Nama : Raihan Rahmanda Junianto

NIM : 222112303

Kelas : 3SD2

**Responsi Praktikum 5 Information Retrieval**

1. **Buat fungsi main untuk menampilkan 3 list dokumen yang terurut berdasarkan cosine similarity pada folder “berita” dengan query “vaksin corona jakarta”.**

Berdasarkan soal di atas, dibangunlah kode program sebagai berikut.

# import library yang dibutuhkan

import os

import re

import math

from spacy.lang.id import Indonesian

from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

from spacy.lang.id.stop\_words import STOP\_WORDS

# path berisi lokasi file-file berita

path = "D:/RAIHAN STIS/Perkuliahan/SEMESTER 5/Praktikum INFORMATION RETRIEVAL/Pertemuan (2)/berita"

nlp = Indonesian()

berita = []

# cleaning file berita

for file\_name in sorted(os.listdir(path)):

    file\_path = os.path.join(path, file\_name)

    with open(file\_path, 'r') as f:

        clean\_txt = re.sub("http\S+", ' ', f.read())

        clean\_txt = re.sub("[^\w\s0-9]|['\d+']|[\'\",.!?:;<>()\[\]{}@#$%^&\*=\_+/\\\\|~-]]|(\'\')", ' ', clean\_txt)

        clean\_txt = re.sub("[\n\n]", ' ', clean\_txt)

        clean\_txt = re.sub(r'\s+', ' ', clean\_txt).strip()

        berita.append(clean\_txt)

# membuat dictionary yang berisi nomor dokumen dan isinya

doc\_dict = {}

for i in range(1, len(berita) + 1):

    words = berita[i - 1].split()

    filtered\_words = [word for word in words if word.lower() not in STOP\_WORDS]

    stemming = StemmerFactory().create\_stemmer()

    stemmed\_words = [stemming.stem(word) for word in filtered\_words]

    doc\_dict[i] = " ".join(stemmed\_words)

# membuat inverted index

token\_arrays = []

for doc in berita:

    text\_low = doc.lower()

    nlp\_doc = nlp(text\_low)

    token\_doc = [token.text for token in nlp\_doc]

    token\_stpwords\_tugas = [w for w in token\_doc if w not in STOP\_WORDS]

    token\_arrays.append(token\_stpwords\_tugas)

inverted\_index = {}

for i in range(len(token\_arrays)):

    for item in token\_arrays[i]:

        item = stemming.stem(item)

        if item not in inverted\_index:

            inverted\_index[item] = []

        if (item in inverted\_index) and ((i+1) not in inverted\_index[item]):

            inverted\_index[item].append(i+1)

vocab = list(inverted\_index.keys())

def termFrequencyInDoc(vocab, doc\_dict):

    tf\_docs = {}

    for doc\_id in doc\_dict.keys():

        tf\_docs[doc\_id] = {}

    for word in vocab:

        for doc\_id,doc in doc\_dict.items():

            tf\_docs[doc\_id][word] = doc.count(word)

    return tf\_docs

def tokenisasi(text):

    tokens = text.split(" ")

    return tokens

def wordDocFre(vocab, doc\_dict):

  df = {}

  for word in vocab:

    frq = 0

    for doc in doc\_dict.values():

      if word in tokenisasi(doc):

        frq = frq + 1

    df[word] = frq

  return df

import numpy as np

def inverseDocFre(vocab,doc\_fre,length):

  idf= {}

  for word in vocab:

    idf[word] = idf[word] = 1 + np.log((length + 1) / (doc\_fre[word]+1))

  return idf

# vektor space model

def tfidf(vocab,tf,idf\_scr,doc\_dict):

  tf\_idf\_scr = {}

  for doc\_id in doc\_dict.keys():

    tf\_idf\_scr[doc\_id] = {}

  for word in vocab:

    for doc\_id,doc in doc\_dict.items():

      tf\_idf\_scr[doc\_id][word] = tf[doc\_id][word] \* idf\_scr[word]

  return tf\_idf\_scr

tf\_idf = tfidf(vocab, termFrequencyInDoc(vocab, doc\_dict), inverseDocFre(vocab, wordDocFre(vocab, doc\_dict), len(doc\_dict)), doc\_dict)

# Term - Document Matrix

TD = np.zeros((len(vocab), len(doc\_dict)))

for word in vocab:

  for doc\_id,doc in tf\_idf.items():

    ind1 = vocab.index(word)

    ind2 = list(tf\_idf.keys()).index(doc\_id)

    TD[ind1][ind2] = tf\_idf[doc\_id][word]

print("Term - Document Matrix: ")

print(TD)

query = "vaksin corona jakarta"

def termFrequency(vocab, query):

    tf\_query = {}

    for word in vocab:

        tf\_query[word] = query.count(word)

    return tf\_query

tf\_query = termFrequency(vocab, query)

idf = inverseDocFre(vocab, wordDocFre(vocab, doc\_dict), len(doc\_dict))

# Term - Query Matrix

TQ = np.zeros((len(vocab), 1)) #hanya 1 query

for word in vocab:

    ind1 = vocab.index(word)

    TQ[ind1][0] = tf\_query[word]\*idf[word]

print("\nTerm - Query Matrix: ")

print(TQ)

def cosine\_sim(vec1, vec2):

    vec1 = list(vec1)

    vec2 = list(vec2)

    dot\_prod = 0

    for i, v in enumerate(vec1):

        dot\_prod += v \* vec2[i]

    mag\_1 = math.sqrt(sum([x\*\*2 for x in vec1]))

    mag\_2 = math.sqrt(sum([x\*\*2 for x in vec2]))

    return dot\_prod / (mag\_1 \* mag\_2)

print("\nSkor cosine similarity: ")

print("skor cosine similarity query dan berita1.txt: ", cosine\_sim(TQ[:, 0], TD[:, 0])) #query & berita1

print("skor cosine similarity query dan berita2.txt: ", cosine\_sim(TQ[:, 0], TD[:, 1])) #query & berita2

print("skor cosine similarity query dan berita3.txt: ", cosine\_sim(TQ[:, 0], TD[:, 2])) #query & berita3

print("skor cosine similarity query dan berita4.txt: ", cosine\_sim(TQ[:, 0], TD[:, 3])) #query & berita4

print("skor cosine similarity query dan berita5.txt: ", cosine\_sim(TQ[:, 0], TD[:, 4])) #query & berita5

from collections import OrderedDict

def exact\_top\_k(doc\_dict, TD, q, k):

    relevance\_scores = {}

    i = 0

    for doc\_id in doc\_dict.keys():

        relevance\_scores[doc\_id] = cosine\_sim(q, TD[:, i])

        i = i + 1

    sorted\_value = OrderedDict(sorted(relevance\_scores.items(), key=lambda x: x[1], reverse = True))

    top = {j: sorted\_value[j] for j in list(sorted\_value)[:k]}

    return top

top\_3 = exact\_top\_k(doc\_dict, TD, TQ[:, 0], 3)

print("\nSkor top 5 berita yang paling relevan dengan query: ")

print(top\_3)

Pada program di atas proses pembentukan indeks terbalik serta matriks Term Document (TD) masih sama seperti praktikum sebelumnya. Pada praktikum kali ini, sebelum dokumen menjalani proses preprocessing, terdapat suatu prosedur untuk membersihkan teks dalam dokumen tersebut.

A computer screen shot of a code

Description automatically generated

Setelah kode di atas dijalankan, maka teks pada dokumen akan bersih dari ekspresi-ekspresi yang tidak diinginkan, seperti link, dan lain sebagainya. Langkah selanjutnya adalah mencari matriks Term Query (TQ) untuk dicari cosine similaritynya terhadap matriks TD.

Proses penghitungan matriks TQ adalah sebagai berikut.

A computer screen shot of a program code

Description automatically generated

Pada praktikum kali ini, query yang digunakan adalah “vaksin corona Jakarta”. Query tersebut akan dihitung frekuensi kemunculan kata-katanya menggunakkan fungsi “termFrequency(vocab, query)” di mana vocab dan query sudah didefinisikan sebelumnya. Output dari fungsi “termFrequency(vocab, query)” akan disimpan pada variabel tf\_query. Selanjutnya terdapat perhitungan IDF menggunakkan fungsi “inverseDocFre(vocab, wordDocFre(vocab, doc\_dict), len(doc\_dict))”.

Kemudian, matriks TQ digunakan untuk menyimpan nilai-nilai TF-IDF dari setiap kata dalam query. Matriks ini berukuran n × 1, di mana n menyatakan jumlah kata dalam vocab. Lalu, dilakukan iterasi pada vocab yang di dalamnya terdapat proses perkalian antara tf\_query dengan idf. Dengan demikian, setiap sel dalam matriks TQ akan berisi nilai TF-IDF yang sesuai untuk kata dalam query.

Proses selanjutnya adalah menghitung cosine similarity antara matriks TD dan TQ. Proses perhitungan cosine similarity masih sama seperti praktikum sebelumnya, untuk melihat kode secara keseluruhan dapat dilihat di bagian atas laporan ini. Namun, untuk mengetahui dokumen yang terurut berdasarkan skor cosine similarity dilakukan proses sebagai berikut.

A computer screen shot of colorful text

Description automatically generated

Fungsi “exact\_top” akan mengembalikan lima berita teratas yang paling relevan dengan query yang diberikan berdasarkan skor cosine similarity sehingga nilai k yang diberikan adalah 5. Fungsi ini menggunakkan moduOrderedDict dari pustaka collections yang digunakan untuk mempertahankan urutan elemen-elemen dalam kamus sesuai dengan nilai-nilai relevansinya.

Berikut merupakan output lengkap dari keseluruhan kode di atas.

A screenshot of a computer

Description automatically generated

A black screen with a black background

Description automatically generated

A black screen with a black background

Description automatically generated

A black screen with a black background

Description automatically generated

A screenshot of a computer screen

Description automatically generated

1. **Lakukan efisiensi dengan menggunakan index elimination sederhana.**

Berdasarkan soal di atas, dibangunlah kode program sebagai berikut.

# import library yang dibutuhkan

import os

import re

import math

from spacy.lang.id import Indonesian

from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

from spacy.lang.id.stop\_words import STOP\_WORDS

# path berisi lokasi file-file berita

path = "D:/RAIHAN STIS/Perkuliahan/SEMESTER 5/Praktikum INFORMATION RETRIEVAL/Pertemuan (2)/berita"

nlp = Indonesian()

berita = []

# cleaning file berita

for file\_name in sorted(os.listdir(path)):

    file\_path = os.path.join(path, file\_name)

    with open(file\_path, 'r') as f:

        clean\_txt = re.sub("http\S+", ' ', f.read())

        clean\_txt = re.sub("[^\w\s0-9]|['\d+']|[\'\",.!?:;<>()\[\]{}@#$%^&\*=\_+/\\\\|~-]]|(\'\')", ' ', clean\_txt)

        clean\_txt = re.sub("[\n\n]", ' ', clean\_txt)

        clean\_txt = re.sub(r'\s+', ' ', clean\_txt).strip()

        berita.append(clean\_txt)

# membuat dictionary yang berisi nomor dokumen dan isinya

doc\_dict = {}

for i in range(1, len(berita) + 1):

    words = berita[i - 1].split()

    # eliminasi stopwords

    filtered\_words = [word for word in words if word.lower() not in STOP\_WORDS]

    # stemming

    stemming = StemmerFactory().create\_stemmer()

    stemmed\_words = [stemming.stem(word) for word in filtered\_words]

    # karena sebelumnya masih dalam per kata disatukan kembali menjadi kalimat untuk variabel doc\_dict

    doc\_dict[i] = " ".join(stemmed\_words)

def tokenisasi(text):

    tokens = text.split(" ")

    return tokens

def stemming(text):

    from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

    # create stemmer

    factory = StemmerFactory()

    stemmer = factory.create\_stemmer()

    # stemming process

    output = stemmer.stem(text)

    return output

def index\_elim\_simple(query, doc\_dict):

    remove\_list =[]

    for doc\_id,doc in doc\_dict.items():

        n = 0

        for word in tokenisasi(query):

            if stemming(word) in doc:

                n = n+1

        if n==0:

            remove\_list.append(doc\_id)

    for key in remove\_list:

        del doc\_dict[key]

    return doc\_dict

query = "vaksin corona jakarta"

doc\_dict = index\_elim\_simple(query, doc\_dict)

print(doc\_dict)

Berdasarkan program di atas, operasi untuk melakukan eliminasi indeks menggunakkan fungsi “index\_elim\_simple(query, doc\_dict)” yang akan mengembalikan suatu kamus yang berisi semua dokumen yang mengandung term pada query “vaksin corona Jakarta”. Berikut disajikan output dari kode program tersebut. Note: 1 menunjukkan berita1.txt, 2 menunjukkan berita2.txt, dan seterusnya.

A black screen with white text

Description automatically generated