MDI343txtmining

December 10, 2017

1 TP MDI343: Application à la classification : l'analyse d'opinions

1.1 Implémentation du classifieur

In [22]:
```
import os.path as op
import numpy as np
import pandas as pd
from math import log
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from nltk import pos_tag
from sklearn.svm import LinearSVC
from glob import glob
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from nltk import SnowballStemmer
import nltk
import re
```

In [2]:
```
# Chargement des textes de critiques
print("Loading dataset")

from glob import glob
filenames_neg = sorted(glob(op.join('.', 'data', 'imdb1', 'neg', '*.txt')))
filenames_pos = sorted(glob(op.join('.', 'data', 'imdb1', 'pos', '*.txt')))

texts_neg = [open(f).read() for f in filenames_neg]
texts_pos = [open(f).read() for f in filenames_pos]
texts = texts_neg + texts_pos
y = np.ones(len(texts), dtype=np.int)
y[:len(texts_neg)] = 0.


print("%d documents" % len(texts))
```

```
Loading dataset
2000 documents
```

1. Compléter la fonction count_words qui va compter le nombre d'occurrences de chaque mot dans une liste de string et renvoyer le vocabulaire.

In [3]:
```
#Méthode countword:

        def count_words(texts):
            """Vectorize text : return count of each word in the text snippets

            Parameters
            ----------
            texts : list of str
                The texts

            Returns
            -------
            vocabulary : dict
                A dictionary that points to an index in counts for each word.
            counts : ndarray, shape (n_samples, n_features)
                The counts of each word in each text.
                n_samples == number of documents.
                n_features == number of words in vocabulary.
            """
            words = set()
            vocabulary = {}
            list_voc1 = []

            for text in texts:
                voc1 = {}
                for word in text.split(" "):
                    if word not in vocabulary :
                        vocabulary[word] = 1
                        if word not in voc1:
                            voc1[word] = 1
                        else:
                            voc1[word] += 1
                    else:
                        vocabulary[word] += 1
                        if word not in voc1:
                            voc1[word] = 1
                        else:
                            voc1[word] += 1

                list_voc1.append(voc1)
                #pass
```

```
        counts = pd.DataFrame(list_voc1).fillna(0)

        return vocabulary, counts, list_voc1
```

```
In [4]: countword = count_words(texts)
```

La méthode retourne * le vocabulaire présent dans les textes en format dictionnaire * Une table avec le nombre d'occurence des mots du vocabulary dans chaque textes en format Dataframe, avec en colonne les mots et en ligne les textes. (le format permet ainsi de faire de la recherche spécifique sur un texte). * La liste des vocabulaires dans chaque texte

2) Expliquer comment les classes positives et négatives ont été assignées sur les critiques de films (voir fichier poldata.README.2.0) a set of ad-hoc rule:

- numerical ratings and star ratings. ("8/10", "four out of five", and "OUT OF ****: ***" are examples of rating indications we recognize.)
- five star/four star/letter grade notation stystem

3) Compléter la classe NB pour qu'elle implémente le classifieur Naive Bayes en vous appuyant sur le pseudo-code de la figure 1 et sa documentation ci-dessous :

```
In [5]: class NB(BaseEstimator, ClassifierMixin):

        #def __init__(self,Vocabulary,countDocs):
            #self.Voc = Vocabulary
            #self.countDoc = countDocs


        def fit(self,X, y):
            N=len(X)
            A = X.copy()
            A['class'] = y
            B = A.groupby(['class']).sum()

            self.prob_pos = (B.iloc[0]+1)/(B.iloc[0]+1).sum()
            self.prob_neg = (B.iloc[1]+1)/(B.iloc[1]+1).sum()
            self.prior_neg = sum(y)/N
            self.prior_pos = (N - sum(y))/N
            return (self)


        def predict(self,X):
            self.classpred = np.zeros(len(X))
            scorec_pos = np.zeros(len(X))
            scorec_neg = np.zeros(len(X))
            for i in range (len(X)):
                scorec_pos[i] = log(self.prior_pos)
                scorec_neg[i] = log(self.prior_neg)
```

```
                for word in X.columns[X.iloc[i,:] !=0]:
                    if word in self.prob_pos.index:
                        scorec_pos[i] += log(self.prob_pos[word])
                        scorec_neg[i] += log(self.prob_neg[word])
                    else:
                        pass
                self.classpred[i] = np.array(np.argmax([scorec_pos[i],scorec_neg[i]]))
            return (self.classpred)


        def score(self, X, y):
            return np.mean(self.predict(X) == y)
```

