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

# 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.

A screen shot of a computer

Description automatically generated

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.

A screen shot of a computer code

Description automatically generated

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.

A screen shot of a computer code

Description automatically generated

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

A screenshot of a computer program

Description automatically generated

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.

A black screen with white text

Description automatically generated