MODUL 4: TERM WEIGHTING, VECTOR SPACE MODEL, DAN UKURAN KEMIRIPAN TEKS

4.1 Deskripsi Singkat

Term frequency (tf) adalah banyaknya kemunculan suatu term di dalam corpus, sedangkan document frequency (df) adalah banyaknya kemunculan suatu term di dalam setiap dokumen. Untuk menghindari pemberian bobot yang besar ke term yang muncul di hampir semua dokumen, inverse document frequency (idf) digunakan sebagai kebalikan dari df.

Representasi sekumpulan dokumen berdasarkan term-nya dalam suatu vektor disebut dengan vector space model.

Terdapat beberapa ukuran yang dapat digunakan untuk menghitung kemiripan suatu teks, diantaranya:

1. Edit Distance: jumlah minimum point mutation yang diperlukan untuk merubah suatu string ke string yang lain. Point mutation tersebut adalah mengganti, menambah dan menghapus sebuah karakter.
2. Jaccard Similarity: ukuran kemiripan berdasarkan jumlah overlap dari suatu teks.
3. Euclidian Distance: ukuran ini menghitung jarak antar vektor dalam euclidean space. Vektor yang digunakan dalam perhitungan ini adalah vektor yang merepresentasikan teks.
4. Cosine Similarity : ukuran ini menghitung nilai cosinus sudut antar dua vektor. Vektor yang digunakan dalam perhitungan ini adalah vektor yang merepresentasikan teks.

4.2 Tujuan Praktikum

1. Dapat membuat vector space model dengan pembobotan tertentu
2. Dapat membuat fungsi untuk menghitung kemiripan suatu teks dengan edit distance, jaccard similarity, euclidian distance, dan cosine similarity

4.3 Material Praktikum

Tidak ada

4.4 Kegiatan Praktikum

A. Term Weighting

Untuk mendapatkan term weighting untuk setiap dokumen, buat terlebih dahulu fungsi berikut.

```
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
```

Fungsi di atas digunakan untuk penghitungan raw tf. Anda dapat melakukan modifikasi untuk menghitung versi tf yang lain, seperti binary tf, normalized tf dan logarithmic tf.

Kemudian, panggil fungsi tersebut dengan memasukkan `vocab` dan `doc_dict` sebagai parameter. `Vocab` adalah suatu list yang berisi kumpulan term pada corpus. Anda dapat menggunakan `inverted_index` pada praktikum sebelumnya untuk mendapatkan daftar vocabulary pada corpus. Sedangkan `doc_dict` adalah dictionary yang terdiri dari isi dokumen (kalimat atau paragraf) yang telah dilakukan preprocessing dengan `doc_id` sebagai key-nya.

```
vocab=list(inverted_index.keys())
doc_dict = {}
#clean after stemming
doc_dict['doc1'] = "kembang sistem informasi jadwal"
doc_dict['doc2'] = "kembang model analisis sentimen berita"
doc_dict['doc3'] = "analisis sistem input output"

print(termFrequencyInDoc(vocab, doc_dict))
```

Kemudian buat fungsi berikut untuk menghitung document frequency and inverse document frequency.

```
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
print(wordDocFre(vocab, doc_dict))
```

```
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
```

Anda dapat mengecek hasil penghitungan idf dengan memanggil fungsi di atas.

```
print(inverseDocFre(vocab, wordDocFre(vocab, doc_dict),
len(doc_dict)))
```

B. Vector Space Model

Buat fungsi berikut untuk membuat dictionary yang berisi tf.idf score.

```
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
```

Panggil fungsi di atas sehingga Anda dapat menghasilkan suatu term-document matrix dengan skor tf.idf.

```
tf_idf = tfidf(vocab, termFrequencyInDoc(vocab, doc_dict),
inverseDocFre(vocab, wordDocFre(vocab, doc_dict),
len(doc_dict)), doc_dict)
print(tf_idf)

# 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)
```

C. Ukuran Kemiripan Teks (Text Similarity)

1. Edit Distance

Diketahui terdapat dua dokumen, kemiripan teks dengan edit distance dapat dihitung dengan fungsi berikut.

```
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
```

Kemudian panggil fungsi di atas untuk menghitung kemiripan antara `doc1` dan `doc2` pada `doc_dict` yang digunakan pada praktikum sebelumnya, serta kemiripan antara `doc1` dan `doc3`. Analisis hasil kemiripan tersebut.

```
print(edit_distance(doc_dict['doc1'], doc_dict['doc2']))
print(edit_distance(doc_dict['doc1'], doc_dict['doc3']))
```

2. Jaccard Similarity

Diketahui terdapat dua dokumen, kemiripan teks dengan jaccard similarity dapat dihitung dengan fungsi berikut.

```
def jaccard_sim(list1, list2):
    intersection = len(list(set(list1).intersection(list2)))
    union = (len(list1) + len(list2)) - intersection
    return float(intersection) / union
```

Kemudian panggil fungsi di atas untuk menghitung kemiripan antara `doc1` dan `doc2` pada `doc_dict` yang digunakan pada praktikum sebelumnya, serta kemiripan antara `doc1` dan `doc3`. Analisis hasil kemiripan tersebut.

```
print(jaccard_sim(doc_dict['doc1'].split(" "),
doc_dict['doc2'].split(" ")))
print(jaccard_sim(doc_dict['doc1'].split(" "),
doc_dict['doc3'].split(" ")))
```

3. Euclidian Distance

Diketahui terdapat dua dokumen, kemiripan teks dengan euclidian distance dapat dihitung dengan fungsi berikut.

```
def euclidian_dist(vec1, vec2):
      # subtracting vector
      temp = vec1 - vec2

      # doing dot product
      # for finding
      # sum of the squares
      sum_sq = np.dot(temp.T, temp)

      # Doing squareroot and
      # printing Euclidean distance
      return np.sqrt(sum_sq)
```

Kemudian panggil fungsi di atas untuk menghitung kemiripan antara `doc1` dan `doc2` pada `doc_dict` yang digunakan pada praktikum sebelumnya dengan melakukan slicing pada term-document matrix, serta kemiripan antara `doc1` dan `doc3`. Analisis hasil kemiripan tersebut.

```
print(euclidian_dist(TD[:, 0], TD[:, 1])) #doc1 & doc2
print(euclidian_dist(TD[:, 0], TD[:, 2])) #doc1 & doc3
```

4. Cosine Similarity

Diketahui terdapat dua dokumen, kemiripan teks dengan cosine similarity dapat dihitung dengan fungsi berikut.

```
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)
```

Kemudian panggil fungsi di atas untuk menghitung kemiripan antara `doc1` dan `doc2` pada `doc_dict` yang digunakan pada praktikum sebelumnya dengan melakukan slicing pada term-document matrix, serta kemiripan antara `doc1` dan `doc3`. Analisis hasil kemiripan tersebut.

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

4.5 Penugasan

1. Buat vector space model dengan menggunakan sekumpulan dokumen pada folder ”berita”.
2. 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.