MODUL 5: SCORING, RANKING, dan EVALUASI IR

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.

Untuk mengevaluasi sistem information retrieval, yang perlu disiapkan adalah:

1. Koleksi dokumen
2. Query
3. Data mengenai relevance judgement, biasanya berupa kelas relevan atau non-relevan untuk setiap pasang query-dokumen.

Secara umum, evaluasi pada information retrieval terbagi dua, yaitu evaluasi untuk unranked retrieval set dan evaluasi untuk ranked retrieval set.

Beberapa ukuran evaluasi untuk unranked retrieval set yaitu:

1. Precision
2. Recall
3. Accuracy
4. F-Measure

Sedangkan beberapa ukuran evaluasi untuk ranked retrieval set diantaranya:

1. Precision-Recall Curve
2. Break-Event Point
3. Mean Average Precision
4. ROC Curve

5.2 Tujuan Praktikum

1. Dapat memahami perbedaan teknik perangkingan 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 perangkingan 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 mofifikasi 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?

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
top_2 = exact_top_k(doc_dict, TD, TQ[:, 0], 2)
```

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)
```

C. Evaluasi untuk Unranked Retrieval Set

Tabel berikut menunjukkan skor yang dihasilkan untuk query ”sistem informasi statistik” pada praktikum sebelumnya, beserta label relevance judgement.

ID	Skor	Ranking	Label Relevansi
doc_1	0.4641504133851462	2	1
doc_2	0	8	0
doc_3	0.10856998991379904	4	0
doc_4	0.35626622628022314	3	1
doc_5	0.10705617011820337	6	1
doc_6	0.10856998991379904	5	0
doc_7	0.7689768599816609	1	1
doc_8	0.08967792817935699	7	1
doc_9	0	9	0
doc_10	0	10	0

Berdasarkan informasi pada tabel di atas, dokumen sebenarnya (ground-truth) yang relevan untuk query tersebut yaitu dokumen dengan id doc_1, doc_4, doc_5, doc_7, doc_8. Kemudian, dokumen yang dihasilkan oleh sistem information retrieval misalnya adalah mengembalikan 3 dokumen teratas. Anda bisa menggunakan hasil dari fungsi `exact_top_k` pada praktikum sebelumnya yang mengembalikan variabel berikut `top_3 = {'doc7': 0.7689768599816609, 'doc1': 0.4641504133851462, 'doc4': 0.35626622628022314}`. Anda dapat menggunakan kode di bawah ini untuk mendapatkan skor precision dan recall.

```
top_3 = {'doc7': 0.7689768599816609, 'doc1':
0.4641504133851462, 'doc4': 0.35626622628022314}
rel_judgement1 = {'doc1':1, 'doc2':0, 'doc3':0, 'doc4':1,
'doc5':1, 'doc6':0, 'doc7':1, 'doc8':1, 'doc9':0, 'doc10':0}
rel_docs = []
for doc_id, rel in rel_judgement1.items():
    if rel==1:
        rel_docs.append(doc_id)

retrieved_rel_doc3 = [value for value in list(top_3.keys()) if
value in rel_docs]
prec3 = len(retrieved_rel_doc3)/len(top_3)*100
rec3 = len(retrieved_rel_doc3)/len(rel_docs)*100
fScore3 = 2 * prec3 * rec3 / (prec3 + rec3)
print(prec3, rec3, fScore3)
```

Kemudian hitung skor precision, recall, F-measure, jika dokumen yang dihasilkan oleh sistem information retrieval adalah 5 dokumen teratas.

```
top_5 = {'doc7': 0.7689768599816609, 'doc1':
0.4641504133851462, 'doc4': 0.35626622628022314, 'doc3':
0.10856998991379904, 'doc6': 0.10856998991379904}
rel_judgement1 = {'doc1':1, 'doc2':0, 'doc3':0, 'doc4':1,
'doc5':1, 'doc6':0, 'doc7':1, 'doc8':1, 'doc9':0, 'doc10':0}
rel_docs = []
for doc_id, rel in rel_judgement1.items():
    if rel==1:
        rel_docs.append(doc_id)
```

```
retrieved_rel_doc5 = [value for value in list(top_5.keys()) if
value in rel_docs]
prec5 = len(retrieved_rel_doc5)/len(top_5)*100
rec5 = len(retrieved_rel_doc5)/len(rel_docs)*100
fScore5 = 2 * prec5 * rec5 / (prec5 + rec5)
print(prec5, rec5, fScore5)
```

Analisis hasil precision, recall, F-measure kedua kasus tersebut. Apa yang dapat Anda simpulkan?