Implémentation de la classe qui permet de retourner en plus de la prédiction sur un texte une mesure de la confiance en la pédiction avec les valeures de prob_pos et neg_pos, permettant d'avoir un pseudo intervalle de confiance.'

```
In [6]: y =list(np.repeat(1,len(texts_neg))) + list(np.repeat(0,len(texts_pos)))
        X = countword[1]

        Xtrain = X[::2]
        ytrain = y[::2]

        Xtest = X[1::2]
        ytest = y[1::2]

        NaivBay = NB()
        NaivBay.fit(Xtrain,ytrain)
        print('Le score de prédiction est : ' ,NaivBay.score(Xtest,ytest))
```

```
Le score de prédiction est :  0.827
```

4. Evaluer les performances de votre classifieur en cross-validation 5-folds.

```
In [8]: print('Les scores de CV sont : ',cross_val_score(NaivBay, X.drop('fit',axis=1), y))
```

```
Les scores de CV sont :  [ 0.80389222  0.81081081  0.83033033]
```

5. Modifiez la fonction count_words pour qu'elle ignore les "stop words" dans le fichier data/english.stop. Les performances sont-elles améliorées?

```
In [13]: # On charge les stops words english fournis.
         stop = open('./data/english.stop').read().split()

         # On rajoute à cette liste la ponctuation.
         ponctuation = ['.','"',")","("," ",";",'\'','  ','!',':','/']
         stop.extend(ponctuation)
```

In [14]:
```
#Méthode countwords en retirant les stops words.

def count_words_stop(texts,stopwords):
    """Vectorize text : return count of each word in the text snippets

    Parameters
    ----------
    texts : list of str
        The texts

    Returns
    -------
    vocabulary : dict
        A dictionary that points to an index in counts for each word.
    counts : ndarray, shape (n_samples, n_features)
        The counts of each word in each text.
        n_samples == number of documents.
        n_features == number of words in vocabulary.
    """
    words = set()
    vocabulary = {}
    list_voc1 = []

    for text in texts:
        voc1 = {}
        for word in text.split():
            if word not in stopwords:
                if word not in vocabulary :
                    vocabulary[word] = 1
                    if word not in voc1:
                        voc1[word] = 1
                    else:
                        voc1[word] += 1

                else:
                    vocabulary[word] += 1
                    if word not in voc1:
                        voc1[word] = 1
                    else:
                        voc1[word] += 1
            else:
                pass
        list_voc1.append(voc1)
        #pass
    counts = pd.DataFrame(list_voc1).fillna(0)

    return vocabulary, counts, list_voc1
```

```
In [15]: wordcount_stop = count_words_stop(texts,stop)
         X = wordcount_stop[1]
         NaivBay_stop = NB()
         print('Les scores de CV sont : ', cross_val_score(NaivBay_stop, X.drop('fit',axis=1), y
```

```
Les scores de CV sont :  [ 0.80538922  0.82282282  0.82582583]
```

Le meilleur score de CV n'est pas amélioré en enlevant les stops words.'

