Text retrieval

Kristian Wichmann

March 7, 2017

1 Information retrieval vs. text retrieval

Information retrieval (IR) is the process of obtaining information from a source of relevant data. Very often, we will think of digital systems, and hence the amounts of data involved are very large. In general, we could be looking for any kind of information, like pictures, audio, or video.

Text retrieval (TR) is the subset of IR which deals with text. The most iconic example is web search based on a text query, like the services provided by search engines such as Google and Bing.

2 Push and pull modes of text access

Text retrieval happens in two major modes: push and pull.

2.1 Push mode

In push mode, the text is suggested (pushed) to the user based on previous interactions. This is what typically happens in a recommender system. For instance, Netflix will suggest movies and series based on your view history.

2.2 Pull mode

In pull mode, the user takes the initiative to get (pull) the text. This may be further subdivided:

- *Querying* - Here, the user knows what to look for, at enters a keyword text: a query. This works well when the user knows what he/she is looking for.

- *Browsing* - Here, the user navigates through the data, following a path enabled by its structure. This works well when the user wants to explore the data.

We will focus on querying in this document.

3 Terminology

Here we describe the various ingredients needed to make a *retrieval model*:

- We work with a *vocabulary* $V = \{w_1, \cdots, w_N\}$ of words.
- A *query* q may be written $q = (q_1, \cdots, q_m)$, where $q_i \in V$.
- A *document* can be written $d_i = (d_{i1}, \cdots, d_{im_i})$.
- A *collection* of documents is $C = \{d_1, \cdots, d_M\}$.

3.1 Term frequency-inverse document frequency

A given vocabulary word $w \in V$ can be described in terms of its appearence in a document/collection. Two common measures are *term frequency* (tf) and *inverse document frequency* (idf).

3.1.1 Term frequency

Term frequency directly describes the occurrence of w in d. There's many variations of this. The simplest two are:

- 0 or 1 depending on whether w is present or not.
- Raw frequency $\text{tf}(f, d)$. A count of the number of occurences.

Later we will look at transformations of tf.

3.1.2 Inverse document frequency

The inverse document frequency describes how unusual a word w is in a collection C. If n_{wC} is the number of documents in C which contains w, the idf is defined as:

$$\text{idf}(w, C) = \log \frac{N+1}{n_{wC}} \tag{3.1}$$

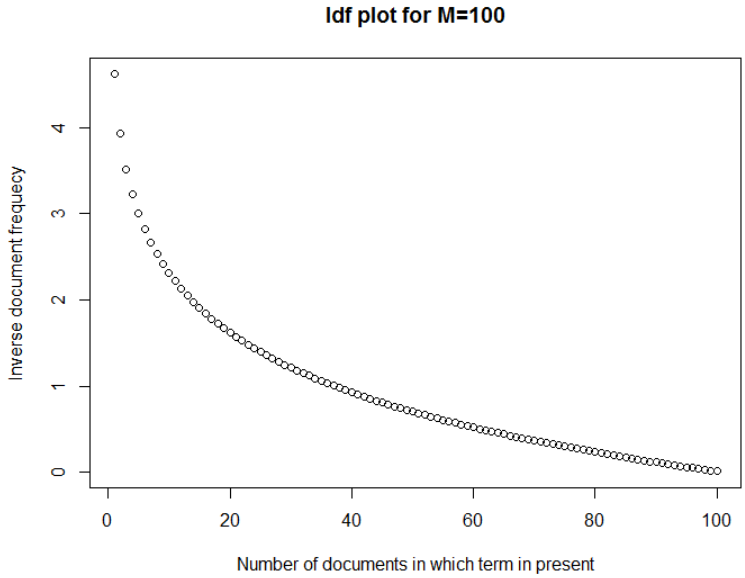

Figure 1: An example idf plot.

To avoid division by zero for non-occuring w, 1 is sometimes added to the denominator. A number of different weighting schemes for idf exists, but this is the one we will primarily use here.

Figure 1 shows an example of how idf decreases with higher document occurrences.

4 The text retrieval problem

Given a query q, we wish to extract the set of *relevant documents* $R(q) \subseteq C$ for the query.

In practice, all we can hope for is an approximation of the relevant documents: $R'(q)$.

5 Strategies

5.1 Document selection strategies

One way to solve the text retrieval system is to build a binary classifier f, which given a document d and a query q returns either 0 or 1, depending on whether or not $d \in R'(q)$:

$$R'(q) = \{d \in C | f(d, q) = 1\} \tag{5.1}$$

Of course, any way to choose $R'(q)$ is technically a binary classifier, but the idea is that f decides the *absolute relevance* of the document - there is no further nuance beyond "yes" or "no".

