```
# Authors: Alexandre Gramfort
#          Chloe Clavel
# License: BSD Style.
# TP Cours ML Telecom ParisTech MDI343
import os
import os.path as op
import numpy as np
import nltk
import re
import operator

from glob import glob

from nltk import word_tokenize
from nltk import pos_tag
from nltk  import SnowballStemmer

from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression


nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
stemmer = SnowballStemmer("english")

cwd = os.getcwd()
os.chdir("data")
```

```
[nltk_data] Downloading package punkt to /Users/famat/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/famat/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
###############################################################################
# Load data
def loadData(Verbose=False):
    if(Verbose):
        print("Loading dataset")

    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.
    if(Verbose):
        print("%d documents" % len(texts))
    return texts, y

texts, y = loadData()
```

```
################################################################################
# Count_words

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 = {}

    for text in texts:
        for word in re.findall(r'\w+', text):
            if word not in words.keys():
                words[word] = 1
            words[word] += 1
    #print(words.keys())

    ## creation de la table de correspondance
    conv = words.keys()
    dictConv = {}
    counter = 0
    for el in conv :
        dictConv[el] = counter
        counter += 1

    n_features = len(words.keys())
    n_samples = len(texts)

    counts = np.zeros((n_samples,n_features))
    for i in range(len(texts)):
        tabTmp = re.findall(r'\w+', texts[i])
        for j in range(len(tabTmp)):
            counts[i][dictConv[tabTmp[j]]] += 1

    vocabulary = dictConv
    return vocabulary, counts
```

```
# Count words in text
vocabulary, X = count_words(texts)
```

```
class NB(BaseEstimator, ClassifierMixin):

    def __init__(self):
        """
        texts_neg, texts_pos, y  = loadData()
        texts =  texts_neg + texts_pos
        vocabulary, counts = count_words(texts)
        Classes = [0, 1]
        prior = {}
        condprob = None
        """

        pass

    def fit(self, X, y):
        return self

    def predict(self, X):
        return (np.random.randn(len(X)) > 0).astype(np.int)

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

```
# Try to fit, predict and score
nb = NB()
nb.fit(X[::2], y[::2])
print(nb.score(X[1::2], y[1::2]))
```

```
0.486
```

Explain how positive and negative classes have been assigned to movie reviews

The positive and negative classes have been assigned from scraping webpages. Only the explicit rating have been kept.
The positive and negative classes are determined from the rating system that the website have used.
If the score of the movie is above 80%, 3.5/5, 3/4 or B or plus , the review is considered as positive.
If the score of the movie is below 2/5, 1.5/4 or c- or below, the review is considered as negative.

```
def loadDataWithClass(Verbose=False):
    if(Verbose):
        print("Loading dataset")

    filenames_neg = sorted(glob(op.join('..', 'data', 'imdb1', 'neg', '*.txt')))
    filenames_pos = sorted(glob(op.join('..', 'data', 'imdb1', 'pos', '*.txt')))
    #class_neg = 0
    #class_pos = 1
    texts_neg = [[open(f).read(),0] for f in filenames_neg]
    texts_pos = [[open(f).read(),1] for f in filenames_pos]
    if(Verbose):
        print(len(texts_neg),"longueur text neg")
        print(len(texts_pos),"longueur text pos")

    texts = texts_neg + texts_pos

    y = np.ones(len(texts), dtype=np.int)
    y[:len(texts_neg)] = 0.
    if(Verbose):
        print("%d documents" % len(texts))
    return  texts, y
```

