222112058_Feza Raffa Arnanda_Penugasan Praktikum 5

September 23, 2023

1 Praktikum 5

Nama : Feza Raffa Arnanda

NIM : 222112058

Kelas : 2KS5

1.0.1 Preprocessing and Building Inverted Index

[ ]:
```
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
```

[ ]:
```
def tokenisasi(text):
    tokens = text.split(" ")
    return tokens
def stemming(text):
    from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
    # create stemmer
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()
    # stemming process
    output = stemmer.stem(text)
    return output
```

[ ]:
```
def stemming_sentence(text):
    output = ""
    for token in tokenisasi(text):
        output = output + stemming(token) + " "
    return output[:-1]
doc_dict_raw = {}
doc_dict_raw['doc1'] = "pengembangan sistem informasi penjadwalan"
doc_dict_raw['doc2'] = "pengembangan model analisis sentimen berita"
doc_dict_raw['doc3'] = "analisis sistem input output"
doc_dict_raw['doc4'] = "pengembangan sistem informasi akademik universitas"
doc_dict_raw['doc5'] = "pengembangan sistem cari berita ekonomi"
doc_dict_raw['doc6'] = "analisis sistem neraca nasional"
doc_dict_raw['doc7'] = "pengembangan sistem informasi layanan statistik"
doc_dict_raw['doc8'] = "pengembangan sistem pencarian skripsi di universitas"
doc_dict_raw['doc9'] = "analisis sentimen publik terhadap pemerintah"
```

```
doc_dict_raw['doc10'] = "pengembangan model klasifikasi sentimen berita"
doc_dict = {}
for doc_id,doc in doc_dict_raw.items():
    doc_dict[doc_id] = stemming_sentence(doc)
doc_dict
```

[ ]:
```
{'doc1': 'kembang sistem informasi jadwal',
 'doc2': 'kembang model analisis sentimen berita',
 'doc3': 'analisis sistem input output',
 'doc4': 'kembang sistem informasi akademik universitas',
 'doc5': 'kembang sistem cari berita ekonomi',
 'doc6': 'analisis sistem neraca nasional',
 'doc7': 'kembang sistem informasi layan statistik',
 'doc8': 'kembang sistem cari skripsi di universitas',
 'doc9': 'analisis sentimen publik hadap perintah',
 'doc10': 'kembang model klasifikasi sentimen berita'}
```

[ ]:
```
vocab = [] # Vocab ini isinya daftar kata yang sudah terstem
inverted_index = {}
for doc_id,doc in doc_dict.items():
    for token in tokenisasi(doc):
        print(token)
        if token not in vocab:
            vocab.append(token)
            inverted_index[token] = []
        if token in inverted_index:
            if doc_id not in inverted_index[token]:
                inverted_index[token].append(doc_id)
print(vocab)
print(inverted_index)
```

```
kembang
sistem
informasi
jadwal
kembang
model
analisis
sentimen
berita
analisis
sistem
input
output
kembang
sistem
informasi
akademik
```

```
universitas
kembang
sistem
cari
berita
ekonomi
analisis
sistem
neraca
nasional
kembang
sistem
informasi
layan
statistik
kembang
sistem
cari
skripsi
di
universitas
analisis
sentimen
publik
hadap
perintah
kembang
model
klasifikasi
sentimen
berita
['kembang', 'sistem', 'informasi', 'jadwal', 'model', 'analisis', 'sentimen',
'berita', 'input', 'output', 'akademik', 'universitas', 'cari', 'ekonomi',
'neraca', 'nasional', 'layan', 'statistik', 'skripsi', 'di', 'publik', 'hadap',
'perintah', 'klasifikasi']
{'kembang': ['doc1', 'doc2', 'doc4', 'doc5', 'doc7', 'doc8', 'doc10'], 'sistem':
['doc1', 'doc3', 'doc4', 'doc5', 'doc6', 'doc7', 'doc8'], 'informasi': ['doc1',
'doc4', 'doc7'], 'jadwal': ['doc1'], 'model': ['doc2', 'doc10'], 'analisis':
['doc2', 'doc3', 'doc6', 'doc9'], 'sentimen': ['doc2', 'doc9', 'doc10'],
'berita': ['doc2', 'doc5', 'doc10'], 'input': ['doc3'], 'output': ['doc3'],
'akademik': ['doc4'], 'universitas': ['doc4', 'doc8'], 'cari': ['doc5', 'doc8'],
'ekonomi': ['doc5'], 'neraca': ['doc6'], 'nasional': ['doc6'], 'layan':
['doc7'], 'statistik': ['doc7'], 'skripsi': ['doc8'], 'di': ['doc8'], 'publik':
['doc9'], 'hadap': ['doc9'], 'perintah': ['doc9'], 'klasifikasi': ['doc10']}
```

1.0.2 A. Exact Top K Document Retrieval

[ ]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import numpy as np

def wordDocFre(vocab, doc_dict):
    df = {}
    for word in vocab:
        frq = 0
        for doc in doc_dict.values():
          #if word in doc.lower().split():
            if word in doc:
                frq = frq + 1
        df[word] = frq
    return df
```

[ ]:
```
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 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)
```

```
[[1.31845373 1.31845373 0.         1.31845373 1.31845373 0.
  1.31845373 1.31845373 0.         1.31845373]
 [1.31845373 0.         1.31845373 1.31845373 1.31845373 1.31845373
  1.31845373 1.31845373 0.         0.        ]
 [2.01160091 0.         0.         2.01160091 0.         0.
  2.01160091 0.         0.         0.        ]
 [2.70474809 0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         2.29928298 0.         0.         0.         0.
  0.         0.         0.         2.29928298]
 [0.         1.78845736 1.78845736 0.         0.         1.78845736
  0.         0.         1.78845736 0.        ]
 [0.         2.01160091 0.         0.         0.         0.
  0.         0.         2.01160091 2.01160091]
 [0.         2.01160091 0.         0.         2.01160091 0.
  0.         0.         0.         2.01160091]
 [0.         0.         2.70474809 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         2.70474809 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         2.70474809 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         2.29928298 0.         0.
  0.         2.29928298 0.         0.        ]
 [0.         0.         0.         0.         2.29928298 0.
  0.         2.29928298 0.         0.        ]
 [0.         0.         0.         0.         2.70474809 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         2.70474809
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         2.70474809
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  2.70474809 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  2.70474809 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         2.70474809 0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         2.70474809 0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         2.70474809 0.        ]
 [0.         0.         0.         0.         0.         0.
```

```
 0.         0.         2.70474809 0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         2.70474809 0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         2.70474809]]
```

Hitung terlebih dahulu bobot TF.IDF untuk suatu query sehingga didapatkan suatu term-query matrix. Misalkan terdapat suatu query ”sistem informasi statistik”, untuk mencari dokumen yang paling sesuai dari koleksi dokumen pada praktikum sebelumnya

[ ]:
```
query = "sistem informasi statistik"
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))
```

Setelah itu, anda dapat menggunakan idf yang telah dihitung pada praktikum sebelumnya untuk mendapatkan bobot tf.idf dari query sehingga didapatkan suatu term-query matrix

[ ]:
```
# 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]
print(TQ)
```

```
[[0.        ]
 [1.31845373]
 [2.01160091]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [0.        ]
 [2.70474809]
 [0.        ]
```

```
[0.        ]
[0.        ]
[0.        ]
[0.        ]
[0.        ]]
```

[ ]:
```python
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)
```

[ ]:
```python
print(cosine_sim(TQ[:, 0], TD[:, 0])) #query & doc1
print(cosine_sim(TQ[:, 0], TD[:, 1])) #query & doc2
print(cosine_sim(TQ[:, 0], TD[:, 2])) #query & doc3
```

```
0.414904809442661
0.0
0.10856998991379904
```

Berdasarkan hasil pengukuran kemiripan tersebut, urutkan dokumen dari skor tertinggi. Dokumen mana yang paling mirip dengan query tersebut? Kemudian lakukan analisis serupa untuk query berikut. 1. sistem sentimen berita 2. analisis jadwal universitas

Anda dapat menyimpan skor kemiripan tersebut dalam suatu list dan mengambil k skor teratas untuk suatu query dengan kode berikut.

Exact Top K Document

[ ]:
```python
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
```

Misalkan k=3, maka:

[ ]:
```
top_3 = exact_top_k(doc_dict, TD, TQ[:, 0], 3)
print(top_3)
```

```
{'doc7': 0.7689768599816609, 'doc1': 0.414904809442661, 'doc4':
0.35626622628022314}
```

1.0.3 B. Inexact Top K Document Retrieval

Index Elimination Untuk mendapatkan top k dokumen dengan index elimination, salah satu cara sederhananya adalah menghitung skor kemiripan pada dokumen yang minimal memiliki satu term yang cocok dengan query. Contohnya, untuk query ”sistem informasi statistik”, hanya dokumen 1, 3, 4, 5, 6, 7 dan dokumen 8 saja yang dihitung skornya

[ ]:
```
def index_elim_simple(query, doc_dict):
    remove_list =[]
    for doc_id,doc in doc_dict.items():
        n = 0
        for word in tokenisasi(query):
            if stemming(word) in doc:
                n = n+1
        if n==0:
            remove_list.append(doc_id)
    for key in remove_list:
        del doc_dict[key]
    return doc_dict
```

Dokumen yang akan dihitung skornya dapat dieliminasi dengan memanggil fungsi di atas.

Selain itu, term pada query yang digunakan untuk mengeliminasi dokumen juga dapat dikurangi dengan hanya menggunakan term dengan nilai idf yang besar, atau dengan batas nilai idf tertentu. Term pada query dapat dieliminasi dengan fungsi berikut.

[ ]:
```
query = "sistem informasi statistik"
doc_dict = index_elim_simple(query, doc_dict)
print(doc_dict)
```

```
{'doc1': 'kembang sistem informasi jadwal', 'doc3': 'analisis sistem input
output', 'doc4': 'kembang sistem informasi akademik universitas', 'doc5':
'kembang sistem cari berita ekonomi', 'doc6': 'analisis sistem neraca nasional',
'doc7': 'kembang sistem informasi layan statistik', 'doc8': 'kembang sistem cari
skripsi di universitas'}
```

[ ]:
```
def elim_query(query, idf_dict, idf_score):
    for term in tokenisasi(query):
        if idf_dict[stemming(term)]<idf_score:
            query = query.replace(term+" ", "")
            query = query.replace(term, "")
    return query
```

Misalnya digunakan idf_score = 1.5 sebagai threshold.

[ ]:
```
query = "sistem informasi statistik"
query = elim_query(query, idf, 1.5)
print(query)
```

```
informasi statistik
```

Champion List Untuk setiap term pada vocabulary, hanya r dokumen dengan weight tertinggi saja yang dimasukkan ke dalam champion list. Hal ini berbeda dengan inverted index atau posting list yang berisi daftar seluruh dokumen dimana term tersebut berada.

[ ]:
```
def create_championlist(inverted_index, tf_idf, r):
    champion_list = {}
    for term in inverted_index.keys():
        weight_scores = {}
        for doc_id,tf in tf_idf.items():
            if tf_idf[doc_id][term]!=0:
                weight_scores[doc_id] = tf_idf[doc_id][term]
    sorted_value = OrderedDict(sorted(weight_scores.items(), key=lambda x:␣
 ↪x[1],reverse = True))
    top_r = {j: sorted_value[j] for j in list(sorted_value)[:r]}
    champion_list[term]=list(top_r.keys())
    return champion_list
```

Kemudian panggil fungsi di atas untuk mendapatkan champion list untuk r tertentu, misalnya r=2. Bandingkan isi champion list dan inverted index yang telah dibuat sebelumnya.

[ ]:
```
r=2
create_championlist(inverted_index, tf_idf, r)
```

[ ]:
```
{'klasifikasi': ['doc10']}
```

1.0.4 Penugasan

Buat fungsi main untuk menampilkan 3 list dokumen yang terurut berdasarkan cosine similarity pada folder ”berita” dengan query ”vaksin corona jakarta”. dan optimalisasi menggunakan index elimination sederhana.

[ ]:
```
import os
import math
from spacy.lang.id import Indonesian
from spacy.lang.id.stop_words import STOP_WORDS
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

nlp = Indonesian()

path = "C:/Users/FEZA/My Drive/00. Drive PC/1.STIS/5. Semester 5/Information␣
 ↪Retrieval [IR] P/Pertemuan 4/berita"
```

```
def read_text_file(file_path):
    with open(file_path, 'r', encoding='utf-8') 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

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)

def index_elim_simple(query, doc_dict):
    remove_list =[]
    for doc_id,doc in doc_dict.items():
        n = 0
        for word in tokenisasi(query):
            if stemming(word) in doc:
                n = n+1
        if n==0:
            remove_list.append(doc_id)
    for key in remove_list:
        del doc_dict[key]
    return doc_dict
```

Kode di atas merupakan function yang dibutuhkan untuk melakukan preprocessing data dan membangun inverted index serta dictionarynya. kemudian disertakan juga function untuk menghitung cosine similarity

[ ]:
```
def main(query, path):
    inverted_index = {}
    doc_dict = {}
```

```
    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[file] = cleaned_tokens

            # Ini untuk inverted index nya
            for token in set(cleaned_tokens):  # Menghindari duplikat
                inverted_index.setdefault(token, []).append(file)

    # Index Elimination
    doc_dict = index_elim_simple(query, doc_dict)

    # Menghitung cosine similarity untuk keseluruhan dokumen
    query_vector = [1 if word in tokenisasi(query) else 0 for word in␣
 ↪inverted_index.keys()]
    cos_sim_scores = []

    for doc_id, doc in doc_dict.items():
        doc_vector = [doc.count(word) for word in inverted_index.keys()]
        cos_sim = cosine_sim(query_vector, doc_vector)
        cos_sim_scores.append((doc_id, cos_sim))

    # Mengurutkan dokument berdasarkan score cosinus
    sorted_docs = sorted(cos_sim_scores, key=lambda x: x[1], reverse=True)

    # Menampilkna top 3 dokumen teratas
    for i, (doc_id, score) in enumerate(sorted_docs[:3]):
        print(f"Rank {i+1}: {doc_id}, Cosine Similarity Score: {score}")

# Panggil fungsi main dengan query "vaksin corona jakarta"
query = "vaksin corona jakarta"
main(query, path)
```

```
Rank 1: berita3.txt, Cosine Similarity Score: 0.16942647665002367
Rank 2: berita4.txt, Cosine Similarity Score: 0.1315903389919538
Rank 3: berita2.txt, Cosine Similarity Score: 0.13067709337232916
```

Pada kode main di atas, mengambil dua parameter, yaitu query yang diinginkan dan path dari folder berita. langkah awal adalah pembangunan inverted index, kemudian dilakukan simple index eliminiation terlebih dahulu. setelah itu, masuk ke tahap penghitungan cosine similarity untuk keseluruhan dokumen dan didapatkan tiga top dokumen yang relevan dengan query berdasarkan cosine similarity