Nama : Raihan Rahmanda Junianto

NIM : 222112303

Kelas : 3SD2

Praktikum 8 Information Retrieval

Permasalahan:

Buat fungsi main untuk menampilkan 3 list dokumen yang terurut berdasarkan BM25 pada folder “berita” dengan query “vaksin corona jakarta”. Bandingkan dengan hasil perankingan cosine similarity pada modul 5.

Solusi:

Berdasarkan permasalahan di atas, dirancang suatu kode program sebagai berikut.

```
# import library yang dibutuhkan
import os
import re
import math
import numpy as np

from spacy.lang.id import Indonesian
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
from spacy.lang.id.stop_words import STOP_WORDS
from collections import OrderedDict
from rank_bm25 import BM25Okapi

nlp = Indonesian()
stemming = StemmerFactory().create_stemmer()

def cleaning_file_berita(path):
    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)
    return berita
```

```python
# membuat dictionary yang berisi nomor dokumen dan isinya
def create_doct_dict(berita):
  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]
      stemmed_words = [stemming.stem(word) for word in filtered_words]
      doc_dict[i] = " ".join(stemmed_words)
  return doc_dict

# membuat inverted index
def create_inverted_index(berita):
  token_arrays = []
  inverted_index = {}

  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)

  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)
  return inverted_index

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

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

# Term - Document Matrix
def termDocumentMatrix(vocab, tf_idf, doc_dict):
  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]
  return TD

def termFrequency(vocab, query):
    tf_query = {}
    for word in vocab:
        tf_query[word] = query.count(word)
    return tf_query

# Term - Query Matrix
def termQueryMatrix(vocab, tf_query, idf):
    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]
    return 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)

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_k = {j: sorted_value[j] for j in list(sorted_value)[:k]}
    return top_k

def exact_top_k_bm25(doc_dict, rank_score, k):
    relevance_scores = {}
    i = 0
    for doc_id in doc_dict.keys():
        relevance_scores[doc_id] = rank_score[i]
        i = i + 1

    sorted_value = OrderedDict(sorted(relevance_scores.items(), key=lambda x:
x[1], reverse = True))
    top_k = {j: sorted_value[j] for j in list(sorted_value)[:k]}
    return top_k

def construct_bm25(query, doc_dict):
  tokenized_corpus = [tokenisasi(doc_dict[doc_id]) for doc_id in doc_dict]
  bm25 = BM25Okapi(tokenized_corpus)
  tokenized_query = tokenisasi(query)
  doc_scores = bm25.get_scores(tokenized_query)
  return doc_scores

def main():
  # path berisi lokasi file-file berita
  path = "D:/RAIHAN STIS/Perkuliahan/SEMESTER 5/Praktikum INFORMATION
RETRIEVAL/Pertemuan (2)/berita"

  berita = cleaning_file_berita(path)

  doc_dict = create_doct_dict(berita)
```

```
  inverted_index = create_inverted_index(berita)
  vocab = list(inverted_index.keys())
  tf_idf = tfidf(vocab, termFrequencyInDoc(vocab, doc_dict),
inverseDocFre(vocab, wordDocFre(vocab, doc_dict), len(doc_dict)), doc_dict)
  TD = termDocumentMatrix(vocab, tf_idf, doc_dict)

  query = "vaksin corona jakarta"
  idf = inverseDocFre(vocab, wordDocFre(vocab, doc_dict), len(doc_dict))
  tf_query = termFrequency(vocab, query)
  TQ = termQueryMatrix(vocab, tf_query, idf)

  top_3 = exact_top_k(doc_dict, TD, TQ[:, 0], 3)
  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan VSM
berbasis Cossine Similarity: ")
  print(top_3)


  doc_scores = construct_bm25(query, doc_dict)

  top_3_bm25 = exact_top_k_bm25(doc_dict, doc_scores, 3)
  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan
Rank Okapi BM25: ")
  print(top_3_bm25)

main()
```

Program di atas merupakan program untuk melakukan perankingan dari sekumpulan dokumen menggunakkan Information Retrieval Probabilistik, yaitu BM25. Metode perankingan ini menggunakkan menggunakan term frequency untuk meranking similarity, lalu meranking dokumen berdasarkan probabilitas. Dalam menghitung term frequency, document frequency, idf, pre-processing, dan lain sebagainya masih menggunakkan sintaks/cara yang sama dengan praktikum sebelumnya. Selanjutnya juga akan dilakukan perbandingan output antara ranking menggunakkan Vector Space Model dan BM25.

