Nama : Raihan Rahmanda Junianto

NIM : 222112303

Kelas : 3SD2

**Penugasan Praktikum 7 Information Retrieval**

**Permasalahan:**

Buat kode untuk mengimplementasikan relevance feedback pada folder “berita” dengan query “vaksin corona jakarta” yang telah dikerjakan pada modul 5. Bandingkan top 5 dokumen yang dikembalikan sebelum dan sesudah dilakukan relevance feedback. Note: Dokumen yang relevan yaitu berita2 dan berita3.

**Solusi:**

Berdasarkan permasalahan di atas, 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

import matplotlib.pyplot as plt

from sklearn.decomposition import TruncatedSVD

# 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

DT = TD.transpose()

print("Matriks DT: ")

print(DT)

model = TruncatedSVD(n\_components=2, random\_state=7).fit(DT)

DT\_reduced = model.transform(DT)

QT\_reduced = model.transform(TQ.transpose())

print("\nMatriks QT yang sudah direduksi:")

print(QT\_reduced)

print("\nMatriks DT yang sudah direduksi:")

print(DT\_reduced)

plt.scatter(DT\_reduced[:, 0], DT\_reduced[:, 1])

plt.scatter(QT\_reduced[:, 0], QT\_reduced[:, 1], color=["red"])

labels=list(doc\_dict.keys())

for i, txt in enumerate(labels):

    plt.annotate(txt, (DT\_reduced[i, 0], DT\_reduced[i, 1]))

plt.annotate("query", (QT\_reduced[0, 0], QT\_reduced[0, 1]))

plt.show()

print("\ntop 5 dokumen yang dikembalikan sebelum dilakukan relevance feedback: ")

top\_5 = exact\_top\_k(doc\_dict, TD, TQ[:, 0], 5)

print(top\_5)

rel\_vecs\_id = [2,3]

nrel\_vecs\_id = [3,4,5]

doc\_ids = list(doc\_dict.keys())

rel\_vecs = []

for doc in rel\_vecs\_id:

    rel\_vecs.append(DT[doc\_ids.index(doc),:])

nrel\_vecs = []

for doc in nrel\_vecs\_id:

    nrel\_vecs.append(DT[doc\_ids.index(doc),:])

query\_vecs = TQ.transpose()

alpha = 1

beta = 0.75

gamma = 0.15

# Update query vectors with Rocchio algorithm

query\_vecs = alpha \* query\_vecs + beta \* np.mean(rel\_vecs, axis=0) - gamma \* np.mean(nrel\_vecs, axis=0)

query\_vecs[query\_vecs<0] = 0 #negative value => 0

print("\ntop 5 dokumen yang dikembalikan setelah dilakukan relevance feedback: ")

top\_5 = exact\_top\_k(doc\_dict, TD, query\_vecs[0, :].transpose(), 5)

print(top\_5)

QT1\_reduced = model.transform(query\_vecs)

plt.scatter(DT\_reduced[:, 0], DT\_reduced[:, 1])

plt.scatter(QT\_reduced[:, 0], QT\_reduced[:, 1], color=["red"])

plt.scatter(QT1\_reduced[:, 0], QT1\_reduced[:, 1], color=["green"])

doc\_ids=list(doc\_dict.keys())

for i, txt in enumerate(doc\_ids):

    plt.annotate(txt, (DT\_reduced[i, 0], DT\_reduced[i, 1]))

plt.annotate("query", (QT\_reduced[0, 0], QT\_reduced[0, 1]))

plt.annotate("new query", (QT1\_reduced[:, 0], QT1\_reduced[:, 1]))

plt.show()

Kode program kali ini bertujuan untuk mengetahui relevance feedback dari query tertentu. Praktikum kali ini masih menggunakkan beberapa baris kode pada praktikum sebelumnya, khususnya untuk membangun matriks Term-Document (TD) dan Term-Query (TQ) serta mendapatkan beberapa dokumen teratas berdasarkan skor relevansinya. Pada kode program ini terdapat dua library tambahan yaitu TruncatedSVD yang digunakan untuk melakukan operasi dekomposisi untuk mereduksi vector menjadi dua dimensi sehingga bisa ditampilkan serta library pyplot untuk memplot vector yang dimaksud.

A screenshot of a computer screen

Description automatically generated

Langkah pertama yang dilakukan adalah mereduksi matriks TQ dan TD menggunakkan operasi Singular Value Decomposition (SVD). Sebelum direduksi, dilakukan operasi transpose terhadap kedua matriks tersebut.

A computer code with colorful text

Description automatically generated

Berikut merupakan tampilan dari matriks TQ dan TD yang sudah direduksi sehingga vector vector penyusunnya dapat diplotkan pada bidang berdimensi dua.

A black screen with white text

Description automatically generated

Selanjutnya, dilakukan visualisasi menggunakkan scatter plot untuk melihat kedekatan antara dokumen terhadap query berdasarkan matriks TQ dan TD yang sudah direduksi sebelumnya.

A computer code with text

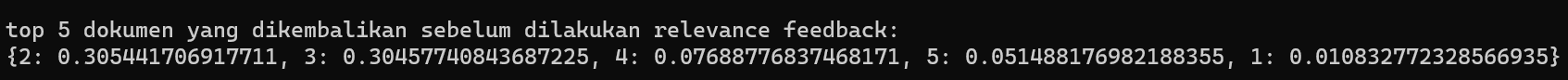
Description automatically generated with medium confidence

Berikut merupakan hasil visualisasi menggunakkan scatter plot berdasarkan kedua matriks tersebut. Pada plot tersebut, label ditambahkan untuk setiap titik, dan warna merah digunakan untuk membedakan titik-titik yang berasal dari matriks QT yang direduksi (query). Berdasarkan plot tersebut terlihat bahwa jarak query dengan dokumen yang relevan yaitu berita2 dan berita3 masih tergolong jauh.

A graph with numbers and points

Description automatically generated

Selanjutnya, proses akan berlanjut untuk melihat perbandingan antara lima dokumen teratas berdasarkan skor relevansi sebelum dan sesudah dilakukan relevance feedback. Berikut disajikan lima dokumen teratas sebelum dilakukan relevance feedback. Hasil yang sama juga terlihat pada praktikum sebelumnya.



Selanjutnya, untuk mendapatkan lima dokumen teratas lainnya, perlu dilakukan perbaikan query menggunakkan algoritma Rocchio. Sebelumnya, dilakukan pelabelan relevansi seperti yang telah dilakukan pada pertemuan sebelumnya. Berdasarkan soal, dokumen yang relevan adalag berita2 dan berita3.

A computer screen shot of a program code

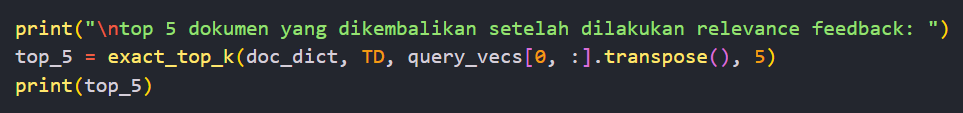
Description automatically generated

Kemudian, menggunakkan nilai , dilakukan pembaruan query menggunakkan formula yang terdapat pada algortima Rocchio. Algoritma ini akan mendekatkan vector query dengan dokumen-dokumen yang relevan dan menjauhkannya dari dokumen-dokumen yang tidak relevan.

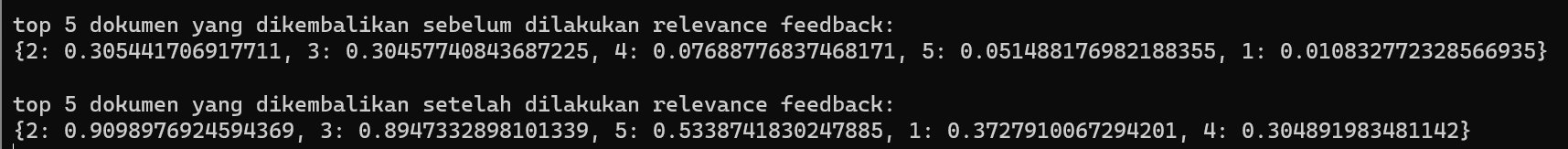
A screen shot of a computer code

Description automatically generated

Selanjutnya, dilakukan pengambilan lima dokumen teratas berdasarkan skor relevansi terhadap query hasil perbaikan tersebut.



Berikut disajikan perbandingan antara perbandingan antara lima dokumen teratas berdasarkan skor relevansi sebelum dan sesudah dilakukan relevance feedback.



Berdasarkan output di atas, terlihat pada terdapat peningkatan skor relevansi masing-masing dokumen terhadap query baru hasil perbaikan. Selain itu, terdapat pula perubahan posisi urutan (rank) dari lima dokumen teratas. Perubahan urutan atau rank di antaranya yaitu berita5 ke posisi 3, berita 1 ke posisi4, serta berita 4 ke posisi 5. Artinya, relevance feedback menggunakkan algortima Rocchio berhasil memperbaiki query agar relevansi dengan dokumen-dokumennya semakin besar, terlihat dari skor relevansi yang semakin besar pula. Lalu, untuk memverifikasi hal tersebut, dilakukan kembali plot berdasarkan query yang baru.

A computer screen shot of text

Description automatically generated

Berikut disajikan visualisasi scatter plot untuk melihat bagaimana kedekatan query baru terhadap dokumen yang terdapat pada folder berita. Pada plot tersebut, label ditambahkan untuk setiap titik, dan warna merah digunakan untuk menunjukkan kueri awal (initial query), serta warna hijau menunjukkan vector kueri hasil dari perbaikan yang telah dilakukan sebelumnya. Berdasarkan plot tersebut, terlihat bahwa kueri baru memiliki jarak yang lebih dekat dengan dokumen yang relevan dibandingkan kueri awal.

A graph with numbers and points

Description automatically generated with medium confidence