SEEM 5680 Text Mining Models and Applications
Text Classification Demo

Python Implementation for:

1. Document pre-processing

2. Feature selection

3. Text classification
    - model training and prediction
    - performance evaluation

Environment setting

Language:	Python https://www.python.org/downloads/
IDE:	VS Code (suitable for all systems) https://code.visualstudio.com/
Package:	nltk, sklearn command line> pip install nltk command line> pip install sklearn (when using nltk, some extra files maybe downloaded via following the given instruction)
Data:	Twitter text classification (Not racist/sexist vs. Racist/sexist)

The data and configuration document for the environment have been uploaded.

VS Code Interface

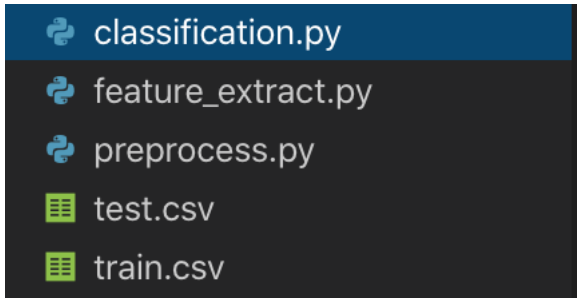
Right: editing area

train.csv | test.csv | preprocess.py | feature_extract.py | classification.py

classification.py > ...

```
42    ##### 1. data preprocess #####
43    x_train, y_train, x_test, y_test = preprocess() ## raw text
44
45    ##### 2. feature extraction #####
46    x = get_feature_binary(x_train + x_test)
47    # x = get_feature_tfidf(x_train + x_test)
48    # x = get_feature_binary_mutualinformation(x_train + x_test, y_train +
49    x_train = x[:len(x_train)] ## feature vectors
50    x_test = x[len(x_train):] ## feature vectors
51
52    ##### 3. train the classifier and predict on test data #####
53    y_predict = classify_NaiveBayesian(x_train, y_train, x_test)
54    # y_predict = classify_SVM(x_train, y_train, x_test)
55    # y_predict = classify_Logistic(x_train, y_train, x_test)
56
57    ##### 4. evaluate the learned model and output result #####
58    evaluation(y_predict, y_test)
```

Left: explorer to show the files in the opened folder



Bottom: click Terminal → New Terminal to test the code. (python xxx.py)

Data format

The data in train.csv and test.csv is in the following format.

Three columns represent:
id: an unique integer for each tweet
label: binary value, 0 means that it is not racist or sexist
tweet: the raw text of the tweet

id,label,tweet
1,0, @user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
2,0,@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked

……

14,1,@user #cnn calls #michigan middle school 'build the wall' chant '' #tcot
15,1,no comment! in #australia #opkillingbay #seashepherd #helpcovedolphins #thecove #helpcovedolphins
16,0,ouch...junior is angryð got7 #junior #yugyoem #omg

File preprocess.py

preprocess()

Obtain preprocessed text data from the dataset file

```
import csv
from nltk import word_tokenize
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
import scapy

def preprocess():
        path_train = 'train.csv'
        path_test = 'test.csv'
        # obtain raw text from given files
        x_train, y_train = get_data_from_file(path_train)
        x_test, y_test = get_data_from_file(path_test)

        # text preprocess
        x_train = text_process(x_train)
        x_test = text_process(x_test)

        return x_train, y_train, x_test, y_test
```

Two functions are used in preprocess:

- get_data_from_text(path)
  Obtain raw text of each sample from the dataset file

- text_process(x)
  Conduct text preprocess for the given raw text data

File preprocess.py

```
# to read x and y data from the csv file
def get_data_from_file(path_file):
        with open(path_file, 'r', encoding='utf-8') as file_input:
                csv_reader = csv.reader(file_input, delimiter = ',')
                next(csv_reader) # skip the header of csv file
                x_list = []
                y_list = []
                for row in csv_reader:
                        x_list.append(row[2])
                        y_list.append(row[1])
        return x_list, y_list
```

```
# text preprocessing using nltk package
def text_process(x_list):
        porter_stemmer = PorterStemmer()
        x_list_new = []
        for x in x_list:
                ####### Lower Case #######
                x = x.lower()
                ####### Tokenize #######
                x_token_list = word_tokenize(x)
                ####### Remove Stopwords #######
                x_token_list = [token for token in x_token_list if token not in
                stopwords.words('english')]
                ####### Stemming #######
                x_token_list = [porter_stemmer.stem(token) for token in x_token_list]

                x_new = ' '.join(x_token_list)
                x_list_new.append(x_new)
        return x_list_new
```

