Annals of the University of Craiova, Mathematics and Computer Science Series
Volume 44(2), 2017, Pages 238–248
ISSN: 1223-6934

Custom IDF weights for boosting the relevancy of retrieved documents in textual retrieval

Ioan Badarinza and Adrian Sterca

Abstract. In this paper we present a technique which allows the user to alter the weights of query terms in a textual retrieval system so that it returns more relevant results. This technique is not meant to increase the relevancy of results returned for general search queries, but is meant to increase the relevancy of the returned results for some specific queries in which the query terms have disproportionate IDF values.

Key words and phrases. text retrieval, idf, query term, bm25f, rank.

1. Introduction

In Information Retrieval (IR), the items we are trying to retrieve are called documents, and each document is described by a collection of terms. These two words, 'document' and 'term', are now traditional in the vocabulary of IR. Usually a document is seen as a piece of text, and a term as a word or phrase which helps to describe the document, and which may occur one or several times in the document. So, for example, a document might be about military activity, and could be described by corresponding terms 'gun', 'action', 'soldier', 'uniform', 'war', 'captain', 'force' and so on. More generally a document can be anything we want to retrieve, and a term any feature that helps describe the document.

In all IR systems, a score of each document from the collection is computed against the query and the top k documents (where k is usually 10) are returned. The score of a document is computed as a sum against each term of the query. The score of a document with respect to a query term usually has two parts:

• a component that measures the importance of the query term for this document (i.e. number of occurrences of the query term in the document; Term Frequency)

• a component that measures the discriminatory power (i.e. specificity) of the query term inside the set of all terms from the document collection (i.e. Inverse Document Frequency)

More formally specified, the score of a document D against a query Q has the following form:

$$Score(Q, D) = \sum_{i=1}^{n} tf(q_i, D) * idf(q_i)$$

where $Q = (q_1, q_2, \cdots q_n)$ is the query, q_i are query terms, $tf(q_i, D)$ is the term frequency of term q_i in document D and $idf(q_i)$ is the inverse document frequency of

Received December 1, 2015.

term q_i. The score of a document D against a query Q can be seen as a weighted sum of term frequencies, where the weight is given by the $idf(q_i)$ term.

In this paper we present a technique for improving the relevancy of the documents returned by an IR system, by allowing the user to alter the default weights of the query terms in computing the document score. This is useful when there is an important absolute difference between the IDF values of the query terms in the same query (i.e. a query term has an IDF much larger than the IDF of the other query terms) and in this case, documents containing this high IDF term would monopolize the list of returned results.

2. Ranking functions used in IR systems

In this section we present some popular ranking functions used in IR systems, namely the Binary Independence Model used with the Probabilistic Ranking Principle, BM25 and BM25F.

2.1. Binary Independence Model and Probabilistic Ranking Principle. In the Binary Independence Model and Probabilistic Ranking Principle [4] we consider tf_i to be a binary variable that will have only the values 0 and 1. This can be interpreted in the following way: if a term is present in a document, the binary variable will be equal to 1 and if the term is not present in a document the variable will be 0. The event when the term is absent is the complement of the event when the term is present; probability of absence is one minus probability of presence [5].

$$w_i = log(\frac{P(t_i|rel)(1 - P(t_i|\overline{rel}))}{(1 - P(t_i|rel))P(t_i|\overline{rel})})$$

As mentioned above, the property of relevancy is represented by a random variable Rel with two possible values: $rel, \overline{rel}$ (relevant or not). Further we will use the short notation $P(rel|d, q)$ to describe $P(Rel = rel|d, q)$ (where d is the document and q is the query). In the above formula: t_i is the event that the term t_i is present in the document; rel is the event that the document is relevant to the query; $P(t_i|rel)$ is the probability that the term t_i is present in a document, knowing that the document is relevant; $P(t_i|\overline{rel})$ is the probability the term t_i is present in the document, knowing that the document is not relevant. These probabilities can be estimated in the following way. Because these are conditional on the relevance property, we can assume that we have some judgments of relevance. First, we can assume that we have a random sample of the whole collection that was judged for relevance, than derive an estimator that will be used further. Let's consider:

N **:** the size of a judged sample;
n_i **:** the number of documents from the judged sample containing t_i;
R **:** the relevant set size (i.e., number of documents judged relevant);
r_i **:** the number of judged relevant documents that contain t_i.