D. Evaluasi untuk Ranked Retrieval Set

Tulis kode fungsi berikut untuk menghitung average precision, recall, dan f-measure, beserta ukuran lainnya.

```
import numpy as np
def compute_prf_metrics(I, score, I_Q):
    """Compute precision, recall, F-measures and other
    evaluation metrics for document-level retrieval

    Args:
        I (np.ndarray): Array of items
        score (np.ndarray): Array containing the score values of the
times
        I_Q (np.ndarray): Array of relevant (positive) items

    Returns:
        P_Q (float): Precision
        R_Q (float): Recall
        F_Q (float): F-measures sorted by rank
        BEP (float): Break-even point
        F_max (float): Maximal F-measure
        P_average (float): Mean average
        X_Q (np.ndarray): Relevance function
        rank (np.ndarray): Array of rank values
        I_sorted (np.ndarray): Array of items sorted by rank
        rank_sorted (np.ndarray): Array of rank values sorted by rank
    """
    # Compute rank and sort documents according to rank
    K = len(I)
    index_sorted = np.flip(np.argsort(score))
    I_sorted = I[index_sorted]
    rank = np.argsort(index_sorted) + 1
    rank_sorted = np.arange(1, K+1)

    # Compute relevance function X_Q (indexing starts with zero)
    X_Q = np.isin(I_sorted, I_Q)


    # Compute precision and recall values (indexing starts with zero)
    M = len(I_Q)
    P_Q = np.cumsum(X_Q) / np.arange(1, K+1)
    R_Q = np.cumsum(X_Q) / M
```

```
    # Break-even point
    BEP = P_Q[M-1]
    # Maximal F-measure
    sum_PR = P_Q + R_Q
    sum_PR[sum_PR == 0] = 1  # Avoid division by zero
    F_Q = 2 * (P_Q * R_Q) / sum_PR
    F_max = F_Q.max()
    # Average precision
    P_average = np.sum(P_Q * X_Q) / len(I_Q)

    return P_Q, R_Q, F_Q, BEP, F_max, P_average, X_Q, rank, I_sorted,
rank_sorted
```

Kemudian panggil fungsi di atas dengan kode berikut.

```
relevance_score1 = {'doc1': 0.4641504133851462, 'doc2': 0.0,
'doc3': 0.10856998991379904, 'doc4': 0.35626622628022314,
'doc5': 0.10705617011820337, 'doc6': 0.10856998991379904,
'doc7': 0.7689768599816609, 'doc8': 0.08967792817935699,
'doc9': 0.0, 'doc10': 0.0}
I = np.array(list(relevance_score1.keys()))
score = np.array(list(relevance_score1.values()))
I_Q = np.array(['doc1', 'doc4', 'doc5', 'doc7', 'doc8'])
output = compute_prf_metrics(I, score, I_Q)
P_Q, R_Q, F_Q, BEP, F_max, P_average, X_Q, rank, I_sorted,
rank_sorted = output

# Arrange output as tables
score_sorted = np.flip(np.sort(score))
df = pd.DataFrame({'Rank': rank_sorted, 'ID': I_sorted,
                   'Score': score_sorted,
                   '$\chi_\mathcal{Q}$': X_Q,
                   'P(r)': P_Q,
                   'R(r)': R_Q,
                   'F(r)': F_Q})
print(df)

print('Break-even point = %.2f' % BEP)
print('F_max = %.2f' % F_max)
print('Average precision =', np.round(P_average, 5))
```

Lalu tambahkan fungsi untuk membuat kurva precision-recall.

```
from matplotlib import pyplot as plt

def plot_PR_curve(P_Q, R_Q, figsize=(3, 3)):
    fig, ax = plt.subplots(1, 1, figsize=figsize)
    plt.plot(R_Q, P_Q, linestyle='--', marker='o', color='k',
mfc='r')
    plt.xlim([0, 1.1])
    plt.ylim([0, 1.1])
    ax.set_aspect('equal', 'box')
    plt.title('PR curve')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
```

```
    plt.grid()
    plt.tight_layout()
    ax.plot(BEP, BEP, color='green', marker='o',
fillstyle='none', markersize=15)
    ax.set_title('PR curve')
    plt.show()
    return fig, ax
```

Panggil fungsi di atas dengan kode berikut.

```
plot_PR_curve(P_Q, R_Q, figsize=(3,3))
```

Analisis ukuran-ukuran yang dihasilkan beserta kurvanya.

5.5 Penugasan

1. Tuliskan laporan praktikum yang merangkum kegiatan praktikum yang telah Anda lakukan pada kegiatan 5.4.
2. Buat fungsi main untuk menampilkan 3 list dokumen yang terurut berdasarkan cosine similarity pada folder ”berita” dengan query ”vaksin corona jakarta”. Jelaskan kode tersebut dalam laporan praktikum.
3. Lakukan efisiensi dengan menggunakan index elimination sederhana. Jelaskan kode tersebut dalam laporan praktikum.
4. Buat fungsi main untuk mengevaluasi hasil sistem information retrieval untuk folder ”berita” dengan query ”vaksin corona jakarta” yang telah dikerjakan pada modul 5. Jelaskan kode tersebut dalam laporan praktikum.
   Note: Dokumen sebenarnya yang sesuai query berdasarkan relevance judgement yaitu berita2 dan berita3.