File feature_extract.py

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.feature_selection import mutual_info_classif
from preprocess import preprocess
from numpy import argmax, argsort, array

# get binary feature vectors from text
def get_feature_binary(sample_list):
        vectorizer = CountVectorizer()
        # data matrix
        data = vectorizer.fit_transform(sample_list).todense()
        data[data > 0] = 1
        # print(data)
        return data

# get tf-idf feature vectors from text
def get_feature_tfidf(sample_list):
        vectorizer = CountVectorizer()
        data_count = vectorizer.fit_transform(sample_list)

        transformer = TfidfTransformer()
        tfidf = transformer.fit_transform(data_count).todense()
        return tfidf
```

Two feature extraction methods are defined:

- get_feature_binary(sample_list)
  Obtain the binary vectors for the text sample list

- get_feature_tfidf(sample_list)
  Obtain the tf-idf vectors for the text sample list

File feature_extract.py

- get_feature_binary_mutualinformation(sample_list)
  A binary feature extraction method utilize the mutual information to select informative features

```
# get binary feature vectors from text, and feature selection with top 50 mutual information
def get_feature_binary_mutualinformation(sample_list, y_list):
        vectorizer = CountVectorizer()
        data = vectorizer.fit_transform(sample_list).todense()
        data[data > 0] = 1

        mutual_info = mutual_info_classif(data, y_list)
        index_mututal_info = argsort(-mutual_info)
        data = array(data)[:, index_mututal_info[:50]]
        print(data)

        return data
```

File classification.py

```
from preprocess import preprocess
from feature_extract import get_feature_binary, get_feature_binary_mutualinformation, get_feature_tfidf
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

# training and prediction with Naive Bayesian classifier
def classify_NaiveBayesian(x, y, x_test):
        # train
        nbayes = MultinomialNB()
        nbayes.fit(x, y)
        # test
        y_predict = nbayes.predict(x_test)
        return y_predict

# training and prediction with Logistic Regression classifier
def classify_Logistic(x, y, x_test):
        lr = LogisticRegression()
        lr.fit(x, y)
        y_predict = lr.predict(x_test)
        return y_predict

# training and prediction with SVM classifier
def classify_SVM(x, y, x_test):
        svm = SVC(kernel='linear')
        svm.fit(x, y)
        y_predict = svm.predict(x_test)
        return y_predict
```

Three model training and predicting methods are defined.

The input contains x and y from training set and x from test set.

- classify_NaiveBayesian(x, y, x_test)
- classify_Logistic(x, y, x_test)
- classify_SVM(x, y, x_test)

File classification.py

```
from sklearn.metrics import classification_report

# evaluation based on the predicted label and ground truth label
def evaluation(y_pred, y_test):
        print(classification_report(y_pred,y_test,target_names= ['not racist/sexist ', 'racist/sexist '],digits=3))
return 0
```

evaluation(p_predict, p_truth_test)

the classification_report function can output several classic metrics for the classification task.

Besides, one can explore the sklearn.metrics for other evaluation methods for classification.

File classification.py The test script

```
##### 1. data preprocess #####
x_train, y_train, x_test, y_test = preprocess() ## raw text

##### 2. feature extraction #####
x = get_feature_binary(x_train + x_test)
# x = get_feature_tfidf(x_train + x_test)
# x = get_feature_binary_mutualinformation(x_train + x_test, y_train + y_test)
x_train = x[:len(x_train)] ## feature vectors
x_test = x[len(x_train):] ## feature vectors

##### 3. train the classifier and predict on test data #####
y_predict = classify_NaiveBayesian(x_train, y_train, x_test)
# y_predict = classify_SVM(x_train, y_train, x_test)
# y_predict = classify_Logistic(x_train, y_train, x_test)

##### 4. evaluate the learned model and output result #####
evaluation(y_predict, y_test)
```

The script is written in the classification.py file, but you can also put the test script in a separated python file or even a folder for testing.

Result of test script

	precision	recall	f1-score	support
not racist/sexist	0.939	0.970	0.954	461
racist/sexist	0.417	0.256	0.317	39
accuracy			0.914	500
macro avg	0.678	0.613	0.636	500
weighted avg	0.898	0.914	0.904	500

You can use different feature extraction methods and different classifier to solve this classification problem.