

File: 40sentimentanalysis.py

```
import os.path as op
import numpy as np

from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import cross_val_score

#####
# Load data
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')))
filenames_neg = sorted(glob(op.join('imdb1', 'neg', '*.txt')))
filenames_pos = sorted(glob(op.join('imdb1', 'pos', '*.txt')))
#filenames_neg = ["ciao.txt", "help.txt", "ciao1.txt"]
#filenames_pos = ["ciao2.txt", "ciao3.txt", "ciao4.txt"]

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

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

#####
# Start part to fill in

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.
    """
    # Create the set of stopwords
    stopwords_set = set()
    for word in stopwords.split("\n"):
        stopwords_set.add(word)
    # Create the set of words
```

```

words = set()
for text in texts:
    # Read a string from text
    for word in text.split(" "):
        # Add to set if not stopword
        if word not in stopwords_set:
            words.add(word)
# Create vocabulary <word, index in counts>
vocabulary = {}
i = 0
for word in words:
    vocabulary[word] = i
    i += 1
# Create the counts matrix - a line for a document, a value is a frequency of term
n_features = len(words)
counts = np.zeros((len(texts), n_features))
k = 0 # index of document
for text in texts:
    for word in text.split(" "):
        if word not in stopwords_set:
            counts[k][vocabulary[word]] += 1 # term frequency
    k += 1
return vocabulary, counts

```

```

class NB(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def fit(self, X, y):
        # V <- extractVocabulary(texts) = (vocabulary) = X
        # N <- countDocs(texts) = len(texts)
        self.N = X.shape[0]
        # C <- classes = unique values of y
        self.C = np.unique(y)
        # count docs in class = count 0/1 in y
        categories_counts = np.bincount(y)
        self.prior = {}
        self.tct = {}
        for c in self.C:
            self.tct[c] = np.zeros(X.shape[1])
            # compute a priori probability
            self.prior[c] = categories_counts[int(c)]/self.N
            # compute freq of words in class
            for i in range(len(y)):
                if y[i] == c:
                    self.tct[c] += X[i]
        # Laplacian smoothing - updating tct
        self.condprob = {}
        for c in self.C:
            self.condprob[c] = np.zeros(X.shape[1])
        for c in self.C:
            den = np.sum(self.tct[c])+len(self.tct[c])
            for t in range(len(self.tct[c])):
                self.condprob[c][t] = (self.tct[c][t]+1)/den
        return self

```

```

def predict(self, X):
    score = np.zeros((self.N, len(self.C)))
    for c in self.C:
        score[:,int(c)] = np.log(self.prior[c])
    for x in range(X.shape[0]):
        for c in self.C:
            for t in range(X.shape[1]):
                if X[x][t] != 0:
                    score[x,int(c)] += np.log(self.condprob[c][t])
    result = [np.argmax(score[x]) for x in range(X.shape[0])]
    return result

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

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

# Try to fit, predict and score
nb = NB()
nb.fit(X[:,2], y[:,2])
# print (nb.score(X[1::2], y[1::2]))
scores = cross_val_score(nb, X[1::2], y[1::2], cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

```