Given this information, we can estimate the probabilities from as follows:

$$P(t_i|rel) = \frac{n_i}{R}$$

$$P(t_i|\overline{rel}) = \frac{n_i - r_i}{N - R}$$

After replacing the probabilities and trying to obtain a more robust estimator by introducing a pseudo-count of frequency 0.5, as demonstrated in [6] we get the well-known Robertson/Sparck Jones weight [5]:

$$w_i = log(\frac{(r_i + 0.5)(N - R - n_i + r_i + 0.5)}{(n_i - r_i + 0.5)(R - r_i + 0.5)})$$

2.2. BM25. BM25 if the fundamental result of the Probabilistic Relevance Framework for document retrieval created in 1970-1980s. BM25 is one of the most successful algorithms used in information retrieval systems. In an information retrieval system we cannot know the values of the relevance property for each document so we might say that the information given by the system is probabilistic. Based on probabilistic 2-Poisson model, BM25 algorithm will return the documents that are potentially relevant with our information need. The score formula of the BM25 algorithm is [1]:

$$Score(Q, D) = \sum_{i=1}^{n} log(\frac{N - n_i + 0.5}{n_i + 0.5}) \frac{f(q_i, D) * (k_1 + 1)}{f(q_i, D) + k_1 * (1 - b + b * \frac{|D|}{avgdl})}$$

where $f(q_i, D)$ is q_i's term frequency in the document D, $|D|$ is the length of the document D in words, and $avgdl$ is the average document length in the text collection from which documents are drawn. k_1 and b are free parameters, usually chosen, in absence of an advanced optimization, as $k_1 \in [1.2, 2]$ and $b = 0.75$ [1]. In this formula the term weight is:

$$log(\frac{N - n_i + 0.5}{n_i + 0.5})$$

2.3. BM25F. BM25F is an extended model of BM25 that also incorporates the structure of the documents into the scoring process. One final equation form of BM25F can be sees as [2]:

$$Score(Q, D) = \sum_{t in Q} idf(t) * \frac{weight(t, D)}{k_1 + weight(t, D)}$$

where $idf(t)$ is the inverse document frequency and $weight(t, D)$ is defined as [2]:

$$weight(t, D) = \sum_{c in D} \frac{occurs_{t,c}^{D} * boost_c}{((1 - b_c) + b_c * \frac{l_c}{avl_c})}$$

where l_c is the field length, avl_c is the average length for field c, b_c is a constant related to the field length which is similar to b in BM25 and $boost_c$ is the boost factor applied to field c.

Other classical ranking functions used in information retrieval are described in [7], [8] and [9].

3. Partial user weighting of query terms

We consider the case when a query has multiple terms, but a query term which has a lower IDF ranking is more relevant for the information need (expressed by the query) than some other query term which has a higher IDF weight (i.e. it appears less frequent in the document collection than the first query term). In probabilistic ranking IR, the score of a document relative to a given query is equal to the sum of

the scores of the given document with respect to each query term. The score of the document D with respect to query term t_i can be viewed as $w(t_i)*s(t_i, D)$ where $w(t_i)$ is a variant of IDF which measures the term's t_i specificity in the whole document collection (i.e. it is the weight of term's t_i importance for the returned results) and $s(t_i, D)$ is a function of the term frequency of t_i which ranks the current document against other documents containing t_i (i.e. it is the actual score of the document relative to term t_i). For example, for the BM25 algorithm, $w(t_i) = log(\frac{N-n_i+0.5}{n_i+0.5})$ and $S(t_i, D) = \frac{f(q_i,D)*(k_1+1)}{f(q_i,D)+k_1*(1-b+b*\frac{|D|}{avgdl})}$. In this perspective, the total score of a document relative to a given query can be considered a weighted sum of scores, the scores of this document relative to each of the query terms. So, the weight of each score of the document relative to a query term depends only on the specificity (depends on the IDF) of that query term in the document collection. This constitutes a problem for queries where not all query terms are equally important. For example, in a two terms query where one term is a polysemantic, high IDF term (i.e. it's $weight/IDF$ is high) and the other term is a low IDF term with the role of discriminating between the meanings of the high IDF term, the results returned by the BM25 algorithm can be overwhelmed by documents containing non-relevant meanings (relative to the query) of the high IDF term. For example if the query is 'jaguar lifespan' and we are searching for the lifespan of the animal jaguar on Google, we get as results many documents about Jaguar, the car, which are not relevant to our query (the documents returned don't even contain the word 'lifespan'). This is because the IDF of 'Jaguar' is a lot higher than the IDF of 'lifespan'. Another example is when we search for the height of the TV presenter Bear Grylls using the query 'bear grylls height', we get many documents containing data about Bear Grylls but no height information, because the term 'Bear Grylls' has a much higher IDF (i.e. weight) than the term 'height' which is far more common. Another example is by considering regional languages; if we look up the romanian language query 'Transilvania bani' (which is "Transilvania money" translated to English) on Google.ro in order to obtain information about the Transilvania Bank which is a Romanian financial institution, we also get among the results documents about the Transilvania highway (which is a highway in the Transilvania region). The term 'bani' should discriminate between the two meanings (i.e. Transilvania bank or Transilvania highway), but its IDF is much lower than the IDF of Transilvania.

