Sentiment analysis

I. Introduction

This project approaches an active research area in Natural Language Processing (NLP), sentiment analysis. Given the exponentially growing of online review data (Amazon, IMDB, etc…), sentiment analysis becomes increasingly important. We are going to build a sentiment classifier, i.e., evaluating a piece of text being either positive or negative.
This assignment requires to gather the raw data, do preprocessing, design suitable ML algorithms and implement the solution.
The "Large Movie Review Dataset"(1) is used for this project.
The dataset is compiled from a collection of 50,000 reviews from IMDB on the condition there are no more than 30 reviews per movie.
Number of positive and negative reviews are equal.
Negative reviews have scores lesser or equal 4 out of 10 while a positive review greater or equal 7 out of 10. Neutral reviews are not included.
Then, 50,000 reviews are divided evenly into the training and test sets

We are then going to train a Stochastic Gradient Descent (SGD) Classifier. While gradient descend is powerful, it can be prohibitively expensive when the dataset is extremely large because every single data point needs to be processed.
However, it turns out that, when the data is large, rather than the entire dataset, SGD algorithm performs just as good with a small random subset of the original data. This is the central idea of Stochastic SGD and particularly handy for the text data since corpus are often humongous.

(1) *https://ai.stanford.edu/ amaas/data/sentiment/*

II. Methodology

1. Data Preprocessing

First the raw database used to train our classifier is combined into a single .csv file which has three columns, *row_number*, *text* and *polarity*. The column *tex* contains review texts from the aclImdb database while the column *polarity* consists of sentiment labels, *1* for positive and *0* for negative.

Similarly the test database used to test our prediction model is a single csv file with two columns: *row_number* and *text*. The column *polarity* is excluded and the objective is to use the trained SGD classifier to predict this information.

In addition, common English stopwords are removed as well as any numbers and punctuation found.

Furthermore, words in the test database that do not appear in the training database are removed in order to avoid any mismatched features.

2. Data Representations

The very first step in solving any NLP problem is finding a way to represent the text data so that machines can understand. A common approach is to use a document-term vector where each document is encoded as a discrete vector that counts occurrences of each word in the vocabulary it contains. This data representation is also called a Unigram model.

For example, consider two one-sentence documents:
d1: "I love this Artificial Intelligence course."
d2: "Artificial Intelligence is awesome"

The vocabulary {artificial, awesome, this, course, I, intelligence, is, love} in the two documents can be encoded as v1 and v2 as follow:

	artificial	awesome	this	course	I	intelligence	is	love
v1	1	0	1	1	1	1	0	1
v2	1	1	0	0	0	1	1	0

Table 1: Unigram Data Representation

A more sophisticated data representation model is the Bigram model where occurrences depend on a sequence of two words rather than an individual one. Using the same example as before, v1 and v2 are now encoded as seen in Table 2:

	artificial | artificial	artificial | awesome	...	intelligence | artificial	...	love | love
v1	0	0	...	1	...	0
v2	0	1	...	0	...	0

Table 2: Bigram Data Representation

Sometimes, a very high word counting may not be meaningful. For example, a common word like "say" may appear 10 times more than a less-common word such as "machine" but it does not mean "say" is 10 times more relevant to our sentiment classifier. To alleviate this issue, we can instead use the term frequency *tf[t]*:

$$tf[t]=1+log(f[t,d])$$

where *f[t,d]* is the count of term *t* in document *d*. The log function dampens the unwanted influence of common English words.

Inverse document frequency (idf) is a similar concept. For example, Computer Science documents often have words such as computers, CPU, programming, appearing over and over. While they are not common English words, because of the document domain, their occurrences are very high. To rectify this, we can adjust using inverse term frequency $idf[t] = log(N / df[t])$ where *df[t]* is the number of documents containing the term *t* and *N* is the total number of document in the dataset.