1.2 Utilisation de scikitlearn

Question 1 : Comparer votre implémentation avec scikitlearn. On utilisera la classe CountVectorizer et un Pipeline :

```
In [16]: CV = CountVectorizer(stop_words = stop)
         MNB = MultinomialNB()
         pipeline = Pipeline([('CV', CV), ('MNB', MNB)])

         parameters = {'CV__analyzer' : ['char', 'word','char_wb'],'CV__ngram_range':[(1,1),(2,2

         GSCV = GridSearchCV(pipeline, parameters,cv=3, return_train_score=False)


         GSCV.fit(texts,y)
         results = pd.DataFrame(GSCV.cv_results_)

         print('Meilleur score :  ', GSCV.best_score_)
         print('Paramètres retenus :  ', GSCV.best_params_)
```

```
Meilleur score :   0.791
Paramètres retenus :   {'CV__analyzer': 'word', 'CV__ngram_range': (1, 1)}
```

Question 2 : Tester un autre algorithme de la librairie scikitlearn (ex : LinearSVC, LogisticRegression).

```
In [17]: CV = CountVectorizer(stop_words = stop)
         LSVC = LinearSVC()
         pipeline = Pipeline([('CV', CV), ('LSVC', LSVC)])

         parameters = {'CV__analyzer' : ['char', 'word','char_wb'],'CV__ngram_range':[(1,1),(1,2

         GSCV = GridSearchCV(pipeline, parameters,cv=3, return_train_score=False)


         GSCV.fit(texts,y)
         results = pd.DataFrame(GSCV.cv_results_)
```

```
print('Meilleur score :  ', GSCV.best_score_)
print('Paramètres retenus :  ', GSCV.best_params_)
```

```
Meilleur score :   0.8185
Paramètres retenus :   {'CV__analyzer': 'word', 'CV__ngram_range': (1, 2)}
```

Question 3 : Utiliser la librairie NLTK afin de procéder à une racinisation (stemming). Vous utiliserez la classe SnowballStemmer.

```
In [19]: stemmer = SnowballStemmer("english")
         texts_stemmed =[" ".join(stemmer.stem(word) for word in text.split()) for text in texts

In [20]: CV = CountVectorizer(stop_words = stop)
         LSVC = LinearSVC()
         pipeline = Pipeline([('CV', CV), ('LSVC', LSVC)])

         parameters = {'CV__analyzer' : ['char', 'word','char_wb'],'CV__ngram_range':[(1,1),(1,2

         GSCV = GridSearchCV(pipeline, parameters,cv=3, return_train_score=False)


         GSCV.fit(texts_stemmed,y)
         results = pd.DataFrame(GSCV.cv_results_)

         print('Meilleur score :  ', GSCV.best_score_)
         print('Paramètres retenus :  ', GSCV.best_params_)
```

```
Meilleur score :   0.835
Paramètres retenus :   {'CV__analyzer': 'word', 'CV__ngram_range': (1, 2)}
```

Question 4 : Filtrer les mots par catégorie grammaticale (POS : Part Of Speech) et ne garder que les noms, les verbes, les adverbes et les adjectifs pour la classification.

```
In [23]: pattern = re.compile(".VB.|NN|JJ|.RB")
         texts_pos = [" ".join(tuple[0] if pattern.match(tuple[1]) else '' \
                                  for tuple in nltk.pos_tag(text.split())) for text in texts]

In [24]: CV = CountVectorizer(stop_words = stop)
         LSVC = LinearSVC()
         pipeline = Pipeline([('CV', CV), ('LSVC', LSVC)])

         parameters = {'CV__analyzer' : ['word','char_wb'],'CV__ngram_range':[(1,2)]}

         GSCV = GridSearchCV(pipeline, parameters,cv=3, return_train_score=False)


         GSCV.fit(texts_pos,y)
```

```
        results = pd.DataFrame(GSCV.cv_results_)

        print('Meilleur score :  ', GSCV.best_score_)
        print('Paramètres retenus :  ', GSCV.best_params_)
```

```
Meilleur score :   0.803
Paramètres retenus :   {'CV__analyzer': 'word', 'CV__ngram_range': (1, 2)}
```

In [ ]:
```
Conclusion: le meilleur résultat est l'algorithme avec un stemmer en pré traitement, qui
```

In [ ]:
```
!jupyter nbconvert --to pdf MDI343txtmining.ipynb
```

In [ ]: