Nama : Raihan Rahmanda Junianto

NIM : 222112303

Kelas : 3SD2

**Responsi Praktikum 4 Information Retrieval**

1. **Buat vector space model dengan menggunakan sekumpulan dokumen pada folder “berita”.**

Berdasarkan soal di atas, maka dibuatlah kode program sebagai berikut.

import os

from spacy.lang.id import Indonesian

from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

from spacy.lang.id.stop\_words import STOP\_WORDS

nlp = Indonesian()

path = "D:/RAIHAN STIS/Perkuliahan/SEMESTER 5/Praktikum INFORMATION RETRIEVAL/Pertemuan (2)/berita"

def read\_text\_file(file\_path):

    with open(file\_path, 'r') as f:

        content = f.read()

    return content

def preprocess\_text(text):

    stemmer = StemmerFactory().create\_stemmer()

    stemmed\_text = stemmer.stem(text)

    doc = nlp(stemmed\_text)

    tokens = [token.text for token in doc if token.text.lower() not in STOP\_WORDS]

    return tokens

inverted\_index = {}

doc\_dict = {}

document\_index = 1

for file in os.listdir(path):

    if os.path.isfile(os.path.join(path, file)) and file.endswith(".txt"):

        file\_path = os.path.join(path, file)

        text = read\_text\_file(file\_path)

        cleaned\_tokens = preprocess\_text(text)

        doc\_dict[document\_index] = " ".join(cleaned\_tokens)

        document\_index += 1

        # Update the inverted index

        for token in set(cleaned\_tokens):  # Use set to avoid duplicate documents

            inverted\_index.setdefault(token, []).append(file)

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 wordDocFre(vocab, doc\_dict):

    df = {}

    for word in vocab:

        frq = 0

        for doc in doc\_dict.values():

            if word in 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

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)

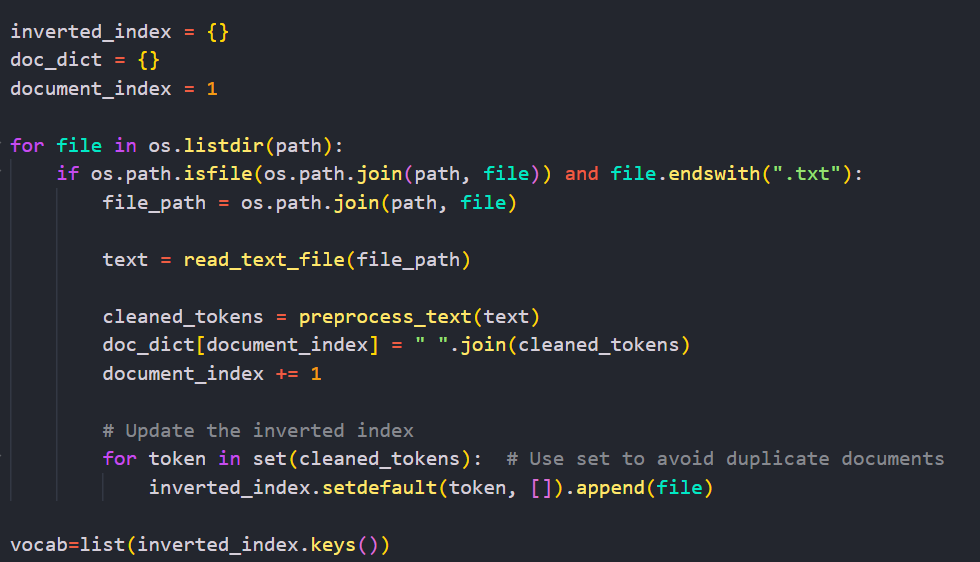
        ind2 = list(tf\_idf.keys()).index(doc\_id)

        TD[ind1][ind2] = tf\_idf[doc\_id][word]

print(TD)

Proses pembuatan vektor space model diawali dengan menyusun indeks terbalik yang telah dilakukan pada praktikum pertemuan sebelumnya. Pada praktikum kali ini, library yang digunakan masih sama seperti sebelumnya, hanya saja ditambahkan library “NumPy” untuk mengakomodasi operasi numerik seperti array multidimensi (matriks).