Therefore, instead of just using a word frequency, a frequency *tf_idf* for each term *t* can be used:

$$tf\_idf[t] = tf[t] * idf[t]$$

3. Training and testing of the classifier

Using the sklearn library in Python, the following steps are done for each data representation model:

- Step 1: Each text document is converted into one of the token-count row of a term-document matrix [number of samples x terms frequencies] .
- Step 2: A fitting of the SGD classifier is done by the minimization of a linear SVM objective function with a l1 penalty term (to bring sparsity to the model). Because SGD only sums on a random portion of the dataset, it is way less computationally extensive than GD even if more updates are necessary before convergence.
- Step 3: The SGD classifier can then be used to predict new polarity labels from test data.

III. **Results**

Four different data representation models were used and a SGD classifier was trained then used to predict sentiment polarity of new test data (whose polarities were in fact known). Table 3 presents a comparison between the prediction efficiency rates obtained for each representation. One observes that the best results are obtained with a model using a Unigram idf model, but that efficiencies between representations only differ by at most 2%. The advantage of using one type of representation model over another is thus not demonstrated here.

	Unigram	Bigram	Unigram idf	Bigram idf
prediction score	85.20%	85.75%	87.26%	86.50%

Table 3: Comparison of predictions between 4 different data representation models

IV. Python Code

```
from __future__ import division
import sys
from sklearn import linear_model
from sklearn import datasets
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
import os
import os.path
import glob
import fileinput
import fnmatch
import time
import re
import gc

# train_path = "../resource/lib/publicdata/aclImdb/train/" # use terminal to ls i
# test_path = "../resource/lib/publicdata/imdb_te.csv" # test data for grade eval

begin = time.clock()
train_path = "C:/MOOC/Edx-MOOC/AI micromaster/AI/Projects/Project5/aclImdb/train'
test_path = "C:/MOOC/Edx-MOOC/AI micromaster/AI/Projects/Project5/imdb_te.csv"

stopwords = set()
for line in open("stopwords.en.txt", 'r'):
    stopwords.add(line[:-1])

stopwords2 = {'\'s', 'n\'t', '\'m', '\'re', '\' ', '/', '\'ll',
         '1', '2','3', '4', '5', '6', '7', '8', '9', '0', 'I', 'My', 'You', 'They
          'The', 'There', 'That', 'This ', 'Those', 'These',
         'A', 'An', 'It', 'Or', ' ', ''}


## Use nltk library for NLP
# from nltk.corpus import stopwords
# # ...
# filtered_words = [word for word in word_list if word not in stopwords.words('en




def imdb_data_preprocess(inpath, outpath="./", name="imdb_tr.csv", mix=False):
    """Implement this module to extract
```

```python
    and combine text files under train_path directory into
    imdb_tr.csv. Each text file in train_path should be stored
    as a row in imdb_tr.csv. And imdb_tr.csv should have two
    columns, "text" and label"""

    posi = os.listdir(inpath + '/pos')
    nega = os.listdir(inpath + '/neg')

    csv_file = pd.DataFrame(columns=['text', 'polarity'])

    print(len(nega))

    for i in range(len(posi)):  # len(posi)
        file = open(inpath + '/pos/' + posi[i])
        text = file.read()
        text1 = re.split('\W+', text)
        text2 = text1[:]
        for word in text1:
            if word in stopwords:
                text2.remove(word)
            if word in stopwords2:
                text2.remove(word)
        text = ""
        for word in text2:
            text += word + ' '
        csv_file.loc[i] = [text, 1]

    for i in range(len(nega)):  # len(nega)
        file = open(inpath + '/neg/' + nega[i])
        text = file.read()
        text1 = re.split('\W+', text)
        text2 = text1[:]
        for word in text1:
            if word in stopwords:
                text2.remove(word)
            if word in stopwords2:
                text2.remove(word)
        text = ""
        for word in text2:
            text += word + ' '
        csv_file.loc[i + len(posi)] = [text, 0]

    csv_file.to_csv(outpath + name, sep=',')

def prepare_data(train_data):
    # prepare train data
    fi_tr = train_data.iloc[:, 1]
    pola = train_data.iloc[:, 2]
    db0_tr = fi_tr.values
    pola0_tr = pola.values
    print('pola_tr:%s' % pola0_tr.shape)
    print('db_tr:%s' % db0_tr.shape)
    pola0_tr = pola0_tr.astype(int)
```

```
    print('type_pola: %s'%type(pola0_tr[2]))
    print('type_db_tr: %s' % type(db0_tr[2]))
    # split train data for cross_validation
    db_tr, X_test, pola_tr, y_test = train_test_split(db0_tr, pola0_tr, test_size
    print(len(db_tr), len(X_test))
    # prepare test data
    test_data = pd.read_csv("imdb_te.csv")
    fi_te = test_data.iloc[:, 1]
    db_te = fi_te.values
    db_te2 = []
    for line in db_te:  # len(nega)
        text1 = re.split('\W+', line)
        text = ""
        text2 = text1[:]
        for word in text1:
            if word in stopwords:
                text2.remove(word)
            if word in stopwords2:
                text2.remove(word)

        for word in text2:
            text += word + ' '
        db_te2.append(text)

    return db_tr, db_te2, X_test, pola_tr, y_test

## Unigram

def unigram(db_tr, db_te2, X_test, pola_tr, y_test):
    vect = CountVectorizer()
    vect.fit(db_tr)
    Xtr = vect.transform(db_tr)
    print('Xtr:(%s, %s)'%(Xtr.shape))
    Xte = vect.transform(db_te2)
    print('Xte:(%s, %s)'%(Xte.shape))
    clf = SGDClassifier(loss='hinge', penalty='l1')
    clf.fit(Xtr, pola_tr)
    pred1 = clf.predict(Xte)
    X_test1 = vect.transform(X_test)
    print('score Unigram, test data: %s' % clf.score(Xte, pred1))
    print('cross validation score Unigram, test data: %s' % cross_val_score(clf,
    print('score Unigram: %s'%clf.score(X_test1, y_test))
    print('cross validation score Unigram: %s'%cross_val_score(clf, X_test1, y_te
    # print(accuracy_score(y_test, pred1))

    out = open('unigram.output.txt','w')
    for i in range(len(pred1)):
        out.write('%s\n'%int(pred1[i]))
    out.close()

    return Xtr, Xte, X_test1

## bigram
```

```
def bigram(db_tr, db_te2, X_test, pola_tr, y_test):
    bigram_vect = CountVectorizer(ngram_range=(1, 2),token_pattern=r'\b\w+\b', mi
    bigram_vect.fit(db_tr)
    Xtr2 = bigram_vect.transform(db_tr)
    Xte2 = bigram_vect.transform(db_te2)

    print('Xtr2:(%s, %s)'%(Xtr2.shape))
    print('Xte2:(%s, %s)'%(Xte2.shape))

    clf = SGDClassifier(loss='hinge', penalty='l1')
    clf.fit(Xtr2, pola_tr)
    pred2 = clf.predict(Xte2)

    X_test2 = bigram_vect.transform(X_test)
    print('score Unigram, test data: %s' % clf.score(Xte2, pred2))
    print('cross validation score Unigram, test data: %s' % cross_val_score(clf,
    print('score Bigram: %s'%clf.score(X_test2, y_test))
    print('cross validation score Bigram: %s'%cross_val_score(clf, X_test2, y_tes
    # print(accuracy_score(y_test, pred2))

    out = open('bigram.output.txt','w')
    for i in range(len(pred2)):
        out.write('%s\n'%int(pred2[i]))
    out.close()

    return Xtr2, Xte2, X_test2

## Tf-idf

def unigram_tdidf(Xtr, Xte, X_test1, pola_tr, y_test):
    transformer = TfidfTransformer()
    Xtr_tfidf = transformer.fit_transform(Xtr)
    Xte_tfidf = transformer.transform(Xte)

    print('Xtr_tfidf:(%s, %s)'%(Xtr_tfidf.shape))
    print('Xte_tfidf:(%s, %s)'%(Xte_tfidf.shape))

    clf = SGDClassifier(loss='hinge', penalty='l1')
    clf.fit(Xtr_tfidf, pola_tr)
    pred3 = clf.predict(Xte_tfidf)

    X_test3 = transformer.transform(X_test1)
    print('score Unigram, test data: %s' % clf.score(Xte_tfidf, pred3))
    print('cross validation score Unigram, test data: %s' % cross_val_score(clf,
    print('score Unigram Tdidf: %s'%clf.score(X_test3, y_test))
    print('cross validation score Unigram Tdidf: %s'%cross_val_score(clf, X_test3
    # print(accuracy_score(y_test, pred3))

    out = open('unigramtfidf.output.txt','w')
    for i in range(len(pred3)):
        out.write('%s\n'%int(pred3[i]))
    out.close()
```

```
    return Xtr_tfidf, Xte_tfidf, X_test3


## bigram Tf-idf

def bigram_tdidf(db_tr, db_te2, X_test, pola_tr, y_test):
    tf_vect = TfidfVectorizer(ngram_range=(1, 2),token_pattern=r'\b\w+\b', min_d
    Xtr2_tfidf = tf_vect.fit_transform(db_tr)
    Xte2_tfidf = tf_vect.transform(db_te2)

    print('Xtr2_tfidf:(%s, %s)'%(Xtr2_tfidf.shape))
    print('Xte2_tfidf:(%s, %s)'%(Xte2_tfidf.shape))

    clf = SGDClassifier(loss='hinge', penalty='l1')
    clf.fit(Xtr2_tfidf, pola_tr)
    pred4 = clf.predict(Xte2_tfidf)

    X_test4 = tf_vect.transform(X_test)
    print('score Unigram, test data: %s' % clf.score(Xte2_tfidf, pred4))
    print('cross validation score Unigram, test data: %s' % cross_val_score(clf,
    print('score Bigram Tdidf: %s'%clf.score(X_test4, y_test))
    print('cross validation score Bigram Tdidf: %s'%cross_val_score(clf, X_test4,
    # print(accuracy_score(Xte2_tfidf, pred4))

    out = open('bigramtfidf.output.txt','w')
    for i in range(len(pred4)):
        out.write('%s\n'%int(pred4[i]))
    out.close()

    return Xtr2_tfidf, Xte2_tfidf, X_test4

if __name__ == "__main__":
    # """train a SGD classifier using unigram representation,
    # predict sentiments on imdb_te.csv, and write output to
    # unigram.output.txt
    # train a SGD classifier using bigram representation,
    # predict sentiments on imdb_te.csv, and write output to
    # bigram.output.txt
    # train a SGD classifier using unigram representation
    # with tf-idf, predict sentiments on imdb_te.csv, and write
    # output to unigramtfidf.output.txt
    # train a SGD classifier using bigram representation
    # with tf-idf, predict sentiments on imdb_te.csv, and write
    # output to bigramtfidf.output.txt"""

    imdb_data_preprocess(train_path)

    train_data = pd.read_csv("imdb_tr.csv")

    db_tr, db_te2, X_test, pola_tr, y_test = prepare_data(train_data)

    Xtr, Xte, X_test1 = unigram(db_tr, db_te2, X_test, pola_tr, y_test)
```

```python

    Xtr2, Xte2, X_test2 = bigram(db_tr, db_te2, X_test, pola_tr, y_test)

    Xtr_tfidf, Xte_tfidf, X_test3 = unigram_tdidf(Xtr, Xte, X_test1, pola_tr, y_t

    Xtr2_tfidf, Xte2_tfidf, X_test4 = bigram_tdidf(db_tr, db_te2, X_test, pola_tr

    end = time.clock()
    diff = end - begin
    print('time: %ss' % diff)
```