```
texts_withClass , y = loadDataWithClass()
```

```
class NB(BaseEstimator, ClassifierMixin):

    def __init__(self):
        C = None
        V = None
        CondProb = None
        prior = None
    def fit(self, X, y):
        texts = []
        EnsembleDesClasses = [0,1]
        for i in range(len(X)):
            texts.append((X[i],y[i]))
        #for i in range(len(y)):
        #    if y[i] not in EnsembleDesClasses:
        #        EnsembleDesClasses.append(y[i])
        return self.trainMulti(texts, EnsembleDesClasses)

    def predict(self, X):
        return self.applyMulti(X)

    def score(self, X, y):
        score = 0
        for i in range(len(X)):
            if(self.predict(X[i]) ==  y[i]):
                score += 1
        return score / len(X)

    def applyMulti(self,d):

        C, V, prior, condProb = self.C, self.V, self.prior, self.condProb
        W = self.ExtractTokensFromDoc(V,d)
        score = {}
        for classe in C :
            score[classe] = np.log(prior[classe])
            for t in W :
                score[classe] += np.log(condProb[t][classe])
        return max(score.items(), key=operator.itemgetter(1))[0]

    def trainMulti(self,texts,EnsembleDesClasses):
        C = EnsembleDesClasses
        D = texts
        V, X = count_words([t[0] for t in texts])
        N = self.CountDocs(D)
        prior = {}
        terme_freq = {}
        condProb = {}
        for classe in C:
            N_C = self.CountDocsInClass(D,classe)
            prior[classe] = (N_C * 1.0) / N
            text_c = self.ConcatenateTextOfAllDocsInClass(D,classe)
            SUM_T_C = 0
            for t in V :
                if t  not in terme_freq.keys():
                    terme_freq[t] = {}
                terme_freq[t][classe] = self.CountTokensOfTerm(text_c, t)
                SUM_T_C += terme_freq[t][classe] + 1
            for t in V :
                if t not in condProb.keys():
                    condProb[t] = {}
                condProb[t][classe] = 1.0*(terme_freq[t][classe] + 1) / SUM_T_C
        self.C, self.V, self.prior, self.condProb = C, V, prior, condProb

        return C, V, prior, condProb
    def CountDocs(self,D):
        return len(D)

    def CountDocsInClass(self,D,classe):
        counter = 0
        for i in range(len(D)):
            if(D[i][1] == classe):
                counter += 1
        return counter

    def ConcatenateTextOfAllDocsInClass(self,D,classe):
        textsToReturn = []
        for i in range(len(D)):
            if(D[i][1] == classe):
                textsToReturn.append(D[i][0])
        return textsToReturn

    def CountTokensOfTerm(self,text,t):
        count = 0
        newText = [t[0] for t in text]
        for text in newText:
            listOfWord = re.findall(r'\w+', text)
            for el in listOfWord:
                if el == t:
                    count += 1
        return count

    def ExtractTokensFromDoc(self,V,d):
        tokens = []
        listOfWord = re.findall(r'\w+', d)
        for el in listOfWord:
            if el in V:
                tokens.append(el)
        return tokens
```

```
#nb_with_all_word = NB()
#EnsembleDesClasses = [0,1]
#text_neg = [t[0] for t in texts_withClass if t[1] == 0 ]
#C, V, prior, condProb = nb_with_all_word.trainMulti(texts_withClass,EnsembleDesCla
#C, V, prior, condProb = nb_with_all_word.trainMulti(text_neg,EnsembleDesClasses)
```

```
texts_data = [t[0] for t in texts_withClass]
texts_target = [t[1] for t in texts_withClass]
```

```
nb_cross_val = NB()
```

```
scores_NB_all_text = cross_val_score(nb_cross_val, texts_data , texts_target, cv=5)
print(max(scores_NB_all_text))
```

```
0.52
```

```
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 = {}
    stop_word = []
    with open('english.stop', 'r') as f:
        for line in f :
            stop_word.append(line.rstrip('\n'))

    for text in texts:
        for word in re.findall(r'\w+', text):
            if word not in stop_word :
                if word not in words.keys():
                    words[word] = 1
                words[word] += 1

    #print(words.keys())