The queries presented above are just specific examples where allowing the user to alter the IDF weights of query terms would help in increasing the relevancy of the returned results (for this specific user). But in a document collection there are numerous pairs of query terms, one with a low IDF and the other with a high IDF, which would benefit from our technique. For example, in the Reuters-RCV1 collection [3], from 21578 indexed documents and 1217726 indexed terms the distribution of IDFs is depicted in Fig.1 and Fig. 2. In Fig. 1 we see for each integer IDF value, the number of indexed terms which have an IDF close to that specific IDF. Because most IDF values in the index are real values (not integer), the IDF of each term is rounded to the closest integer and then, for each integer IDF value the number of terms which have a rounded IDF equal to this one is counted. We can see in Fig. 1 that most indexed terms have an IDF between 3 and 6. But there are numerous terms which have an IDF smaller than 3 or larger than 6, as we can see in this figure. If the query

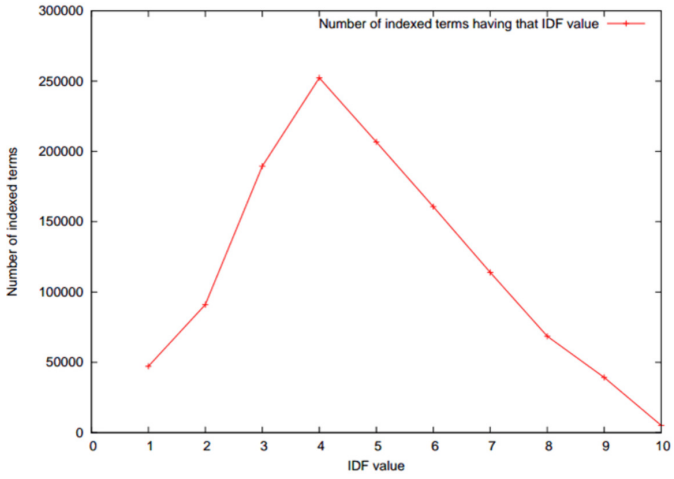

FIGURE 1. Number of indexed terms for specific IDF value in the Reuters collection

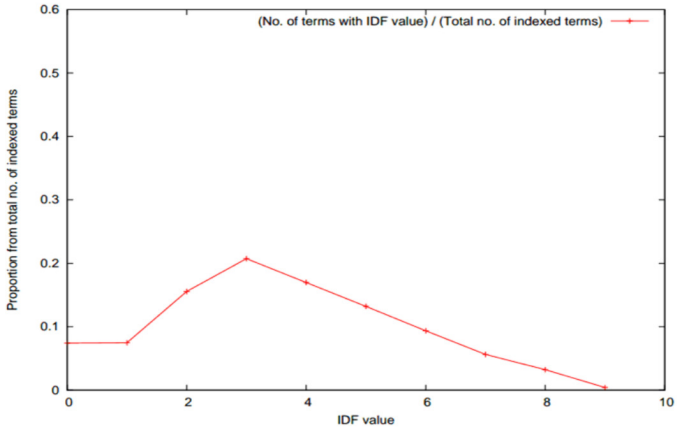

FIGURE 2. IDF value distribution among indexed terms in the Reuters collection

given by the user is formed by a term with an IDF < 3 and another term with an IDF > 6, our technique should improve the relevancy of the returned results.

The same type of IDF values distribution can be seen in Fig. 2 in which the number of indexed terms with a specific IDF value is plotted as a proportion of the

total number of indexed terms. Our technique of allowing the user to alter the IDF weights of query terms is described in this paragraph. We consider a query made from two terms: t_1 and t_2. Let's assume that $IDF(t_1)$ is large and $IDF(t_2)$ is small. We want our IR system to give a weight to t_2 larger or just as high as the one of t_1 because t_2 is more relevant than t_1 in the opinion of the user issuing the query. If there is no document in the collection that contains both t_1 and t_2, a classical IR system will probably return in the first $k = 10$ results only documents containing t_1 (and not t_2). Or even if there are documents in the collection that contain both t_1 and t_2, it may still happen that only documents containing t_1 are returned in the top $k = 10$ results, because the $IDF(t_1)$ is much higher than the $IDF(t_2)$. On the contrary, we want the IR system to return among top $k = 10$ results also documents that contain term t_2, because t_2 is more important for the user than t_1. Assuming that the ranking function of our IR system is:

$$Score(Q, D) = w(t_1) * s(t_1, D) + w(t_2) * s(t_2, D)$$

where $w(t_i)$ is the inverse document frequency of t_i and $s(t_i, D)$ is a function of the term frequency of t_i in document D, we introduce two mechanisms through which the user can alter the default weight of each query term when he specifies the query:

(1) t_1 t_2:++n => by specifying a query like this one, the weight of t_2 will become $w(t_2) + n$, where $n = 0.5, 1, 1.5, \cdots$
(2) t_1 t_2:+n => by specifying a query like this one, the weight of t_2 will become $w(t_2) + n * 0.1 * (w(t_1) - w(t_2))$, where $n = 1, 2, 3, \cdots 10$. When $n = 1$, the weight of t_2 will increase with 10% of the initial weight difference between t_1 and t_2.

