Nama : Raihan Rahmanda Junianto

NIM : 222112303

Kelas : 3SD2

**Penugasan Praktikum 6 Information Retrieval**

Soal:

Buat fungsi main untuk mengevaluasi hasil sistem information retrieval untuk folder “berita” dengan query “vaksin corona jakarta” yang telah dikerjakan pada modul 5. Note: Dokumen sebenarnya yang sesuai query berdasarkan relevance judgement yaitu berita2 dan berita3.

Jawaban:

Berdasarkan persoalan di atas, maka dirumuskan 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()

# 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)

# 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]

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]

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)

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\_k = {j: sorted\_value[j] for j in list(sorted\_value)[:k]}

    return top\_k

def compute\_prf\_metrics(I, score, I\_Q):

    # Compute rank and sort documents according to rank

    K = len(I)

    index\_sorted = np.flip(np.argsort(score))

    I\_sorted = I[index\_sorted]

    rank = np.argsort(index\_sorted) + 1

    rank\_sorted = np.arange(1, K+1)

    # Compute relevance function X\_Q (indexing starts with zero)

    X\_Q = np.isin(I\_sorted, I\_Q)

    # Compute precision and recall values (indexing starts with zero)

    M = len(I\_Q)

    P\_Q = np.cumsum(X\_Q) / np.arange(1, K+1)

    R\_Q = np.cumsum(X\_Q) / M

    # Break-even point

    BEP = P\_Q[M-1]

    # Maximal F-measure

    sum\_PR = P\_Q + R\_Q

    sum\_PR[sum\_PR == 0] = 1 # Avoid division by zero

    F\_Q = 2 \* (P\_Q \* R\_Q) / sum\_PR

    F\_max = F\_Q.max()

    # Average precision

    P\_average = np.sum(P\_Q \* X\_Q) / len(I\_Q)

    return P\_Q, R\_Q, F\_Q, BEP, F\_max, P\_average, X\_Q, rank, I\_sorted, rank\_sorted

top\_3 = exact\_top\_k(doc\_dict, TD, TQ[:, 0], 3)

print("Top 3 document: ")

print(top\_3)

rel\_judgement1 = {1:0, 2:1, 3:1, 4:0, 5:0}

import pandas as pd

from matplotlib import pyplot as plt

def main(top, rel\_judgement):

    rel\_docs = []

    for doc\_id, rel in rel\_judgement.items():

        if rel==1:

            rel\_docs.append(doc\_id)

    retrieved\_rel\_doc3 = [value for value in list(top.keys()) if value in rel\_docs]

    prec3 = len(retrieved\_rel\_doc3)/len(top\_3)\*100

    rec3 = len(retrieved\_rel\_doc3)/len(rel\_docs)\*100

    fScore3 = 2 \* prec3 \* rec3 / (prec3 + rec3)

    print("\nNilai Precision, Recall, dan F-score untuk top-3 document: ")

    print("precision: ", prec3)

    print("recall: ",rec3)

    print("F-score: ", fScore3)

    relevance\_score1 = {}

    i = 0

    for doc\_id in doc\_dict.keys():

        relevance\_score1[doc\_id] = cosine\_sim(TQ[:, 0], TD[:, i])

        i = i + 1

    I = np.array(list(relevance\_score1.keys()))

    score = np.array(list(relevance\_score1.values()))

    I\_Q = np.array([2,3])

    output = compute\_prf\_metrics(I, score, I\_Q)

    P\_Q, R\_Q, F\_Q, BEP, F\_max, P\_average, X\_Q, rank, I\_sorted, rank\_sorted = output

    # Arrange output as tables

    score\_sorted = np.flip(np.sort(score))

    df = pd.DataFrame({'Rank': rank\_sorted, 'ID': I\_sorted,

                        'Score': score\_sorted,

                        '$$ \chi\_\mathcal{Q} $$': X\_Q,

                        'P(r)': P\_Q,

                        'R(r)': R\_Q,

                        'F(r)': F\_Q})

    print('\nTable matriks evaluasi untuk seluruh file berita:')

    print(df)

    print('Break-even point = %.2f' % BEP)

    print('F\_max = %.2f' % F\_max)

    print('Average precision =', np.round(P\_average, 5))

    figsize=(3, 3)

    fig, ax = plt.subplots(1, 1, figsize=figsize)

    plt.plot(R\_Q, P\_Q, linestyle='--', marker='o', color='k', mfc='r')

    plt.xlim([0, 1.1])

    plt.ylim([0, 1.1])

    ax.set\_aspect('equal', 'box')

    plt.title('PR curve')

    plt.xlabel('Recall')

    plt.ylabel('Precision')

    plt.grid()

    plt.tight\_layout()

    ax.plot(BEP, BEP, color='green', marker='o',

    fillstyle='none', markersize=15)

    ax.set\_title('PR curve')

    plt.show()

main(top\_3, rel\_judgement1)

Kode program di atas dirancang untuk melakukan evaluasi terhadap sistem information retrieval yang telah dibuat sebelumnya. Sebagai catatan, pada kode program di atas, proses untuk menentukan indeks terbalik hingga tiga dokumen teratas dengan tingkat cosine similarity tertinggi masih sama seperti kode program praktikum sebelumnya. Evaluasi dilakukan dengan melihat nilai presisi, recall, F-score serta kurva antara ketiga komponen tersebut.

Pada kode program di atas terdapat dua fungsi tambahan. Fungsi yang pertama adalah fungsi “compute\_prf\_metrics(I, score, I\_Q)” yang digunakan untuk menghitung beberapa nilai dalam matriks evaluasi untuk sekumpulan dokumen yang diurutkan berdasarkan skor cosine similarity. Komponen yang dimaksud adalah presisi, recall, dan F-score. Oleh karena itu, fungsi tersebut mengembalikan nilai variabel precision (P\_Q), recall (R\_Q), F-measure (F\_Q), BEP, F-measure maksimum (F\_max), average precision (P\_average), fungsi relevansi (X\_Q), peringkat (rank), dokumen yang diurutkan (I\_sorted), dan peringkat dokumen yang diurutkan (rank\_sorted) yang nantinya akan dimasukkan ke dalam sebuah tabel matriks evaluasi.

A screen shot of a computer program

Description automatically generated

Fungsi selanjutnya adalah fungsi main yang dapat menghasilkan nilai matriks evaluasi, tabel matriks evaluasi, serta kurva dari nilai matriks evaluasinya.

def main(top, rel\_judgement):

    rel\_docs = []

    for doc\_id, rel in rel\_judgement.items():

        if rel==1:

            rel\_docs.append(doc\_id)

    retrieved\_rel\_doc3 = [value for value in list(top.keys()) if value in rel\_docs]

    prec3 = len(retrieved\_rel\_doc3)/len(top\_3)\*100

    rec3 = len(retrieved\_rel\_doc3)/len(rel\_docs)\*100

    fScore3 = 2 \* prec3 \* rec3 / (prec3 + rec3)

    print("\nNilai Precision, Recall, dan F-score untuk top-3 document: ")

    print("precision: ", prec3)

    print("recall: ",rec3)

    print("F-score: ", fScore3)

    relevance\_score1 = {}

    i = 0

    for doc\_id in doc\_dict.keys():

        relevance\_score1[doc\_id] = cosine\_sim(TQ[:, 0], TD[:, i])

        i = i + 1

    I = np.array(list(relevance\_score1.keys()))

    score = np.array(list(relevance\_score1.values()))

    I\_Q = np.array([2,3])

    output = compute\_prf\_metrics(I, score, I\_Q)

    P\_Q, R\_Q, F\_Q, BEP, F\_max, P\_average, X\_Q, rank, I\_sorted, rank\_sorted = output

    # Arrange output as tables

    score\_sorted = np.flip(np.sort(score))

    df = pd.DataFrame({'Rank': rank\_sorted, 'ID': I\_sorted,

                        'Score': score\_sorted,

                        '$$ \chi\_\mathcal{Q} $$': X\_Q,

                        'P(r)': P\_Q,

                        'R(r)': R\_Q,

                        'F(r)': F\_Q})

    print('\nTable Matriks Evaluasi untuk seluruh file berita:')

    print(df)

    print('Break-even point = %.2f' % BEP)

    print('F\_max = %.2f' % F\_max)

    print('Average precision =', np.round(P\_average, 5))

    figsize=(3, 3)

    fig, ax = plt.subplots(1, 1, figsize=figsize)

    plt.plot(R\_Q, P\_Q, linestyle='--', marker='o', color='k', mfc='r')

    plt.xlim([0, 1.1])

    plt.ylim([0, 1.1])

    ax.set\_aspect('equal', 'box')

    plt.title('PR curve')

    plt.xlabel('Recall')

    plt.ylabel('Precision')

    plt.grid()

    plt.tight\_layout()

    ax.plot(BEP, BEP, color='green', marker='o',

    fillstyle='none', markersize=15)

    ax.set\_title('PR curve')

    plt.show()

Sebelum memanggil fungsi main, diperlukan pendefinisian variabel tiga dokumen teratas serta label relevansinya berdasarkan query ‘virus corona jakarta’. Pada label relevansi, nilai untuk berita2 dan berita3 adalah 1 karena kedua dokumen tersebut sesuai dengan query (diketahui pada soal), sedangkan yang lainnya bernilai 0.

A computer code with text

Description automatically generated

Selanjutnya, dilakukan perhitungan nilai matriks evaluasi, yaitu presisi, recall, dan f-score untuk tiga dokumen teratas berdasarkan label relevansinya.

A screen shot of a computer code

Description automatically generated

Lalu, untuk menghitung nilai matriks evaluasi untuk seluruh dokumen pada folder berita dilakukan dengan menghitung skor relevansi untuk semua dokumen serta memanggil kembali fungsi “compute\_prf\_metrics(I, score, I\_Q)” untuk menghitung presisi, recall, serta f-score masing-masing dokumen. Pada kode kali ini, tabel dibuat menggunakkan dataframe dengan library pandas.

A computer screen shot of a program

Description automatically generated

Terakhir, kurva dari presisi dan recall di buat menggunakkan library pyplot. Kode tersebut menghasilkan PR curve dari data precision (P\_Q) dan recall (R\_Q) dengan menandai Break-Even Point (BEP) pada kurva tersebut. Visualisasi ini membantu untuk melihat hubungan antara precision dan recall pada suatu model atau sistem.

A screen shot of a computer code

Description automatically generated

Berikut merupakan keseluruhan output dari program yang dirancang.

A screenshot of a computer

Description automatically generated

A screen shot of a graph

Description automatically generated