With this strategy, there's 2^N possible different outcomes, since each document is either relevant or not.

5.2 Document ranking

Instead, the function f might have a continuum of real values instead of just $\{0, 1\}$. Then we might choose $R'(q)$ based on a *cutoff* θ:

$$R'(q) = \{d \in C | f(d, q) > \theta\} \tag{5.2}$$

Here f is more nuanced, and decides what is called the *relative relevance* of the document. A list of documents sorted by decreasing relevance could be constructed, and θ decides where to stop the list. Or rather, if the user browses such a list, θ is decided by the user.

There's $N!$ different possible outcomes with this strategy, since each ordering of documents is distinct.

5.2.1 The probability ranking principle

The document ranking strategy is (under certain conditions) guaranteed by the *probability ranking principle* (PRP) to be of optimal utility to the user.

6 Vector space model

One approach to building a selection function is to use a *vector space model*. This is a *similarity-based* model, as it tries to give a measure of how similar the query and a document is.

6.1 The bag of words model

In the *bag of words* model (BOW), we disregard the ordering of the words in a document, and simply see them as an unordered list. Clearly, some information is lost in this simplification, but sometimes it will still yield useful results. The vector space model uses BOW.

6.2 Types of vectors - simplest approach

I each of the following cases, the relevant vector space for the model has a dimension of the size of the vocabulary V. In other words is has N dimensions.

6.2.1 Bit vectors

Here, each coordinate can only take on the values 0 or 1, indicating whether or not the word w_i is present in the query/document or not.

6.2.2 Frequency vector

Going a bit further than bit vectors, here coordinate i represents the number of times w_i appears in the document.

6.3 Similarity measure

No matter which model is used, the simplest similarity measure between a query and a document (or between two documents) is the dot product between the two:

$$\text{similarity} = q \cdot d \tag{6.1}$$

For bit vectors this is equal to the distinct number of query terms matched in the document. For frequency vectors, it is the total count of occurrences of any query term in the document.

6.4 Including idf: tf-idf weight vector

The vector types mentioned above doesn't take into account how commonly used a word is. For instance, occurences of the words "the", "a" or "is" have the same impact on the similarity score than more specific terms. Some kind of *weighting* is needed.

One approach, it to use the inverse document frequency, since it will be low for common words. In other words, the vector coordinate for the word w should be the frequency of the words in d times the idf for the word in the relevant collection C. In other words it is:

$$\text{tf}(w, d) \cdot \text{idf}(w, C) \tag{6.2}$$

Because of this, it is known as the *tf-idf weight vector*.

6.5 Tf transformation: BM25

To simply have the raw tf count in the formula above is also problematic. It rewards simple repetition of a given term too much; the first occurence of a search term is valued as highly as the fiftieth. In order to remedy this, we need to perform some transformation t of tf.

We've already seen one such transform when we considered the binary representation of whether a term is present or not. Here, the transformation is:

$$t(\text{tf}) = \begin{cases} 0 & \text{when tf} = 0 \\ 1 & \text{otherwise} \end{cases} \tag{6.3}$$

For this purpose however, it is too drastic. A better looking option is a logarithmic transformation:

$$t(\text{tf}) = \log(1 + \text{tf}) \tag{6.4}$$

Or even a nested logarithm:

$$t(\text{tf}) = \log(1 + \log(1 + \text{tf})) \tag{6.5}$$

However, it turns out that in many cases what works best is a transformation of the following type:

$$t(\text{tf}) = \frac{(k+1)\text{tf}}{\text{tf} + k} \tag{6.6}$$

This is known as a *BM25 transformation*. Figure 2 shows this transformation for various k values.

6.6 Taking document length into account

Longer documents will usually include more words. Therefore, they will tend to get higher vector coordinates simply because of length. We may attempt to rememdy this by introducing a length *normalization*.

We will require this normalization function to be 1 at average document length for C, a number we will denoted avdl. This may be accomplished by the following linear function:

$$n(l) = 1 - b + b\frac{l}{\text{avdl}} \tag{6.7}$$

For $b > 0$ this function grows larger for large document sizes, which is what we're looking for, since we want to divide by $n(d)$.