Setelah indeks terbalik tersedia, indeks tersebut akan disimpan ke dalam variabel vocab yang berisi kumpulan term pada corpus. Selanjutnya, terdapat juga inisialisasi variabel doc\_dict yang nantinya akan memuat isi dokumen (kalimat atau paragraf) yang telah dilakukan preprocessing, termasuk tokenisasi dan stemming dengan doc\_id sebagai key-nya. Pada potongan kode program ini juga menjelaskan bahwa terdapat inisialisasi document\_index yang berperan sebagai indeks bagi variabel doc\_dict untuk kemudahan pengaksesan kamus tersebut. Sebagai gambaran, potongan kode program dapat dilihat pada gambar berikut.



Proses selanjutnya adalah menghitung document frequency serta inverse document frequency. Fungsi pertama yang dijalankan adalah fungsi “termFrequencyInDoc(vocab, doc\_dict)”. Fungsi tersebut akan menghitung Term Frequency (TF) yaitu jumlah kemunculan setiap kata dalam setiap dokumen yang terdapat di dalam variabel doc\_dict. Dalam kata lain, TF mengukur seberapa sering sebuah kata muncul dalam dokumen tertentu. Fungsi ini mengembalikan kamus (dictionary) yang berisi TF untuk setiap kata dalam setiap dokumen.

A screen shot of a computer code

Description automatically generated

Fungsi selanjutnya adalah fungsi “wordDocFre(vocab, doc\_dict)” yang menghitung Document Frequency (DF) untuk setiap kata dalam vocab. DF mengukur berapa banyak dokumen yang mengandung sebuah kata tertentu. Fungsi ini mengembalikan variabel kamus yang berisi DF untuk setiap kata dalam vocab. Pada fungsi ini tidak perlu dilakukan tokenisasi lagi pada doc karena variabel doc\_dict sudah melalui proses preprocessing, termasuk tokensasi.

A computer screen with colorful text

Description automatically generated

Selanjutnya, terdapat fungsi “inverseDocFre(vocab,doc\_fre,length)” yang menggunakan library “NumPy” untuk menghitung Inverse Document Frequency (IDF) untuk setiap kata dalam vocab. IDF digunakan untuk memberikan bobot kepada kata-kata yang jarang muncul dalam dokumen tetapi mungkin memiliki nilai informatif yang tinggi. Fungsi ini mengembalikan kamus yang berisi IDF untuk setiap kata dalam vocab. Fungsi ini menggunakan dengan rumus logaritma dari total dokumen yang ada dibagi oleh Document Frequency dari kata tersebut.

A computer code with colorful text

Description automatically generated

Setelah proses di atas selesai, maka dilanjutkan dengan menghitung nilai TF-IDF yang nantinya akan dimasukkan ke dalam matriks Term-Document (TD). Pada fungsi “tfidf(vocab,tf,idf\_scr,doc\_dict)”, nilai TF-IDF dihitung dengan mengalikan variabel “tf” dan “idf” berdasarkan vocab serta doc\_id.

A computer screen shot of a computer code

Description automatically generated

Setelah nilai TF-IDF masing-masing kata dalam setiap dokumen diperoleh, maka akan dimasukkan ke dalam matriks TD.

A screen shot of a computer code

Description automatically generated

Setelah semua proses dilakukan, maka output yang tertampil adalah sebagai berikut.

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **Dari 5 file pada folder “berita”, hitung skor kemiripan antara berita yang satu dan lainnya masing-masing dengan edit distance, jaccard similarity, euclidian distance, dan cosine similarity.**

Melanjutkan program pada nomor 1, dilakukan penghitungan skor kemiripan antara berita yang satu dan lainnya masing-masing dengan edit distance, jaccard similarity, euclidian distance, dan cosine similarity.

Proses edit distance dilakukan oleh fungsi “edit\_distance(string1, string2)”, yaitu untuk menghitung jarak edit (edit distance) antara dua string string1 dan string2. Jarak edit adalah jumlah minimum dari operasi (penghapusan, penggantian, atau penyisipan) yang diperlukan untuk mengubah satu string menjadi string lainnya. Fungsi tersebut akan mengembalikan nilai berupa angka jarak kedua string tersebut.

A computer screen shot of code

Description automatically generated

Selanjutnya, terdapat fungsi “jaccard\_sim(list1, list2)” untuk menghitung indeks kesamaan Jaccard antara dua list list1 dan list2. Indeks ini mengukur kesamaan antara dua himpunan (set) dengan menghitung rasio jumlah elemen yang sama dengan jumlah elemen yang berbeda antara kedua set. Fungsi ini mengembalikan nilai kesamaan dalam bentuk pecahan (float).

A screen shot of a computer code

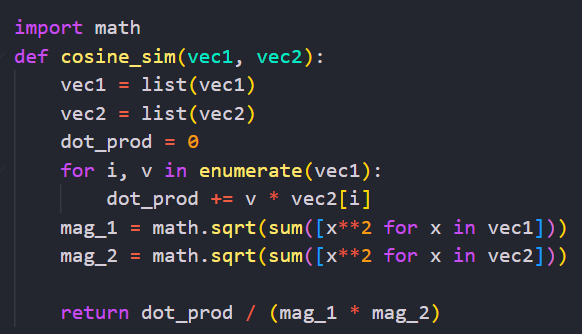
Description automatically generated

Selain itu, terdapat juga fungsi “euclidian\_dist(vec1, vec2)” yang berfungsi untuk menghitung jarak Euclidean antara dua vektor vec1 dan vec2. Jarak Euclidean adalah panjang garis lurus antara dua titik dalam ruang berdimensi n. Fungsi ini mengembalikan nilai jarak antara dua vektor.

A computer code with colorful text

Description automatically generated

Dan yang terakhir terdapat fungsi “cosine\_sim(vec1, vec2)” menggunakkan library “Math” untuk melakukan operasi matematika. Fungsi ini digunakan untuk menghitung kesamaan kosinus antara dua vektor vec1 dan vec2. Kesamaan kosinus mengukur sejauh mana dua vektor memiliki arah yang serupa dalam ruang berdimensi n. Fungsi ini mengembalikan nilai kesamaan kosinus dalam bentuk pecahan



Berikut disajikan tampilan keseluruhan kode program beserta output yang dihasilkan.

import os

from spacy.lang.id import Indonesian

from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

from spacy.lang.id.stop\_words import STOP\_WORDS

nlp = Indonesian()

path = "D:/RAIHAN STIS/Perkuliahan/SEMESTER 5/Praktikum INFORMATION RETRIEVAL/Pertemuan (2)/berita"

def read\_text\_file(file\_path):

    with open(file\_path, 'r') as f:

        content = f.read()

    return content

def preprocess\_text(text):

    stemmer = StemmerFactory().create\_stemmer()

    stemmed\_text = stemmer.stem(text)

    doc = nlp(stemmed\_text)

    tokens = [token.text for token in doc if token.text.lower() not in STOP\_WORDS]

    return tokens

inverted\_index = {}

doc\_dict = {}

document\_index = 1

for file in os.listdir(path):

    if os.path.isfile(os.path.join(path, file)) and file.endswith(".txt"):

        file\_path = os.path.join(path, file)

        text = read\_text\_file(file\_path)

        cleaned\_tokens = preprocess\_text(text)

        doc\_dict[document\_index] = " ".join(cleaned\_tokens)

        document\_index += 1

        # Update the inverted index

        for token in set(cleaned\_tokens):  # Use set to avoid duplicate documents

            inverted\_index.setdefault(token, []).append(file)

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 wordDocFre(vocab, doc\_dict):

    df = {}

    for word in vocab:

        frq = 0

        for doc in doc\_dict.values():

            if word in 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

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]

def edit\_distance(string1, string2):

    if len(string1) > len(string2):

        difference = len(string1) - len(string2)

        string1[:difference]

        n = len(string2)

    elif len(string2) > len(string1):

        difference = len(string2) - len(string1)

        string2[:difference]

        n = len(string1)

    for i in range(n):

        if string1[i] != string2[i]:

            difference += 1

    return difference

print("\nSkor Kemiripan dengan Edit Distance")

print("Berita 1 dengan berita 2 : ", edit\_distance(doc\_dict[1], doc\_dict[2]))

print("Berita 1 dengan berita 3 : ", edit\_distance(doc\_dict[1], doc\_dict[3]))

print("Berita 1 dengan berita 4 : ", edit\_distance(doc\_dict[1], doc\_dict[4]))

print("Berita 1 dengan berita 5 : ", edit\_distance(doc\_dict[1], doc\_dict[5]))

print("Berita 2 dengan berita 3 : ", edit\_distance(doc\_dict[2], doc\_dict[3]))

print("Berita 2 dengan berita 4 : ", edit\_distance(doc\_dict[2], doc\_dict[4]))

print("Berita 2 dengan berita 5 : ", edit\_distance(doc\_dict[2], doc\_dict[5]))

print("Berita 3 dengan berita 4 : ", edit\_distance(doc\_dict[3], doc\_dict[4]))

print("Berita 3 dengan berita 5 : ", edit\_distance(doc\_dict[3], doc\_dict[5]))

