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.

```
# cleaning file berita
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)
```

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.

```
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)
```

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.

```
from collections import OrderedDict
def exact_top(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 = exact_top(doc_dict, TD, TQ[:, 0], 5)
print("\nSkor top 5 berita yang paling relevan dengan query: ")
print(top)
```

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.

```
(base) D:\RAIHAN STIS\Perkuliahan\SEMESTER 5\Praktikum INFORMATION RETRIEVAL\Pertemuan (5)>python penugasan5_1.py
Term - Document Matrix:
[[ 1.69314718  0.          0.          0.          1.69314718]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.          6.          4.          1.          3.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 1.40546511  0.          1.40546511  1.40546511  0.        ]
 [ 1.          1.          1.          4.          1.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 1.69314718  1.69314718  0.          0.          0.        ]
 [ 4.19722458  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 6.29583687  0.          0.          0.          0.        ]
 [ 6.29583687  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 1.69314718  1.69314718  0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 4.19722458  0.          0.          0.          0.        ]
 [ 7.02732554  4.21639532 11.24372086  9.83825576  5.62186043]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
 [ 2.09861229  0.          0.          0.          0.        ]
```

```
Term - Query Matrix:
[[0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [1.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [1.69314718]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]]
```

```
Skor cosine similarity:
skor cosine similarity query dan berita1.txt:  0.010832772328566935
skor cosine similarity query dan berita2.txt:  0.305441706917711
skor cosine similarity query dan berita3.txt:  0.30457740843687225
skor cosine similarity query dan berita4.txt:  0.07688776837468171
skor cosine similarity query dan berita5.txt:  0.051488176982188355

Skor top 3 berita yang paling relevan dengan query:
{2: 0.305441706917711, 3: 0.30457740843687225, 4: 0.07688776837468171}
```

2. **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.

```
(base) D:\RAIHAN STIS\Perkuliahan\SEMESTER 5\Praktikum INFORMATION RETRIEVAL\Pertemuan (5)>python penugasan5_2.py
{1: 'wilayah bebas covid cek kab kota zona hijau baru jakarta perintah rencana terap laku batas giat masyarakat ppkm level hitung des
ember januari menteri sehat ri bijak ppkm level tahap kaji direktur cegah kendali sakit tular langsung p pml kemenkes ri dr siti nadi
a tarmizi ppkm level terap covid signifikan picu tingkat mobilitas longgar protokol sehat', 2: 'vaksin covid rutin gantung jelas jaka
rta beri booster dosis tiga vaksin covid indonesia rencana januari lantas ada vaksinasi covid vaksinasi influenza ketua satgas covid
ikat dokter indonesia idi prof zubairi djoerban pasti kait turut vaksin covid booster vaksinasi covid', 3: 'ri suntik booster ampuh l
awan varian delta cs jakarta pakar aku vaksin vaksin dosis alami turun efektivitas varian corona varian delta booster dosis tiga vaks
in covid indonesia jenis vaksin ikut strain virus baru ketua satgas covid ikat dokter indonesia idi prof zubairi djoerban singgung ri
set kait efektivitas vaksin covid dosis sebut dasar riset efektivitas vaksin covid pfizer moderna bukti turun lawan varian delta', 4:
 'alert varian delta covid dki tingkat jakarta data baru balitbangkes kemenkes ri november tambah varian delta tambah jawa barat dki
jakarta sulawesi utara balitbangkes dki jakarta alami tingkat varian delta signifikan varian varian alpha varian delta beta indonesia
 asal dki jakarta total', 5: 'corona as dadak serang delta reda jakarta covid wilayah amerika serikat as covid catat stabil pasca ser
ang varian delta musim panas kepala nasihat medis gedung putih dr anthony fauci senin nasional turun persen minggu puncak gelombang v
arian delta musim panas pasien covid area barat timur laut dadak'}
```