Nama : Raihan Rahmanda Junianto

NIM : 222112303

Kelas : 3SD2

**Penugasan Praktikum 9 Information Retrieval**

Permasalahan:

Buat fungsi untuk menampilkan 3 list dokumen yang terurut pada folder “berita” dengan query “vaksin corona jakarta”, berdasarkan standar query likelihood model serta query likelihood model dengan Laplace Smoothing, Jelinek-Mercer Smoothing, dan Dirichlet Smoothing. Bandingkan dengan hasil perankingan BM25 pada modul 8 serta cosine similarity pada modul 5.

Solusi:

Berdasarkan permasalahan di atas, dirancang 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 exact\_top\_k\_likelihood(doc\_dict, rank\_score, k):

    relevance\_scores = {}

    rank\_score = list(rank\_score.values())

    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 likelihood(tokenized\_query, doc\_dict):

  likelihood\_scores = {}

  vocab = set()

  for doc\_id in doc\_dict.keys():

      likelihood\_scores[doc\_id] = 1

      tokens = tokenisasi(doc\_dict[doc\_id])

      vocab.update(tokens)

      for q in tokenized\_query:

          likelihood\_scores[doc\_id]=likelihood\_scores[doc\_id]\*tokens.count(q)/len(tokens)

  return likelihood\_scores

def likelihood\_laplace(tokenized\_query, doc\_dict, vocab, alpha):

  likelihood\_scores = {}

  for doc\_id in doc\_dict.keys():

      likelihood\_scores[doc\_id] = 1

      tokens = tokenisasi(doc\_dict[doc\_id])

      for q in tokenized\_query:

          likelihood\_scores[doc\_id]=likelihood\_scores[doc\_id]\*(tokens.count(q)+alpha)/(len(tokens)+len(vocab)\*alpha)

  return likelihood\_scores

def likelihood\_jm(tokenized\_query, tokenized\_corpus, doc\_dict, lamda):

  likelihood\_scores = {}

  for doc\_id in doc\_dict.keys():

      likelihood\_scores[doc\_id] = 1

      tokens = tokenisasi(doc\_dict[doc\_id])

      for q in tokenized\_query:

          likelihood\_scores[doc\_id]=likelihood\_scores[doc\_id]\*((lamda\*tokens.count(q)/len(tokens))+((1-lamda)\*tokenized\_corpus.count(q)/len(tokenized\_corpus)))

  return likelihood\_scores

def likelihood\_dirichlet(tokenized\_query, tokenized\_corpus, doc\_dict, miu):

  likelihood\_scores = {}

  for doc\_id in doc\_dict.keys():

      likelihood\_scores[doc\_id] = 1

      tokens = tokenisasi(doc\_dict[doc\_id])

      for q in tokenized\_query:

          likelihood\_scores[doc\_id]=likelihood\_scores[doc\_id]\*(tokens.count(q)+miu\*tokenized\_corpus.count(q)/len(tokenized\_corpus))/(len(tokens)+miu)

  return likelihood\_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"

  tokenized\_query = tokenisasi(query)

  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)

  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan Rank Okapi BM25: ")

  print(exact\_top\_k\_bm25(doc\_dict, doc\_scores, 3))

  tokenized\_corpus = [j for sub in [tokenisasi(doc\_dict[doc\_id]) for doc\_id in doc\_dict] for j in sub]

  vocab = set(tokenized\_corpus)

  likelihood\_scores = likelihood(tokenized\_query, doc\_dict)

  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan standar likelihood query model: ")

  print(exact\_top\_k\_likelihood(doc\_dict, likelihood\_scores, 3))

  likelihood\_scores\_laplace = likelihood\_laplace(tokenized\_query, doc\_dict, vocab, 1)

  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan Laplace Smoothing: ")

  print(exact\_top\_k\_likelihood(doc\_dict, likelihood\_scores\_laplace, 3))

  likelihood\_scores\_jm = likelihood\_jm(tokenized\_query, tokenized\_corpus, doc\_dict, 0.5)

  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan Jelinek-Mercer Smoothing: ")

  print(exact\_top\_k\_likelihood(doc\_dict, likelihood\_scores\_jm, 3))

  likelihood\_scores\_dirichlet = likelihood\_dirichlet(tokenized\_query, tokenized\_corpus, doc\_dict, 2)

  print("\nSkor top 3 berita yang paling relevan dengan query menggunakkan Dirichlet Smoothing: ")

  print(exact\_top\_k\_likelihood(doc\_dict, likelihood\_scores\_dirichlet, 3))

main()

Program di atas merupakan suatu kode program yang digunakan untuk tiga list dokumen terurut dari folder berita dengan query yang telah ditentukan berdasarkan standar query likelihood model serta query likelihood model dengan Laplace Smoothing, Jelinek-Mercer Smoothing, dan Dirichlet Smoothing. Selain itu, program ini jug membandingkan model query likelihood dengan model-model sebelumnya, seperti BM25 dan Vector Space Model (VSM) berbasis Cosine Similarity. Sebagian besar fungsi dan library yang digunakan oleh program ini masih sama seperti praktikum sebelumnya, seperti fungsi VSM, BM25, menampilkan k dokumen teratas, dan lain sebagainya. Perbedaan yang terlihat yaitu adanya penambahan fungsi model likelihood beserta beberapa metode smoothingnya. Jika dilihat lebih lanjut, fungsi dari model likelihood dengan metode smoothingnya hampir sama, yang membedakan hanya formula untuk menghitung skor likelihoodnya saja.

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer

Description automatically generated

A colorful lines on a black background

Description automatically generated

A computer screen with colorful text

Description automatically generated

Selain itu, terdapat fungsi untuk mendapatkan top k dokumen berdasarkan skor likehoodnya.

A screen shot of a computer code

Description automatically generated

Kemudian, jika fungsi main dijalankan maka akan terlihat output sebagai berikut.

A black screen with white text

Description automatically generated

Berdasarkan output di atas, dapat diketahui bahwa top 3 dokumen yang dihasilkan melalui standar likelihood query model adalah berita3, berita1, dan berita2. Selain itu, perhatikan juga bahwa skor yang dihasilkan oleh model tersebut terdapat yang bernilai nol. Hal tersebut dikarenakan Standard query likelihood model akan menghasilkan probabilitas nol ketika tidak ada query yang muncul dalam dokumen sehingga tidak menghasilkan top k dokumen secara tepat. Oleh karena itu, perlu dilakukan smoothing menggunakkan beberapa metode, seperti Laplace, Jelinek-Mercer, dan Dirichlet.

Jika membandingkan masing-masing metode seperti model VSM, BM25, dan Likelihood beserta beberapa metode smoothingnya, dapat diketahui bahwa urutan top k dokumen yang dihasilkan oleh BM25 dan Dirichlet smooting sama, yaitu berita3, berita2, dan berita5. Urutan tersebut sedikit berbeda dengan metode Laplace smoothing yang menghasilkan urutan berita3, berita2, dan berita4 dan metode Jelinek-Mercer yang menghasilkan urutan berita3, berita5, dan berita2. Di sisi lain, hasil dari VSM juga tidak jauh berbeda dengan Metode Laplace. Perbedaan urutan dokumen yang relevan disebabkan oleh formulasi yang berbeda untuk masing-masing metode.