print("Berita 4 dengan berita 5 : ", edit\_distance(doc\_dict[4], doc\_dict[5]))

def jaccard\_sim(list1, list2):

    intersection = len(list(set(list1).intersection(list2)))

    union = (len(list1) + len(list2)) - intersection

    return float(intersection) / union

print("\nSkor Kemiripan dengan Jaccard Similarity")

print("Berita 1 dengan berita 2 : ", jaccard\_sim(doc\_dict[1].split(" "), doc\_dict[2].split(" ")))

print("Berita 1 dengan berita 3 : ", jaccard\_sim(doc\_dict[1].split(" "), doc\_dict[3].split(" ")))

print("Berita 1 dengan berita 4 : ", jaccard\_sim(doc\_dict[1].split(" "), doc\_dict[4].split(" ")))

print("Berita 1 dengan berita 5 : ", jaccard\_sim(doc\_dict[1].split(" "), doc\_dict[5].split(" ")))

print("Berita 2 dengan berita 3 : ", jaccard\_sim(doc\_dict[2].split(" "), doc\_dict[3].split(" ")))

print("Berita 2 dengan berita 4 : ", jaccard\_sim(doc\_dict[2].split(" "), doc\_dict[4].split(" ")))

print("Berita 2 dengan berita 5 : ", jaccard\_sim(doc\_dict[2].split(" "), doc\_dict[5].split(" ")))

print("Berita 3 dengan berita 4 : ", jaccard\_sim(doc\_dict[3].split(" "), doc\_dict[4].split(" ")))

print("Berita 3 dengan berita 5 : ", jaccard\_sim(doc\_dict[3].split(" "), doc\_dict[5].split(" ")))

print("Berita 4 dengan berita 5 : ", jaccard\_sim(doc\_dict[3].split(" "), doc\_dict[5].split(" ")))

def euclidian\_dist(vec1, vec2):

    temp = vec1 - vec2

    sum\_sq = np.dot(temp.T, temp)

    return np.sqrt(sum\_sq)

print("\nSkor Kemiripan dengan Euclidian Distance")

print("Berita 1 dengan berita 2 : ", euclidian\_dist(TD[:, 0], TD[:, 1]))

print("Berita 1 dengan berita 3 : ", euclidian\_dist(TD[:, 0], TD[:, 2]))

print("Berita 1 dengan berita 4 : ", euclidian\_dist(TD[:, 0], TD[:, 3]))

print("Berita 1 dengan berita 5 : ", euclidian\_dist(TD[:, 0], TD[:, 4]))

print("Berita 2 dengan berita 3 : ", euclidian\_dist(TD[:, 1], TD[:, 2]))

print("Berita 2 dengan berita 4 : ", euclidian\_dist(TD[:, 1], TD[:, 3]))

print("Berita 2 dengan berita 5 : ", euclidian\_dist(TD[:, 1], TD[:, 4]))

print("Berita 3 dengan berita 4 : ", euclidian\_dist(TD[:, 2], TD[:, 3]))

print("Berita 3 dengan berita 5 : ", euclidian\_dist(TD[:, 2], TD[:, 4]))

print("Berita 4 dengan berita 5 : ", euclidian\_dist(TD[:, 3], TD[:, 4]))

import math

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)

print("\nSkor Kemiripan dengan Cosine Similarity")

print("Berita 1 dengan berita 2 : ", cosine\_sim(TD[:, 0], TD[:, 1]))

print("Berita 1 dengan berita 3 : ", cosine\_sim(TD[:, 0], TD[:, 2]))

print("Berita 1 dengan berita 4 : ", cosine\_sim(TD[:, 0], TD[:, 3]))

print("Berita 1 dengan berita 5 : ", cosine\_sim(TD[:, 0], TD[:, 4]))

print("Berita 2 dengan berita 3 : ", cosine\_sim(TD[:, 1], TD[:, 2]))

print("Berita 2 dengan berita 4 : ", cosine\_sim(TD[:, 1], TD[:, 3]))

print("Berita 2 dengan berita 5 : ", cosine\_sim(TD[:, 1], TD[:, 4]))

print("Berita 3 dengan berita 4 : ", cosine\_sim(TD[:, 2], TD[:, 3]))

print("Berita 3 dengan berita 5 : ", cosine\_sim(TD[:, 2], TD[:, 4]))

print("Berita 4 dengan berita 5 : ", cosine\_sim(TD[:, 3], TD[:, 4]))

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated