Determine Word Relevance in Document Queries Using TF-IDF

Ashok Koujalagi

Computer Science (Network Engineering) Document Query Retrieval Concept

ABSTRACT

In this paper, we examine the results of applying Term Frequency Inverse Document Frequency (TF-IDF) to determine what words in a corpus of documents might be more favorable to use in a query. As the term implies, TF-IDF calculates values for each word in a document through an inverse proportion of the frequency of the word in a particular document to the percentage of documents the word appears in. Words with high TF-IDF numbers imply a strong relationship with the document they appear in, suggesting that if that word were to appear in a query, the document could be of interest to the user. We provide evidence that this simple algorithm efficiently categorizes relevant words that can enhance query retrieval.

Introduction

Before proceeding in depth into our experiments, it is useful to describe the nature of the query retrieval problem for a corpus of documents and the different approaches used to solve it, including TF-IDF.

Query Retrieval Problem

The task of retrieving data from a user-defined query has become so common and natural in recent years that some might not give it a second thought. However, this growing use of query retrieval warrants continued research and enhancements to generate better solutions to the problem.

Informally, query retrieval can be described as the task of searching a collection of data, be that text documents, databases, networks, etc., for specific instances of that data. First, we will limit ourselves to searching a collection of English documents. The refined problem then becomes the task of searching this corpus for documents that the query retrieval system considers relevant to what the user entered as the query.

Let us describe this problem more formally. We have a set of documents $D$, with the user entering a query $q = w_1, w_2, ..., w_n$ for a sequence of words $w_i$. Then we wish to return a subset $D' \subseteq D$ such that for each $d \in D'$, we maximize the following probability:

$$P(d | q, D) (1)$$

(Berger & Lafferty, 1999). As the above notation suggests, numerous approaches to this problem involve probability and statistics, while others propose vector-based models to enhance the retrieval.

Algorithms for Ad-Hoc Retrieval

Let us briefly examine other approaches used for responding to queries. Intuitively, given the formal notation we present for the problem, the use of statistical methods has proven both popular and efficient in responding to the problem. (Berger & Lafferty, 1999) for example, propose a probabilistic framework that incorporates the user’s mindset at the time the query was entered to enhance their approximations. They suggest that the LSI approach can be extended to more than two languages.

The formal procedure for implementing TF-IDF has some minor differences over all its applications, but the overall approach works as follows. Given a document collection $D$, a word $w$, and an individual document $d \in D$, we calculate pronouns, and prepositions, which by themselves hold no relevant meaning in a query (unless the user explicitly wants documents containing such common words). Such common words thus receive a very low TF-IDF score, rendering them essentially negligible in the search.

Finally, suppose $fw, d$ is large and $fw, D$ is small. Then $\log |D|/fw, D$ will be rather large, and so $wd$ will likewise be large. This is the case we’re most interested in, since words with high $wd$ imply that $w$ is an important word in $d$ but not common in $D$. This $w$ term is said to have a large discriminatory power.
Therefore, when a query contains this \( w \), returning a document \( d \) where \( wd \) is large will very likely satisfy the user.

**Encoding TF-IDF**

The code for TF-IDF is elegant in its simplicity. Given a query \( q \) composed of a set of words \( wi \), we calculate \( wi, d \) for each \( wi \) for every document \( d \in D \). In the simplest way, this can be done by running through the document collection and keeping a running sum of \( fw, d \) and \( fw, D \). Once done, we can easily calculate \( wd \) according to the mathematical framework, presented before. Once all \( wi, ds \) are found, we return a set \( D \) containing documents \( d \) such that we maximize the following equation:

\[
\text{maximize } wd = \sum_{wi \in D} \log \left( \frac{|D|}{|D_w|} \right) \cdot |D_w| \cdot |D| \cdot |D_w| \cdot |D|.
\]

Either the user or the system can arbitrarily determine the size of \( D \) prior to initiating the query. Also, documents are returned in a decreasing order according to equation (3).

This is the traditional method of implementing TF-IDF. We will discuss extensions of this algorithm in later sections, along with an analysis of TF-IDF according to our own results.

\[
wd = f_w, d \times \log \left( \frac{|D|}{|D_w|} \cdot |D| \cdot |D_w| \cdot |D| \right).
\]

where \( f_w, d \) equals the number of times \( w \) appears in \( d \), \( |D| \) is the size of the corpus, and \( f_w, D \) equals the number of documents in which \( w \) appears in \( D \) (Salton & Buckley, 1988, Berger, et al, 2000). There are a few different situations that can occur here for each word, depending on the values of \( f_w, d \), \( |D| \), and \( f_w, D \), the most prominent of which we’ll examine.

**Experiment**

**Data Collection and Formatting**

We tested our TF-IDF implementation on a collection of 1400 documents from the LDC’s United Nations Parallel Text Corpus. These documents were gathered arbitrarily from a larger collection of documents from the UN’s 1988 database. The documents were encoded with the SGML text format, so we decided to leave in the formatting tags to account for noisy data and to test the robustness of TF-IDF. We simulated more noise by enforcing case-sensitivity. Due to certain constraints, we had to limit the number of queries used to perform information retrieval on to 86. We calculate TF-IDF weights for these queries according to equation (3), and then return the first 100 documents that maximize equation (3). The returned documents were returned in descending order, with documents with higher weight sums appearing first. To compare our results, we also performed in parallel the brute force (and rather naive) method of performing query retrieval based only on the term \( f_w \). Naturally, this latter method is intuitively flawed for the larger problem of query retrieval, since this approach would simply return documents where non-relevant words appear most (i.e. long documents with plenty of articles and prepositions that might not have any relevance to the query). We will provide evidence that TF-IDF, though relatively simple, is a big improvement over this naive approach.

**3.2 Experimental Results**

<table>
<thead>
<tr>
<th>Return Pos.</th>
<th>Document #</th>
<th>Sum ( f_w, d )</th>
<th>Sum ( md )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>139</td>
<td>1.83</td>
</tr>
<tr>
<td>2</td>
<td>879</td>
<td>136</td>
<td>4.52</td>
</tr>
<tr>
<td>3</td>
<td>1037</td>
<td>121</td>
<td>2.08</td>
</tr>
<tr>
<td>4</td>
<td>324</td>
<td>107</td>
<td>0.91</td>
</tr>
<tr>
<td>5</td>
<td>710</td>
<td>98</td>
<td>7.22</td>
</tr>
<tr>
<td>6</td>
<td>161</td>
<td>93</td>
<td>3.95</td>
</tr>
<tr>
<td>7</td>
<td>1175</td>
<td>87</td>
<td>0.24</td>
</tr>
<tr>
<td>8</td>
<td>402</td>
<td>86</td>
<td>5.13</td>
</tr>
</tbody>
</table>

Table 1. First eight documents with highest \( f_w, d \) returned by our naive algorithm for query “the trafficking of drugs in Colombia”. The high \( J_i, d \) comes mostly from long documents with plenty of articles and prepositions. These documents had very low \( w \), scores and are mostly useless to the query.

**Query Retrieval of Naive**

Let us now consider the results from running TF-IDF on our data. In this case, documents at the top of the list have a high sum of \( md \), so a query containing \( w \) would likely receive document \( d \) as a return \( \text{run} \) value.

**REFERENCE**