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.c
ount(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.

```
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
```

```python
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
```

```python
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
```

```python
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
```

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

```python
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
```

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

```
(base) D:\RAIHAN STIS\Perkuliahan\SEMESTER 5\Praktikum INFORMATION RETRIEVAL\Pertemuan (9)>python penugasan9.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}

Skor top 3 berita yang paling relevan dengan query menggunakkan standar likelihood query model:
{3: 2.3995488848096554e-05, 1: 0.0, 2: 0.0}

Skor top 3 berita yang paling relevan dengan query menggunakkan Laplace Smoothing:
{3: 3.553029408043729e-06, 2: 1.4927113702623906e-06, 4: 8.865638063517687e-07}

Skor top 3 berita yang paling relevan dengan query menggunakkan Jelinek-Mercer Smoothing:
{3: 1.9140602898614845e-05, 5: 7.23470433090469e-06, 2: 6.70915077655013e-06}

Skor top 3 berita yang paling relevan dengan query menggunakkan Dirichlet Smoothing:
{3: 2.3935720572870266e-05, 2: 7.721935801618969e-07, 5: 6.73163010036337e-07}
```

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.