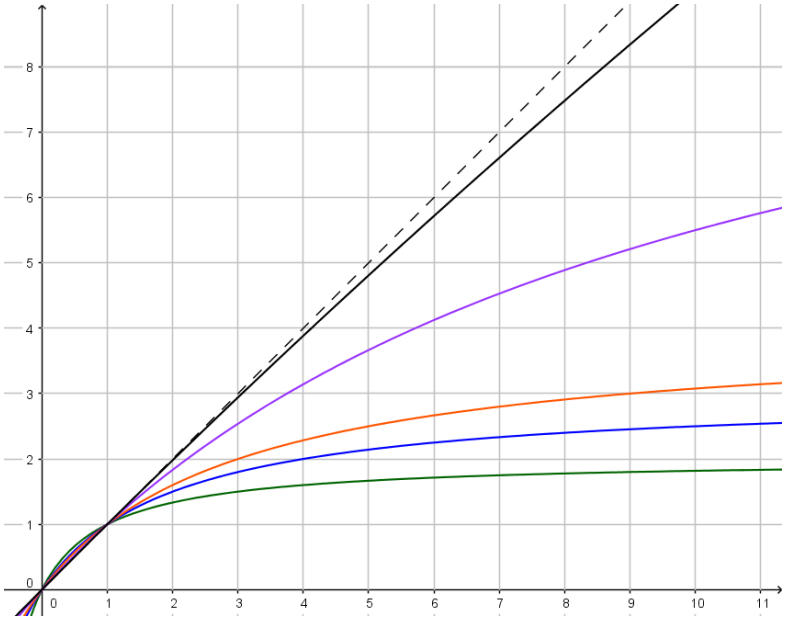

Figure 2: The BM25 transformation for $k = 2, 3, 4, 10$ and 100 respectively. The asymptote $x = k + 1$ is not clear from this plot.

7 Term-Frequency Matrix

The chosen weights can be collected in what is known as the *term-frequency matrix* (TFM). The name refers to frequency vectors, but it can be used for any weighting scheme.

In the TFM, words from the vocabulary are in the x-direction, while documents from the corpus are in the y-direction. An example will make this clearer.

7.1 Example of TFM

Consider a corpus consisting of the following three text files:

1. "this is good"
2. "this is bad"
3. "this is very very bad"

Here, the vocabulary consists of the following five words: this, is, good, bad and very. The corresponding TFM (using the frequency vector weight scheme) is shown in table 7.1.

Docoment/Word	**this**	**is**	**good**	**bad**	**very**
1	1	1	1	0	0
2	1	1	0	1	0
3	1	1	0	1	2

Table 1: TFM example

7.2 Abstract form

Note how in the example each row is a document vector. In other words, the TFM could abstractly be written as:

$$\begin{pmatrix} — & d_1^T & — \\ \vdots & \vdots & \vdots \\ — & d_M^T & — \end{pmatrix} = \begin{pmatrix} | & \cdots & | \\ d_1 & \cdots & d_M \\ | & \cdots & | \end{pmatrix}^T \tag{7.1}$$

Here d_i is the column vector for document i.

7.3 Inverted index

The *inverted index* form of the TDM, is basically the same data, but viewed columnwise: For instance, from the fourth column in the example above, we can immediately infer that the word "bad" appears once in documents 2 and 3. This is much quicker than scanning through the entire corpus for the word.

Since in general, the TFM matrix will be rather sparse, different compression schemes for the inverted index are often used in practice.

8 Document classification: Naive Bayes

Assume the corpus C is divided into k of disjoint categories $C_1, C_2, \ldots, C_k$. We now want to develop a way to evaluate in which category a new document d should be classified.

For the *naive Bayes* classifier, we assume that the occurence of a word w_i in a document in C_j is stochastic, and happens independently of the occurence of other words. It is naive in the sense, that this is obviously not true. But it may still yield good results.

8.1 Theory

The new document d can be written $d = (w_1, \ldots, w_n)$, where all the words w_i are in the vocabulary V. Now, consider the probability that a random

document from C_j contains all the words of d. According to the assumptions of naive Bayes, this is simply a product of the probabilities of each individual word being in a random document from C_j:

$$p(d|C_j) = \prod_{i=1}^{n} p(w_i|C_j) \tag{8.1}$$

Of course, the probability we're interested in is $p(C_j|d)$: The probability of being in C_j assuming that the words in d are all present. To calculate this, *Bayes' theorem* is used. The general version is:

$$p(A|B) \propto p(B|A)p(A) \tag{8.2}$$

Since we're only interested in relative probabilities (i.e. which j is most likely?), the normalization constant is irrelevant here. In this case:

$$p(C_j|d) \propto p(d|C_j)p(C_j) = p(C_j) \prod_{i=1}^{n} p(w_i|C_j) \tag{8.3}$$