Problem2-code

April 23, 2020

[48]:

```
import os
from nltk.stem.snowball import SnowballStemmer
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import heapq
import pandas as pd


dir = './Problem2-data/'
stop_words_dir = dir + 'stop-words.txt'
data_dir = dir + 'dataset/files/'
query_dir = dir + 'dataset/queries/'

d = 25    # preserve the first k eigenvectors
top = 3   # find the top 3 similiarity


def get_stop_words():
    stop_words = []
    stop_words_file = open(stop_words_dir)
    file_content = stop_words_file.read()
    stop_words = file_content.split(',')
    return stop_words


def remove_stop_words(stop_words, text):
    text = text.split(' ')
    new_words = []
    for word in text:
        word = word.strip()
        if word not in stop_words:
            new_words.append(word)

    new_text = ' '.join(new_words)
    return new_text


def stemmerize_text(text):
    stemmer = SnowballStemmer("english", ignore_stopwords=True)

    text = text.split(' ')
    for i in range(len(text)):
        word = text[i]
        word = word.strip()
        stem_word = stemmer.stem(word)
        text[i] = stem_word

    new_text = ' '.join(text)

    return new_text


def text_preprocess(text):
    text = text.strip()
    text = text.replace('\n', ' ')

    # to lower case
    text = text.lower()

    punc_free = []
    for ch in text:
        if ch not in string.punctuation:
            punc_free.append(ch)
        else:
            punc_free.append(' ')
    new_text = ''.join(punc_free)

    # replace number
    # text = text.translate(text.maketrans('', '', string.digits))

    return new_text


def load_data():
    texts = []
    num_files = len([f for f in os.listdir(data_dir)if os.path.isfile(os.path.
 ↪join(data_dir, f))])
    print('Find ' + str(num_files) + ' files.')
    for i in range(num_files):
        filepath = data_dir + 'file-' + str(i+1) + '.txt'
        texts.append(open(filepath).read())
    return texts


def get_clean_text(texts):
    '''
        texts: a list of strings
    '''

    stop_words = get_stop_words()

    for i in range(len(texts)):
        # preprocess data
        texts[i] = text_preprocess(texts[i])

        # remove stop words
        texts[i] = remove_stop_words(stop_words, texts[i])

        # stemmerize
        texts[i] = stemmerize_text(texts[i])

    return texts


def load_query():

    queries = []
    num_queries = len([f for f in os.listdir(query_dir)if os.path.isfile(os.path.
 ↪join(query_dir, f))])
    print('Find ' + str(num_queries) + ' queries.')
    for i in range(num_queries):
        filepath = query_dir + 'query-' + str(i+1) + '.txt'
        queries.append(open(filepath).read())

    return queries


def map_query(S, u, v, query_v):
    S = np.linalg.inv(S)
    query_m = []
    for q in query_v:
        q_m = np.dot(np.dot(q, u), S)
        query_m.append(q_m)
    return query_m


def cosine(q, d):
    '''
        consine similiarity
    '''
    # normalize the vectors
    denominator = np.linalg.norm(q) * np.linalg.norm(d)
    numerator = np.sum(np.multiply(q, d))
    return numerator / denominator


def svd_decomposition(matrix):

    u, s, v = np.linalg.svd(matrix)
    # u (3842, 3842)
    # s (35,)
    # v (35, 35)
    u = u[:, :d]

    S = np.zeros((d, d))
    eigenvalues = 0
    for i in range(d):
        S[i][i] = s[i]
        eigenvalues += s[i]

    print('preverse first ' + str(d) + ' eigenvectors')
    print('preserve ' + str(np.round(eigenvalues/np.sum(s)*100)) + '%' + '␣
 ↪information')

    v = v[:d, :]

    return S, u, v


def query(text, queries, matrix_type='counter'):

    vectorizer = None
    if matrix_type == 'counter':
        vectorizer = CountVectorizer()
    elif matrix_type == 'tf-idf':
        vectorizer = TfidfVectorizer()

    # TD Matrix
    X = vectorizer.fit_transform(text)
    TD_matrix = np.array(X.toarray()).T

    # list of words
    dictionary = vectorizer.get_feature_names()
    # vectorize queries
    query_v = [vectorizer.transform([q]).toarray() for q in queries]

    # SVD
    S, u, v = svd_decomposition(TD_matrix)

    # map query to new feature space
    query_m = map_query(S, u, v, query_v)

    resultSet = []
    for q in query_m:
        similarity = []
        for d in v.T:
            similarity.append(cosine(q, d))
        topk_index = list(map(similarity.index, heapq.nlargest(top, similarity)))
        topk_index = [index + 1 for index in topk_index]
        resultSet.append(topk_index)

    return TD_matrix, dictionary, resultSet


def main():
    global d
    texts = load_data()
    clean_text = get_clean_text(texts)

    queries = load_query()
    queries = get_clean_text(queries)
    print('queries: ')
    print(queries)

    # Count
    TD_matrix, dictionary, resultSet = query(clean_text, queries,␣
 ↪matrix_type='counter')
    print('Query by Matrix A1:')
    print(resultSet)

    # print the matrix A1
    TD = pd.DataFrame(data=TD_matrix, index=dictionary, columns=range(1,␣
 ↪len(texts)+1))
    TD.to_csv('TD-Matrix.csv')


    # TF-IDF

    d = 27
    TD_matrix, dictionary, resultSet = query(clean_text, queries,␣
↪matrix_type='tf-idf')
    print('Query by Matrix TF-IDF: ')
    print(resultSet)

    TD_TFIDF = pd.DataFrame(data=TD_matrix, index=dictionary, columns=range(1,␣
↪len(texts)+1))
    TD_TFIDF.to_csv('TFIDF-Matrix.csv')

    return None


main()
```

```
Find 35 files.
Find 6 queries.
queries:
['music play compos', 'ancient univers island', 'patient friend', 'loan pay
money', 'work wealth fun', 'cold rain fog']
preverse first 25 eigenvectors
preserve 82.0% information
Query by Matrix A1:
[[27, 25, 26], [20, 2, 4], [12, 15, 6], [23, 14, 19], [1, 7, 27], [33, 7, 28]]
preverse first 27 eigenvectors
preserve 80.0% information
Query by Matrix TF-IDF:
[[27, 25, 26], [20, 35, 2], [15, 12, 6], [23, 3, 14], [32, 27, 7], [34, 31, 33]]
```