CS 5123: Cloud Computing and Distributed Systems
TF-IDF Simple Example

Following is the simple example to understand how TF-IDF works, particularly in the query search scenario. The given example is very simple and does not mimic the real world examples.

Consider there are following 3 simple documents:
Doc 1: Flash is a speedster who can travel time
Doc 2: Quicksilver is a speedster like Flash
Doc 3: Reverse Flash travel with Flash

Step – 1: Text processing

Consider we do only case folding (converting Uppercases to Lowercases) for this simple example. Actual indexing requires text processing methods (stemming, lemmatization, stop-words removal, etc.) that were discussed in the class.

Step – 2: Calculate Term Frequency (TF)

We now calculate simple term frequency of each term in each document. So we get an index like Table - 1

Table 1: Term Frequency of words

Word/Term	Doc 1	Doc 2	Doc 3
flash	1	1	2
is	1	1	0
a	1	1	0
speedster	1	1	0
who	1	0	0
can	1	0	0
time	1	0	0
travel	1	0	1
quicksilver	1	1	0
like	0	1	0
reverse	0	0	1
with	0	0	1

Step – 3: Calculate log-weighted Term Frequency

Calculate the weighted term frequency of a term using the following formula:

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus the values in Table – 1 gets updated to values in Table – 2

Table 2: Weighted Term Frequency

Word/term	Doc 1	Doc 2	Doc 3
flash	1	1	0.13
is	1	1	0
a	1	1	0
speedster	1	1	0
who	1	0	0
can	1	0	0
time	1	0	0
travel	1	0	1
quicksilver	1	1	0
like	0	1	0
reverse	0	0	1
with	0	0	1

Just the above table weights can be used for retrieving results when we get a query. But we will look at tf-idf calculation also in this example.

Step – 4: Calculate Document Frequency

Calculate the number of documents in which the word appears.

Table 3: Document Frequency

Word/term	Document Frequency
flash	3
is	2
a	2
speedster	2
who	1
can	1
time	1
travel	2
quicksilver	1
like	1
reverse	1
with	1

Step – 5: Calculate Inverse Document Frequency

With the Document Frequency of each term 't' (**df$_t$**) and total number of documents in the collection **N**, we calculate Inverse Document Frequency of each term 't' using:

$$idf_t = \log_{10} \frac{N}{(1 + df_t)}$$

Thus we get the following index:

Table 4: Inverse Document Frequency

Word/term	DF	IDF
flash	3	-0.12
is	2	0
a	2	0
speedster	2	0
who	1	0.18
can	1	0.18
time	1	0.18
travel	2	0
quicksilver	1	0.18
like	1	0.18
reverse	1	0.18
with	1	0.18

Step – 6: Calculate TF-IDF

With Weighted Term Frequency values from Table – 2 and Inverse Document Frequency from Table – 4, we can calculate TF-IDF using:

$$tf - idf_{t,d} = \left(1 + \log_{10} tf_{t,d}\right).\, log\, \frac{N}{(1 + df_t)}$$

which result in tf-idf index as given in Table – 5, where each document is represented as a vector of tf-idf weights.

Table 5: TF-IDF weights

Word/term	Doc 1	Doc 2	Doc 3
flash	-0.12	-0.12	-0.02
is	0	0	0
a	0	0	0
speedster	0	0	0
who	0.18	0	0
can	0.18	0	0
time	0.18	0	0
travel	0	0	0
quicksilver	0.18	0.18	0
like	0	0.18	0
reverse	0	0	0.18
with	0	0	0.18

Step – 7: Query Processing

Sample Query: **Can Flash travel time**

As discussed in the class we have to treat the query also as a document. Also, we discussed of using different weighting schemes for documents (**logarithmic TF, NO IDF, Cosine normalization**) and queries (**logarithmic TF, IDF, NO normalization**). To make this example simplistic, we follow **logarithmic TF, TF-IDF, Cosine normalization (ltc)** for both documents and queries.

Similar to other documents in the collection, we want to process text (*case folding* in this example) in queries also.

Processed query: can flash travel time ➔ can, flash, travel, time

The naïve approach is to construct high dimensional vectors with all words in vocabulary for other documents as given in Table – 5. We are going to use a simple approach to avoid using this high dimensional vectors. We will construct a vector of size = no. of words in the processed query.

Thus, for each word in the processed query we calculate TF, IDF, tf-idf, and Cosine Norm. TF can be calculated only for the query, DF and IDF can be retrieved from Table – 5. tf-idf is the final weight of the given word from TF and IDF. We normalize each weight using cosine normalization (divide each vector component by Euclidean length):

Euclidean length of the given query = $\sqrt{(0.18)^2 + (-0.12)^2 + (0.18)^2}$ = 0.28
Therefore, 0.18/0.28 = 0.64 and -0.12/0.28 = -0.43

Table 6: Weighting the processed query

Term	TF	Weighted TF	DF	IDF	tf-idf	Cosine Norm (c_t)
can	1	1	1	0.18	0.18	0.64
flash	1	1	3	-0.12	-0.12	-0.43
travel	1	1	2	0	0	0
time	1	1	1	0.18	0.18	0.64

Similarly, we can calculate Euclidean length and cosine normalized values of tf-idf of *query words* in Doc 1, Doc 2, and Doc 3 from their respective vectors as we are looking at **ltc** for indexing also. Euclidean length of

Doc 1 = 0.38
Doc 2 = 0.28
Doc 3 = 0.26

Once we have normalized weights of words in documents, we can calculate the final weight of the document for the given query by taking product of normalized weights of terms in the query (**c_t**) and cosine normalized weights of TF-IDF in the document (**$CTFIDF_{d1}$/ $CTFIDF_{d2}$/$CTFIDF_{d3}$**), and taking sum of all products. Following tables give weight of each document for the given query.

Doc 1

Term	**tf-idf**	**Cosine Norm ($CTFIDF_{d1}$)**	**Product = c_t * $CTFIDF_{d1}$**
can	0.18	0.47	0.30
flash	-0.12	-0.32	0.14
travel	0	0	0
time	0.18	0.47	0.30

Total score: 0.74

Doc 2

Term	**tf-idf**	**Cosine Norm ($CTFIDF_{d2}$)**	**Product = c_t * $CTFIDF_{d2}$**
can	0	0	0
flash	-0.12	-0.43	0.19
travel	0	0	0
time	0	0	0

Total score: 0.19

Doc 3

Term	**tf-idf**	**Cosine Norm ($CTFIDF_{d3}$)**	**Product = c_t * $CTFIDF_{d3}$**
can	0	0	0
flash	-0.02	-0.08	0.03
travel	0	0	0
time	0	0	0

Total score: 0.03

Rank of documents: Doc 1 – 1, Doc 2 – 2, Doc 3 – 3

Exercise queries you can try:

1. Can Quicksilver travel like Flash
2. Is Reverse Flash a Flash
3. What will be document ranks for the above queries if we use different weighting schemes for documents and queries?