Report

April 28, 2020

Multimedia Retrieval and Content-Based Search

Assignment 1 - Report

Oliver Schweiger, 01220194

Task 1

What are the shortcomings of the raw term frequency calculation and how do they affect a retrieval system?

Answer: Though raw frequency of a term is serviceable as an indicator of document relevance, there are just too many things not considered by it. Primarily, you want to have at least some weighting that prevents the tf-value from growing too large and thus distorting the results when comparing large documents (common words like 'a' or 'the'): the logarithm used in log frequency weighting does exactly that. Also, you might want to have a weight related to term relevance - terms that appear often should not be weighted as much as rare terms (independent from raw tf). The idf-value does just that. Other things like term ordering are also lost when using just raw tf.

Evaluate the following statements:

Using a tf-idf weighting scheme, the weight of a term t in a document d is…. …lowest when t occurs many times in many documents The term occurring in many documents is an indicator of it being less relevant, but it still occurs many times per document and it doesn't occur in every document so its idf will be low to moderate.

…highest when t occurs very often in all documents False, if a term occurs in all documents, as can be show in the operation below, its weight will be zero.

[14]:
```
import math

N = 5; df = 5;
math.log10( N / df )
```

[14]:
```
0.0
```

…highest when t occurs many times in few documents True, a high tf and high inverse df (meaning a low df) is an indicator of high relevance.

…growing when the occurrence of t increases proportionally with the amount of documents t occurs in (assuming the collection size stays the same): Not generally. It depends on the Collection Size. For low df values (with respect to the collection size) there is a growth to be observed, but after

a certain value the idf part of the equation gets too close to zero for the tf-part to counteract it, resulting in a decrease. This behavior is shown in the cell below.

[9]:
```
import math

N = 1000 #collection size
i = 1
while i<10:
    tf = i * 2;
    df = i * 3;
    tf_idf = (1 + math.log10(tf)) * math.log10( N / df)
    print('tf-idf: %.3f' %tf_idf)
    i += 1
```

```
tf-idf: 3.282
tf-idf: 3.560
tf-idf: 3.638
tf-idf: 3.655
tf-idf: 3.648
tf-idf: 3.628
tf-idf: 3.601
tf-idf: 3.570
tf-idf: 3.538
```

Why is a document-level statistic for term weighting to be preferred over a collection-wide statistic? Using the df over the cf (collection frequency) for the calculation of the tf-idf is better because a low df value is a better indicator for term rarity than the cf. A term might appear often in (domain specific) document but not in general. This results in low df's but still way higher cf's, making the df's inverse higher, thus assigning the more rare term a higher relevance. This way, we can also decrease the influence of possible fake sites that try to manipulate the tf-idfs. The df value makes their (fake) contributions matter much less than the cf would. Of course, for queries it also makes more sense to use the doc-related tf as a basis since we want our queries to be answered document based and not collection based.

Task 3

Think about the relationship between information need and query – what is a query?

A query is a textual abstraction that describes my information needs and which then gets processed within the IR System to deliver a result that best fulfills this need. Neither query nor query-results will be 100% precise, but it's important to get close, by having the user formulate a precise query and the IR system offer high precision/recall.

Critically think about the differences between the Boolean retrieval model and a ranked retrieval model building upon e.g. the vector space model.

For some Use Cases/Systems a boolean model might still be a good solution, but in the context of web search engines, the ranked model has prevailed since there is often not "the" right answer to a query but multiple ones that might be interesting to the user to different degrees. For other systems (for example law) this might not be the case.

Explain why a query for the term “a” produces the given result

[13]:
```
%run ../Task3/Implementation.ipynb

table_tfidf = calculate_tfidf('../collection2.txt')
print(perform_query(table_tfidf, 'a'), '\n')

print(perform_query(table_tfidf, 'b'), '\n')

print(perform_query(table_tfidf, 'c'), '\n')

print(perform_query(table_tfidf, 'b c'), '\n')

print(perform_query(table_tfidf, 'a b c d'), '\n')
```

```
[(5, 1.0), (6, 1.0), (1, 0.9449), (4, 0.5), (2, 0.2236), (3, 0.0)]

[(2, 0.6708), (4, 0.5), (1, 0.189), (3, 0.1562), (5, 0.0), (6, 0.0)]

[(2, 0.6708), (4, 0.5), (3, 0.3123), (1, 0.189), (5, 0.0), (6, 0.0)]

[(2, 0.9487), (4, 0.7071), (3, 0.3313), (1, 0.2673), (5, 0.0), (6, 0.0)]

[(4, 1.0), (2, 0.8944), (1, 0.7559), (3, 0.7028), (5, 0.5), (6, 0.5)]
```

Doc 5 and 6 rank highest since it only consists of a’s. Also it doesn’t matter how many a’s are in the doc, since its vector representation is normed, making its length irrelevant -> (1, 0, 0) and (3, 0, 0) is the same, when normed. For the other ones: the more a’s in proportion to the rest of the document, the higher the similarity.

Why does Doc1 receive a higher rank than Doc3 when the term “b” is queried: Because in Doc 1, there are more b’s with respect to other entries (1/8 is ‘b’), where as 1/9 is ‘b’ for Doc 3.

Why is the rank of Doc3 lower than the rank of Doc2 although Doc2 contains more “c” terms when a query for the term “c” is issued: Doc2 contains more c’s and is also shorter, which makes it higher ranked.

Why does Doc2 receive a higher rank than Doc4 when the query contains the terms “b c”: Half of the Doc4 is made out of “b c”, but 6/8 of Doc2 is made out of “b c”, making it the better match.

Consider the query “a b c d” – which document receives the highest rank and why: Doc4, since it only consists of these terms. It ranks higher than 1, 2 and 3 because the proportion of the terms within the query fits better than for the other docs.

Critically think about what do these numbers represent and how do they express similarity? The higher the number, the higher the similarity. As described above, tf offers a decent but not perfect measure for similarity, since it doesn’t account for term rarity. It also doesn’t check for ordering. The cosine similarity also has the property of just comparing the vectors angles. This is the reason why two docs “a” and “a a” are considered identical - their vectors point in the exact same direction.

Task 4:

Outline how the result of a PageRank calculation can be integrated into the calculation process of a retrieval (similarity) score

Easiest way to do it would be to just multiply a page's rank onto the similarity score, making popular pages rise in query-similarity (more likely to be seen) and less popular ones rank lower (even though they might have had a higher base similarity in terms of tf-idf)

Think about the nature of the PageRank and ascertain whether it is a query-dependent or query-independent measure?

Page Ranks are calculated independent from a single query (on a search engines web server for example) a-priori. So in principle, they are independent from a specific query. Page Rank, however, is calculated based on web page's visit rates, meaning that there could very well a high-level relationship between queries and Page Rank, because users tend to visit pages that conform to their queries, in turn making their page rank higher, meaning that they do indeed indirectly increase page's page ranks through their queries.

[ ]: