_feature_names()
)

display(df)
```

Appel de fonction de prétraitement dans TfidfVectorizer

In [ ]:

```
# Appel d'un pré-prétraiment
vectorizer = TfidfVectorizer(
    preprocessor=clean_text
)

X = vectorizer.fit_transform(corpus)


df = pd.DataFrame(
    data=vectorizer.transform(corpus).toarray(),
    columns=vectorizer.get_feature_names()
)

display(df)

```

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 [ ]:

```
# Appel d'un pré-prétraiment
vectorizer = TfidfVectorizer(
    preprocessor=clean_text,
    ngram_range=(1, 2)
)

X = vectorizer.fit_transform(corpus)


df = pd.DataFrame(
    data=vectorizer.transform(corpus).toarray(),
    columns=vectorizer.get_feature_names()
)

display(df)
```

1.2 Exemple de classification

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

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

Téléchargement d'une partie des topics

In [ ]:

```
categories = ['alt.atheism', 'talk.religion.misc',
              'rec.sport.hockey','comp.graphics', 'sci.space']

news = fetch_20newsgroups(subset='all',
                          categories=categories)

print ("Taille du jeu de données\n")
print (news.filenames.shape)

print ("Un exemple de données\n")
print (news.data[5], '\n TOPIC : ',news.target[5],
       '\n********************')


```

Nettoyage des données

In [ ]:

```

def clean_news (data):
    for i in range(len(data)):
        #print (i)
        data[i]=clean_text(data[i])
    return data


```

In [ ]:

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

Vectorisation

In [ ]:

```
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(news.data)


```

X et y

In [ ]:

```
#Specification des variables à prédire et de la classe X et y
X = vectors.toarray()

y = news.target

```

Création du jeu d'apprentissage et de test

In [ ]:

```
from sklearn.model_selection import train_test_split

validation_size=0.3 #30% du jeu de données pour le test

testsize= 1-validation_size
seed=30
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)


```

Utilisation du classifieur

In [ ]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from time import time

seed=7
k_fold = KFold(n_splits=10, shuffle=True, random_state=seed)
clf = GaussianNB()

scoring = 'accuracy'
t0 = time()
score = cross_val_score(clf, X, y, cv=k_fold, scoring=scoring)
print("Réalisé en %0.3fs" % (time() - t0))

print('Les différentes accuracy pour les 10 évaluations sont : \n',
      score,'\n')
print ('Accuracy moyenne : ',score.mean(),
       ' standard deviation', score.std())
```

Mise en place d'un pipeline

In [ ]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.metrics import accuracy_score, confusion_matrix
from time import time
from sklearn.metrics import classification_report

pipeline = Pipeline([('vect', TfidfVectorizer()),
                ('clf', SGDClassifier(loss='hinge',
                                      penalty='l2',
                                      alpha=1e-3,
                                      random_state=42,
                                      max_iter=5, tol=None)),
                ])




X=news.data
y=news.target


X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)


t0 = time()
pipeline.fit(X_train, y_train)
print("Fit réalisé en %0.3fs" % (time() - t0))

t0 = time()
result = pipeline.predict(X_test)
print("Prédiction réalisée en %0.3fs" % (time() - t0))

print('\n accuracy:',accuracy_score(result, y_test),'\n')




conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)



print ('\n',classification_report(y_test, result))

```

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 [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from time import time
from sklearn.svm import SVC
import pickle


# Specification des pipelines
# programmation à optimiser par une fonction :)
pipeline_SGDC = Pipeline([('tfidf', TfidfVectorizer()),
                    ('clf', SGDClassifier())])


parameters_SGDC = [
    {'clf__max_iter': (5,),
    'clf__alpha': (0.00001, 0.000001),
    'clf__penalty': ('l2', 'elasticnet')}
]

pipeline_DT = Pipeline([('tfidf', TfidfVectorizer()),
                   ('clf', DecisionTreeClassifier())])


#param_range = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
param_range = [1, 5, 8, 10]
parameters_DT = [
    {'clf__min_samples_leaf': param_range,
        'clf__criterion': ['gini', 'entropy'],
        'clf__max_depth': param_range,
        'clf__min_samples_split': param_range[1:]}
]




X=news.data
y=news.target


X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)

# Creation des GridSearchCV avec les pipelines spécifiques

gs_SGDC = GridSearchCV(pipeline_SGDC,
                       parameters_SGDC,
                       cv=3,
                       n_jobs=-1,
                       scoring='accuracy')


gs_DT = GridSearchCV(pipeline_DT,
```

```
                      parameters_DT,
                      cv=3,
                      n_jobs=-1,
                      scoring='accuracy')



grids = [gs_SGDC, gs_DT]
grid_dict={0:'Linear classifiers', 1:'Decision Tree'}

best_acc = 0.0
best_clf = 0.0
best_gs = ''

for idx,gs in enumerate(grids):
    print('\nClassifier: %s' % grid_dict[idx])
    t0 = time()
    gs.fit(X_train, y_train)
    print("Fit réalisé en %0.3fs" % (time() - t0))

    print('Meilleurs paramètres : %s' % gs.best_params_)

    print("Meilleur score d'accuracy sur l'entrainement: %.3f" % gs.best_score
    # Prediction sur le jeu de test avec les meilleurs paramètres
    t0 = time()
    result = gs.predict(X_test)
    print("Prédiction réalisée en %0.3fs" % (time() - t0))

    print("Score d'accuracy pour les meilleurs paramètres sur jeu de test : %

    print ('\n matrice de confusion \n',confusion_matrix(y_test, result))

    print ('\n',classification_report(y_test, result))

    #Modele avec la meilleure accuracy sur le jeu de test
    if accuracy_score(y_test, result) > best_acc:
        best_acc = accuracy_score(y_test, result)
        best_gs = gs
        best_clf = idx



print('\nClassifier avec la meilleur accuracy sur le jeu de test\n',
      grid_dict[best_clf])


```

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 donnés récupéreront dont la matrice du TfidfVectorizer et les pré-traitements associés).

In [ ]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.metrics import accuracy_score, confusion_matrix
from time import time
from sklearn.metrics import classification_report


#Recupération des données pour l'exemple
#et partir proprement
categories = ['alt.atheism', 'talk.religion.misc',
              'rec.sport.hockey','comp.graphics', 'sci.space']

news = fetch_20newsgroups(subset='all',
                          categories=categories)









pipeline = Pipeline([('vect', TfidfVectorizer(preprocessor=clean_text)),
                ('clf', SGDClassifier(loss='hinge',
                                      penalty='l2',
                                      alpha=1e-05,
                                      random_state=42,
                                      max_iter=5,
                                      tol=None)),
                ])




X=news.data
y=news.target


X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               train_size=validation_size,
                                               random_state=seed,
                                               test_size=testsize)



t0 = time()
print ("Lancement du fit \n")
pipeline.fit(X_train, y_train)
print("Fit réalisé en %0.3fs" % (time() - t0))

t0 = time()
print ("Lancement de la prédiction \n")
result = pipeline.predict(X_test)
print("Prédiction réalisée en %0.3fs" % (time() - t0))

print('\n accuracy:',accuracy_score(result, y_test),'\n')

conf = confusion_matrix(y_test, result)
print ('\n matrice de confusion \n',conf)


```

```
60
61  print ('\n',classification_report(y_test, result))
62
63  print("\nSauvegarde du pipeline grid search")
64  filename = 'thebestone1.pkl'
65  pickle.dump(pipeline, open(filename, 'wb'))
66
67
```

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 [ ]:

```
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 séléctionnées\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])
```