Ranking Retrieval

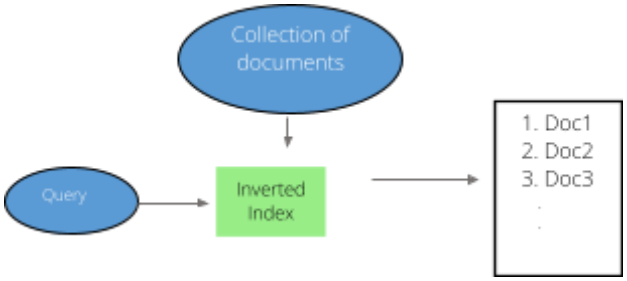
Profesor Heider Sanchez

P3. Utilice Notebook de Python y elabore una matriz de similitud de coseno entre los documentos de la colección “El Señor de los Anillos”. Debe aplicar los pesos TF-IDF.

	Doc1	Doc2	Doc3	Doc4	Doc5	Doc6
Doc1	1	0.057	0.046	0.05	0.04	0.092
Doc2	0.057	1	0.072	0.062	0.049	0.097
Doc3	0.046	0.072	1	0.045	0.067	0.69
Doc4	0.05	0.062	0.045	1	0.065	0.089
Doc5	0.04	0.049	0.067	0.065	1	0.064
Doc6	0.092	0.097	0.069	0.089	0.064	1

11/05

P4. Además**,** Implemente el índice invertido para dar soporte al modelo de **ranked retrieval** con la similitud de coseno.



1- Estructura del índice invertido en Python:

```
Python
index = {

w1 : [(doc1, tf_w1_doc1), (doc3, tf_w1_doc3),(doc4,
tf_w1_doc4),(doc10, tf_w1_doc10)],

w2 : [(doc1, tf_w2_doc1 ), (doc2, tf_w2_doc2)],

w3 : [(doc2, tf_w3_doc2), (doc3, tf_w3_doc3),(doc7,
tf_w3_doc7)],

}


idf = {
```

```
w1 : idf_w1,
w2 : idf_w2,
w3 : idf_w3,
}

length ={
doc1: norm_doc1,
doc2: norm_doc2,
doc3: norm_doc3,
...
}
```

2- Algoritmo para construir el índice:

Computing cosine scores

```
CosineScore(q)
1  float Scores[N] = 0
2  float Length[N]
3  for each query term t
4  do calculate w_{t,q} and fetch postings list for t
5     for each pair(d, tf_{t,d}) in postings list
6     do Scores[d] += w_{t,d} × w_{t,q}
7  Read the array Length
8  for each d
9  do Scores[d] = Scores[d]/Length[d]
10 return Top K components of Scores[]
```

3- Función de recuperación usando la similitud de coseno:

```python
def retrieval(self, query, k):
        self.load_index(self.index_file)
        # diccionario para el score
```

```
        score = {}
        # preprocesar la query: extraer los terminos unicos
        query_terms = self.get_terms(query)

        # calcular el tf-idf del query
        tfidf_query = self.get_tfidf(query)

        # aplicar similitud de coseno y guardarlo en el diccionario score
        for term in query_terms:
            list_pub = self.index[term]
            idf = self.idf[term]
            for (docid, tf) in list_pub:
                if docid not in score:
                    score[docid] = 0
                tfidf_doc = tf * idf
                score[docid] += tfidf_query[term] * tfidf_doc

        # norma
        for docid in self.length:
            score[docid] = score[docid] / (self.lenght[docid] *
self.get_norm(tfidf_query))

        # ordenar el score de forma descendente
        result = sorted(score.items(), key=lambda tup: tup[1], reverse=True)

        # retornamos los k documentos mas relevantes (de mayor similitud al
query)
        return result[:k]
```