    ## creation de la table de correspondance
    conv = words.keys()
    dictConv = {}
    counter = 0
    for el in conv :
        dictConv[el] = counter
        counter += 1

    n_features = len(words.keys())
    n_samples = len(texts)

    counts = np.zeros((n_samples,n_features))
    for i in range(len(texts)):
        tabTmp = re.findall(r'\w+', texts[i])
        for j in range(len(tabTmp)):
            if tabTmp[j] not in stop_word:
                counts[i][dictConv[tabTmp[j]]] += 1

    vocabulary = dictConv
    return vocabulary, counts
```

```
class NB(BaseEstimator, ClassifierMixin):

    def __init__(self):
        C = None
        V = None
        CondProb = None
        prior = None

    def fit(self, X, y):
        texts = []
        EnsembleDesClasses = [0,1]
        for i in range(len(X)):
            texts.append((X[i],y[i]))
        #for i in range(len(y)):
        #    if y[i] not in EnsembleDesClasses:
        #        EnsembleDesClasses.append(y[i])
        return self.trainMulti(texts, EnsembleDesClasses)

    def predict(self, X):
        return self.applyMulti(X)

    def score(self, X, y):
        score = 0
        for i in range(len(X)):
            if(self.predict(X[i]) ==  y[i]):
                score += 1
        return score / len(X)

    def applyMulti(self,d):

        C, V, prior, condProb = self.C, self.V, self.prior, self.condProb
        W = self.ExtractTokensFromDoc(V,d)
        #print("prior",prior)
        score = {}
        for classe in C :
            score[classe] = np.log(prior[classe])
            for t in W :
                score[classe] += np.log(condProb[t][classe])
        #print( "neg:",score[0],"pos:", score[1])
        return max(score.items(), key=operator.itemgetter(1))[0]

    def trainMulti(self,texts,EnsembleDesClasses):
        C = EnsembleDesClasses
        D = texts
        V, X = count_words([t[0] for t in texts])
        N = self.CountDocs(D)
        prior = {}
        terme_freq = {}
        condProb = {}

        for classe in C:
            #print(classe)
            N_C = self.CountDocsInClass(D,classe)
            prior[classe] = (N_C * 1.0) / N
            text_c = self.ConcatenateTextOfAllDocsInClass(D,classe)
            SUM_T_C = 0
            for t in V :
                if t  not in terme_freq.keys():
                    terme_freq[t] = {}
                terme_freq[t][classe] = self.CountTokensOfTerm(text_c, t)
                SUM_T_C += terme_freq[t][classe] + 1
            for t in V :
                if t not in condProb.keys():
                    condProb[t] = {}
                condProb[t][classe] = 1.0*(terme_freq[t][classe] + 1) / SUM_T_C

        self.C, self.V, self.prior, self.condProb = C, V, prior, condProb

        return C, V, prior, condProb

    def CountDocs(self,D):
        return len(D)

    def CountDocsInClass(self,D,classe):
        counter = 0
        for i in range(len(D)):
            if(D[i][1] == classe):
                counter += 1
        return counter

    def ConcatenateTextOfAllDocsInClass(self,D,classe):
        textsToReturn = []
        for i in range(len(D)):
            if(D[i][1] == classe):
                textsToReturn.append(D[i][0])
        return textsToReturn

    def CountTokensOfTerm(self,text,t):
        count = 0
        newText = [t[0] for t in text]
        for text in newText:
            listOfWord = re.findall(r'\w+', text)
            for el in listOfWord:
                if el == t:
                    count += 1
        return count

    def ExtractTokensFromDoc(self,V,d):
        tokens = []
        listOfWord = re.findall(r'\w+', d)
        for el in listOfWord:
            if el in V:
                tokens.append(el)
        return tokens
```

```
texts_data = [t[0] for t in texts_withClass]
texts_target = [t[1] for t in texts_withClass]
nb = NB()
scores_with_prepro = cross_val_score(nb, texts_data , texts_target, cv=5)
```

```
print("score sans les stop words :" ,max(scores_with_prepro))
print("score avec les stop words:" ,max(scores_NB_all_text))
```

```
score sans les stop words : 0.5275
score avec les stop words: 0.52
```

```
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('clf', MultinomialNB()),
])
```

```
text_SVC = Pipeline([('vect', CountVectorizer()),
                     ('SVC', LinearSVC()),
])
```

```
text_LR = Pipeline([('vect', CountVectorizer()),
                    ('LR', LogisticRegression()),
])
```

```
def stemmerDIY(texts):
    textsSTEMMED = []
    for text in texts:
        textTMP = ""
        for word in re.findall(r'\w+',text):
            textTMP += " " + stemmer.stem(word)
        textsSTEMMED.append(textTMP)
    return textsSTEMMED
```

```
texts_data_stemmed = stemmerDIY(texts_data)
```

```
texts_data_treated = []
for text in texts_data:
    tokenized = word_tokenize(text)
    tagged = pos_tag(tokenized)
    texttmp = ""
    for word, tag in tagged:
        if tag in ('NN', 'VB', 'ADJ','ADV'):
            texttmp += " "
            texttmp += word

    texts_data_treated.append(texttmp)
```

```
texts_data_stemmed_and_pos = stemmerDIY(texts_data_treated)
```

```
scores_sklearn_SVC = cross_val_score(text_SVC, texts_data , texts_target, cv=5)
scores_sklearn_LR = cross_val_score(text_LR , texts_data , texts_target, cv=5)
scores_sklearn = cross_val_score(text_clf, texts_data , texts_target, cv=5)
```

```
scores_sklearn_LR_STEM = cross_val_score(text_LR , texts_data_stemmed , texts_targe
scores_sklearn_SVC_STEM = cross_val_score(text_SVC, texts_data_stemmed , texts_targ
scores_sklearn_STEM = cross_val_score(text_clf, texts_data_stemmed , texts_target,
```

```
scores_sklearn_LR_POS = cross_val_score(text_LR , texts_data_treated , texts_target
scores_sklearn_SVC_POS = cross_val_score(text_SVC, texts_data_treated , texts_targe
scores_sklearn_POS = cross_val_score(text_clf, texts_data_treated , texts_target, c
```

```
scores_sklearn_LR_ALL = cross_val_score(text_LR , texts_data_stemmed_and_pos , text
scores_sklearn_SVC_ALL = cross_val_score(text_SVC, texts_data_stemmed_and_pos , tex
scores_sklearn_ALL = cross_val_score(text_clf, texts_data_stemmed_and_pos , texts_t
```

```
print("scores Avec Texte non traité : \n ")
print("Logistic R \t SVC \t NB")
print(max(scores_sklearn_LR), "\t \t", max(scores_sklearn_SVC),max(scores_sklearn))
```

```
scores Avec Texte non traité :

Logistic R      SVC     NB
0.8675          0.85 0.825
```

```
print("scores Avec STEM: \n ")
print("Logistic R \t SVC \t NB")
print(max(scores_sklearn_LR_STEM),"\t \t", max(scores_sklearn_SVC_STEM),max(scores_
```

```
scores Avec STEM:

Logistic R      SVC     NB
0.8625          0.85 0.8325
```

```
print("scores Avec POS TAGGING : \n ")
print("Logistic R \t SVC \t NB")
print(max(scores_sklearn_LR_POS),"\t \t", max(scores_sklearn_SVC_POS),max(scores_sk
```

```
scores Avec POS TAGGING :

Logistic R      SVC     NB
0.805          0.7975 0.7825
```

```
print("scores Avec POS TAGGING et STEM : \n ")
print("Logistic R \t SVC \t NB")
print(max(scores_sklearn_LR_ALL),"\t \t", max(scores_sklearn_SVC_ALL),max(scores_sk
```

```
scores Avec POS TAGGING et STEM :
```

```
Logistic R      SVC     NB
0.8          0.7925 0.7775
```