

Shane Cincotta

Hui Fang

Search and Data Mining

04/20/20

### Building Indexes

Twitter is one of the most popular social media websites in use as of 2020, this inherently leads to mass quantities of tweets per day. Without a proper index, our search engine would parse through every single document in the corpus. The corpus could contain millions of documents and as such could take considerable amounts of time to comb through. This is why indexing is so important to IR, proper indexing will reduce the overall amount of documents which are parsed through, drastically reducing retrieval time. For my index, I need to take into account several factors such as: the index data structure, index size and lookup speed.

To begin, I first need a data structure to hold the data of the indices. When creating my index I need to think of the types of data that will be in any given index. Since we're trying to find relevant tweets, we know that each tweet is at most 280 characters, this means that the indices are not holding that much information, i.e there isn't a lot to parse through (max 280 chars). For this reason, I will be using an inverted index to store and rank the documents. As shown in figure 1, an inverted index stores a list of the documents containing each query word. Because of the static max tweet size, we can also preallocate specific amounts of memory for each index (since we know the index doesn't have to be larger than the longest possible tweet). One drawback of this approach occurs with shorter tweets. If we always assume that our tweet is a worst case scenario (280 characters), but the tweet is short, say for example, 1 char, then we have allocated room for an extra 279 chars, even though we won't be using them.

Word	Document Number
hello	Document 1, 3, 5
world	Document 2, 4

**Figure 1:** Example of inverted index

An inverted index can use direct access to find the document associated with each query word, this is especially beneficial in our case because the amount of query words will be low since there is a character limit. If our document was known to be long (much longer than 280 characters) then an inverted index might not be the best choice as the direct access time would be substantially greater because there are more words to keep track of.

Despite the advantages of an inverted index, we still need more information. An inverted index only tells us whether a word exists in a document, but not how often that word occurs within a document. To compensate for this, we will need to add another parameter to the index, frequency. This will keep track of the amount of time a word was used in the document. My inverted index will now look like figure 2, with the frequency of the word in curly braces.

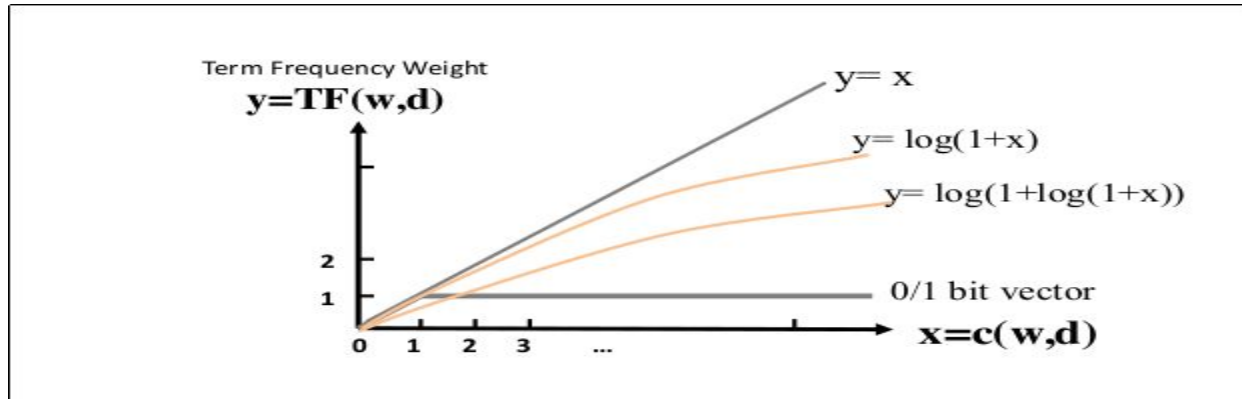
Word	Document Number
hello	Document 1{5}, 3{7}, 5{1}
world	Document 2{9}, 4{8}

**Figure 2:** Example of modified inverted index

### Ranking Tweets

Now that we have built an index which contains all the relevant documents, we need to rank these documents. In our scenario. We're asked to find all the relevant tweets, not just the most relevant. It's for this reason that I will be comparing each tweet to a threshold value. If the tweets relevance is greater than that value, I will deem the tweet as relevant and include it in the results. For ranking my tweets, I will be considering term frequency, document frequency and popular terms.

For term frequency, I will make the assumption that the relevance score of a document is related to the occurrence of a query term. To account for term frequency in my ranking, I will be using the formula,  $1 + \ln(1 + \ln(c(t, D)))$ , where  $c(t, D)$  is the count of the term in the document. The purpose of the dual natural logs is to create a limit as to how much the term frequency can contribute to the overall relevance score. If the relationship was linear, the score could be artificially inflated with an abnormally high term frequency, even though the document isn't relevant. Figure 3 shows how the use of natural logs limits the maximum score of a document with high term frequency counts.



**Figure 3:** How logs affect relevance score with high term frequency

Hui Fang lecture slide 05-vs.pdf

For document frequency, I will be operating under the assumption that the importance of a term is related to the document frequency of that term, i.e the amount of documents which contain that term. I can use this assumption to then penalize popular terms within a document with IDF weighting. To do this, I will be using the formula  $\ln((N+1) / df(t))$  where  $df$  is the total number of documents containing the document frequency and  $N$  is the total number of documents in the collection. Again a natural log is used to limit the maximum score.

For term frequency, I will simply be using  $c(t, Q)$ , which is the count of the terms in the query.

Finally, my relevance score can be thought of as

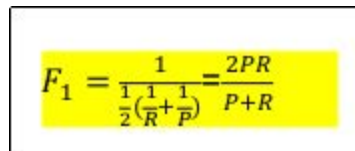
$$S(Q, D) = \sum_{t \in Q, D} \frac{1}{1 + \ln(1 + \ln(c(t, D)))} * c(t, Q) * \ln((N + 1)/df(t)).$$

For my score, I didn't take into account any length normalization. The reason for this is that most tweets are going to be roughly the same size (again, because of the character limit). Since each tweet is going to be roughly the same size ( $0 < \text{size} < 281$ ), normalizing the length would have minimal effect. Although most of the time normalizing the length won't change the score, it could affect the score as the tweet length gets closer to the extreme values.

### Evaluating Results

After my ranker is built, I can begin to use my program to test in real time. To evaluate the results, I will primarily be considering two criteria: effectiveness and efficiency.

To test effectiveness, I will be considering the accuracy of the search results. To determine the accuracy, I will be specifically looking at precision and recall. Precision is the ratio of relevant documents retrieved to the total documents retrieved. Recall is the ratio of relevant documents retrieved to the total amount of documents retrieved. I can combine both precision and recall together using the F1-Measure. The F1-Measure is a harmonic average of precision and recall, that is, an average of their inverses. The F1-Measure heavily penalizes low values of either precision or recall (Hui Fang lecture slide *08-evaluation.pdf*).


$$F_1 = \frac{1}{\frac{1}{2} \left( \frac{1}{P} + \frac{1}{R} \right)} = \frac{2PR}{P + R}$$

**Figure 4:** F1-Measure formula

The score returned from this formula will be highest with both precision and recall are at their highest and also closer together. I will be determining if a document is relevant by reading it manually.

To test efficiency, I will be considering the time taken to return results to the user. To accomplish this, I will create a testbench which tracks the time taken to return the results. I will then print out this time along with other relevant information such as the amount of results, much in the same way that Google does. The lower the time takes per document, the higher the efficiency score will be.