Ranking Retrieval

Profesor Heider Sanchez

P1. Normalización de la longitud:

Se tiene la siguiente tabla de conteos para la consulta y dos documentos.

		Frequency (tf)		
i	term	Q	Doc1	Doc2
1	afecto	115	15	30
2	celosa	10	5	0
3	chisme	2	20	22
4	borrascoso	0	25	0

Se le pide aplicar las siguientes técnicas de scoring:

a) El score es la sumatoria del *log-frecuency weigth* de los términos comunes.

$$Score1(Q,D) = \sum_{i \in Q \cap D} qi \cdot di$$

En donde, $qi = log_{10}(1 + tf_{i,Q})$ y $di = log_{10}(1 + tf_{i,D})$

b) El score ahora se aplica con los vectores unitarios.

$$Score2(Q,D) = \sum_{i \in Q \cap D} nq_i \cdot nd_i$$

En donde, $nq_i = qi/\|\vec{Q}\|_2$, de igual forma para ndi.

$$\|\vec{Q}\|_2 = \sqrt{\sum_{i=1}^{|Q|} qi^2}$$

Llene el siguiente cuadro, analice los resultados y de una explicación de dicho comportamiento.

	(Q, Doc1)	**(Q, Doc2)**
Score1	3.9270750671960504	3.7285628573705587
Score2	0.6903164162829324	0.7819962724645979
Explicación	El query con el Score1 se parece más al Doc1. Pero con el Score2 se parece más al Doc2. Se pude observar que el Doc1 es casi el doble que el Doc2. Además de que hay pocos elementos de intersección, donde casi la mayoría son del Doc2.	

P2. TF-IDF:

Dada la siguiente tabla en donde se distribuye los pesos TF-IDF para dos documentos de la colección y para la consulta, se pide calcular el score sin normalizar (dot-product) y el score normalizado (cosine similarity) entre Q y cada documento.

Id	Término	Doc1 (TF-IDF)	Doc2 (TF-IDF)	Q (TF-IDF)
T1	Clima	1.452	0	0
T2	Biblioteca	0	2.093	1.345
T3	Universidad	2.122	0	1.453
T4	Alcalá	3.564	0	1.987
T5	España	4.123	4.245	0
T6	Libros	0	1.234	2.133
T7	Geografía	0	0	0
T8	Población	2.342	0	0
T9	Electricidad	0	0	0
T10	Ciencia	0	0	0
T11	Social	0	2.345	0
T12	Luz	1.975	0	0
T13	Unamuno	4.543	2.135	3.452
T14	Física	0	0	0
T15	Fluidos	6.134	0	0
T16	Literatura	2.234	3.456	4.234

$$DotProduct(Q, D) = \sum_{i \in Q \cap D} qi \cdot di$$

$$cosine(Q, D) = \frac{\sum_{i \in Q \cap D} qi \cdot di}{\sqrt{\sum_{i=1}^{|Q|} {q_i}^2 \times \sum_{i=1}^{|Q|} {d_i}^2}}$$

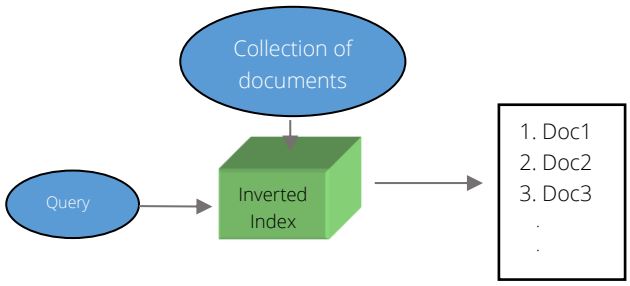
En donde *qi* es el peso tf-idf del término Ti respecto al documento Q. Lo mismo con di.

Llene el siguiente cuadro, analice los resultados y de una explicación de dicho comportamiento.

	(Q, Doc1)	**(Q, Doc2)**
DotProduct	35.306126000000006	27.449931
cosine	0.520305818475368	0.6230857580291272

El query con el **DotProduct** se parece más al Doc1. Pero con el **cosine** se parece más al Doc2.Se pude observar que el Doc1 es casi el doble que el Doc2. Además de que hay pocos elementos de intersección, donde casi la mayoría son del Doc2.

P3. Esquematice el funcionamiento del índice invertido para dar soporte al modelo de ranked retrieval.



Función de recuperación usando la similitud de coseno:

Ranked Retrieval: similitud coseno

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[]
```

Grafique el índice invertido:

```
index = {
    w1: {idf: 1.6, pub: [(2, tf2), (4, tf4), (9, tf9)]},
    w2: {idf: 1.7, pub: [(3, tf3), (4,tf4)]},
    w3: {idf: 10.8, pub: [(1, tf1), (3, tf3), (5, tf5)]}
    ...
}

length = {
    doc1: norm1,
    doc2: norm2,
    doc3: norm3
}
```

Indique los pasos de la construcción del índice:

```
def cosine(q, doc):
    return np.dot(q, doc) / (np.linalg.norm(q) * np.linalg.norm(doc))

def retrievalCosine1(collection, queryText, k):
    result = []
    queryTerms = getTerms(queryText)
```

```
7.      query = getTfIdf(queryTerms)
8.      for i in range(collection.size):
9.          doc = collection.getDocument(i, queryTerms)
10.         sim = cosine(query, doc)
11.         result.append( (doc.id, sim) )
12.     result.sort(key = lambda  tup: tup[1])
13.     return result[:k]
14.
15. def retrievalCosine2(index, queryText, k):
16.     score = {}
17.     queryTerms = getTerms(queryText)
18.     query = getTfIdf(queryTerms)
19.     for i in range(len(queryTerms)):
20.         listPub = index.get(queryTerms[i])[pub]
21.         idf = index.get(queryTerms[i])[idf]
22.         for par in listPub:
23.             score[par.docId] += (idf * par.tf) * query[i]
24.
25.     normas = index.getNorms()
26.     for docId in score:
27.         score[docId] = score[docId] / (normas[docId] * norm(query))
28.
29.     result = []
30.     for docId in score:
31.         result.append( (docId, score[docId]) )
32.     result.sort(key = lambda  tup: tup[1])
33.     return result[:k]
```