

MODUL 5: SCORING AND RANKING

5.1 Deskripsi Singkat

Output dari suatu sistem *information retrieval* adalah sekumpulan dokumen yang terkait dengan *query* yang dituliskan untuk memenuhi kebutuhan informasi, diurutkan berdasarkan relevansinya terhadap *query*. Oleh karena itu, penghitungan skor relevansi antara *query* dan dokumen dibutuhkan untuk membuat ranking dokumen-dokumen yang relevan. Secara umum terdapat dua jenis teknik perankingan, yaitu Exact Top K Document Retrieval dan Inexact Top K Document Retrieval.

Top k document retrieval berarti mengambil k dokumen dengan skor relevansi teratas, contohnya berdasarkan skor cosine similarity. Untuk mengurangi computing cost, agar tidak perlu menghitung seluruh skor relevansi dokumen, maka digunakan Inexact Top K Document Retrieval.

Beberapa teknik Inexact Top K Document Retrieval yang akan dibahas pada modul ini diantaranya:

1. Index elimination: hanya mempertimbangkan dokumen yang memiliki term dari *query* dengan skor idf melebihi batas nilai tertentu.
2. Champion list: menghitung terlebih dahulu untuk setiap term t , r dokumen dengan bobot tertinggi pada posting list t , dimana $r < K$. Pada saat *query*, hanya menghitung skor dokumen dari champion list.

5.2 Tujuan Praktikum

1. Dapat memahami perbedaan teknik perankingan dalam document retrieval dengan Exact dan Inexact Top K Document Retrieval

5.3 Material Praktikum

Tidak ada

5.4 Kegiatan Praktikum

Pada kegiatan praktikum ini, dokumen yang digunakan untuk ilustrasi perankingan ditambah jumlahnya menjadi 10. Gunakan fungsi tokenisasi dan stemming pada praktikum sebelumnya untuk membersihkan dokumen.

```
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)
print(doc_dict)

```

Selanjutnya Anda akan memproses dokumen pada variabel `doc_dict`. Buat vocabulary, kemudian modifikasi fungsi inverted index pada praktikum sebelumnya dengan kode berikut.

```

vocab = []
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)

```

A. Exact Top K Document Retrieval

Untuk mendapatkan dokumen yang paling sesuai untuk query tertentu, kita perlu menghitung terlebih dahulu skor kemiripan antara query dengan setiap dokumen yang ada dalam corpus. Untuk

itu, 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. Maka term frequency dapat dihitung dengan kode berikut.

```
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)
```

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)
```

Untuk menghitung kemiripan antara query dengan 3 dokumen yang ada pada corpus, gunakan kode berikut.

```
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
...
```

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.

```

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)

```

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.

```

query = "sistem informasi statistik"
doc_dict = index_elim_simple(query, doc_dict)

```

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.

```
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)
```

Perhatikan query yang dihasilkan sebelum dan setelah dilakukan eliminasi term query. Mengapa term dengan nilai idf rendah dieliminasi?

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.

Berikut fungsi yang dapat digunakan untuk membuat champion list.

```
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)
```

5.5 Penugasan

1. Buat fungsi main untuk menampilkan 3 list dokumen yang terurut berdasarkan cosine similarity pada folder "berita" dengan query "vaksin corona jakarta".
2. Lakukan efisiensi dengan menggunakan index elimination sederhana.