Guna melakukan perangkingan probabilistic, program ini menggunakkan satu library tambahan, yaitu BM25Okapi. Library ini digunakan untuk melakukan pemeringkatan dokumen menggunakkan kaidah probabilistik, yaitu BM25.

```
# import library yang dibutuhkan
import os
import re
import math
import numpy as np

from spacy.lang.id import Indonesian
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
from spacy.lang.id.stop_words import STOP_WORDS
from collections import OrderedDict
from rank_bm25 import BM25Okapi
```

Selanjutnya, terdapat fungsi "construct_bm25(query, doc_dict)" yang digunakkan untuk model BM25 untuk menghitung skor relevansi antara sebuah kueri dan setiap dokumen dalam kamus dokumen. Oleh karena itu, fungsi ini memiliki dua parameter, yaitu doct_dict dan query. Fungsi ini mengembalikan nilai skor relevansi antara kueri dan dokumen.

```
def construct_bm25(query, doc_dict):
  tokenized_corpus = [tokenisasi(doc_dict[doc_id]) for doc_id in doc_dict]
  bm25 = BM25Okapi(tokenized_corpus)
  tokenized_query = tokenisasi(query)
  doc_scores = bm25.get_scores(tokenized_query)
  return doc_scores
```

Kemudian, terdapat fungsi "exact_top_k_bm25(doct_dict, rank_score, k)" yang digunakkan untuk mengambil tiga dokumen tertinggi berdasarkan nilai relevansi yang diranking menggunakkan BM25. Fungsi ini berbeda dengan fungsi "exact_top_k" sebelumnya karena fungsi ini menggunakkan model BM25.

```
def exact_top_k_bm25(doc_dict, rank_score, k):
    relevance_scores = {}
    i = 0
    for doc_id in doc_dict.keys():
        relevance_scores[doc_id] = rank_score[i]
        i = i + 1

    sorted_value = OrderedDict(sorted(relevance_scores.items(), key=lambda x: x[1], reverse = True))
    top_k = {j: sorted_value[j] for j in list(sorted_value)[:k]}
    return top_k
```

Guna menjalankan program yang telah dibangun, dibuatlah sebuah fungsi main yang digunakkan untuk mendefinisikan berbagai variabel yang digunakan, memanggil fungsi, dan lain sebagainya.

```python
def main():
  # path berisi lokasi file-file berita
  path = "D:/RAIHAN STIS/Perkuliahan/SEMESTER 5/Praktikum INFORMATION RETRIEVAL/Pertemuan (2)/berita"

  berita = cleaning_file_berita(path)

  doc_dict = create_doct_dict(berita)

  inverted_index = create_inverted_index(berita)
  vocab = list(inverted_index.keys())
  tf_idf = tfidf(vocab, termFrequencyInDoc(vocab, doc_dict), inverseDocFre(vocab, wordDocFre(vocab, doc_dict), len(doc_dict)), doc_dict)
  TD = termDocumentMatrix(vocab, tf_idf, doc_dict)

  query = "vaksin corona jakarta"
  idf = inverseDocFre(vocab, wordDocFre(vocab, doc_dict), len(doc_dict))
  tf_query = termFrequency(vocab, query)
  TQ = termQueryMatrix(vocab, tf_query, idf)

  top_3 = exact_top_k(doc_dict, TD, TQ[:, 0], 3)
  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan VSM berbasis Cossine Similarity: ")
  print(top_3)


  doc_scores = construct_bm25(query, doc_dict)

  top_3_bm25 = exact_top_k_bm25(doc_dict, doc_scores, 3)
  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan Rank Okapi BM25: ")
  print(top_3_bm25)

main()
```

Setelah program dijalankan, maka akan terlihat output sebagai berikut. Terlihat bahwa terdapat perbedaan urutan antara perankingan menggunakkan VSM berbasis Cosine Similarity dan perankingan menggunakkan Okapi BM25. Hal tersebut dikarenakan BM25 juga memerhatikan panjang dari dokumen yang diranking. Perhatikan juga bahwa skor kemiripan menggunakkan BM25 merupakan penjumlahan skor kemiripan dari masing-masing term penyusunnya sehingga skor kemiripan yang dihasilkan bisa lebih dari 1.

```
(base) D:\RAIHAN STIS\Perkuliahan\SEMESTER 5\Praktikum INFORMATION RETRIEVAL\Pertemuan (8)>python penugasan8.py

Skor top 3 berita yang paling relevan dengan query menggunakkan VSM berbasis Cossine Similarity:
{2: 0.305441706917711, 3: 0.30457740843687225, 4: 0.07688776837468171}

Skor top 3 berita yang paling relevan dengan query menggunakkan Rank Okapi BM25:
{3: 1.1354275051625886, 2: 0.8285454488053711, 5: 0.562780808607297}
```