4. Evaluations

In order to evaluate the two mechanisms of specifying weights for the query terms, we built an IR system and indexed the Reuters document collection [3]. We emphasize here that our mechanisms for boosting the IDF weights of some query terms are not meant to improve the relevancy of the returned results for general querries, but they are meant to improve the relevancy of returned results for specific querries and specific users. Hence the utility of these mechanisms can not be evaluated in a fully objective manner, because they depend on the personal information need of the user which is subjective in nature. In other words, the employment of our mechanism can be good for a user (i.e. it returns more relevant results based on his/her information need) and bad for some other user (i.e. because other users can have other information needs for the same query) - that is why the human user, not the search engine itself, should choose whether to activate our mechanism by specifying the expressions $: + + n$ or $: + n$ for a query term. In this section we will only present experiments showing that our mechanisms indeed increase the rank of some documents and reorder the returned documents based on the user's personal view of the query, so that more results which contain the low IDF query term are among the top 15 returned results.

We considered a query formed by two terms, *coke* and *prices*, where the IDF of prices is 2.52 and the IDF of coke is 7.07. For evaluating the first mechanism, we ran 11 searches using the query '**coke prices:++n**' where n is 0 in the first search, 0.5 in the second search, 1 in the third one and so on until $n = 5$ in the 11th search. So the first search is a regular search and the remaining 10 searches use our mechanism

for boosting the weight of the **prices** query term. The ranking function used by our search engine is the popular BM25 function described earlier in Section 2.2. In each search operation we took the top 15 documents returned. Considering all 11 searches, there were 27 distinct documents returned. Some of those 27 documents appeared in the top 15 results of all 11 searches performed and some other appeared only in a subset of top 15 results of the 11 searches.

In Fig. 3 we can see the rank (i.e. position from 1 to 15; rank 1 represents the most relevant document/position) of each of the 27 documents in each search performed. Out of the 27 documents depicted, documents with IDs 1-7,9-12 contain only the term 'coke', documents with IDs 13-27 contain only the term 'prices' and the document 'doc8' contains both terms. For a better view, the ranks achieved by each document are connected by a line. Also for a better view, we showed in Fig. 4 only the documents containing only the term 'coke' and in Fig. 5 we showed only the documents containing only the term 'prices' and the document 'doc8' which contains both terms. We can see in these 3 figures, that as parameter n increases across searches, the documents containing the low IDF term, **prices** start to get a higher rank (i.e. a 'high rank' actually means a value closer to 1 in these figures). For example, in Fig. 3 the document 'doc8' which is the only document containing both query terms, is ranked in the first search (i.e. the classical BM25 search) only the 8th in the top 15 results returned. Then, as the weight of the query term **prices** is increased in subsequent searches, we can see that the document 'doc8' was ranked as the 7th in search no. 2 and then, the 6th in search no. 3 and so on until it settles at a rank between 1 and 2 (i.e. the first and the second returned result) in searches no. 6-9. Then, because the weight of the **prices** term is too high, 'doc8' starts to have lower ranks in searches 10 and 11 (rank 3 in search no. 10 and it does not make the top 15 returned results in search no. 11). In Fig. 4 we can see that as the search number increases on the OX axis (so the weight of the **prices** query term increases) the rank of the documents containing only **coke** and not containing **prices** decreases (i.e. it moves away from the value 1 which is the highest rank). And finally, in the last searches these documents are not even among the top 15 returned results. In Fig. 5 we see the opposite for documents containing **prices**: their rank increases (i.e. it gets closer to the value 1) as the search number increases.

We performed a similar evaluation for the second mechanism where the weight of the low IDF term is increased with a percent from the weight difference between the two terms. This time we used the query 'coke prices:+n' for our 11 searches and in the first search $n = 0$, then $n = 1$ for the second search and so on until $n = 10$ for the 11th search. We also considered the top 15 results of each search and we obtain 26 distinct documents from all 11 searches. We can see in figures 6, 7 and 8 the same type of results as we have seen in the evaluation of the first mechanism. As n increases and so the weight of **prices** increase, the documents containing this term get a higher rank (i.e. a rank closer to 1). For example, in Fig. 8 we can see that the only document containing both query terms (**coke** and **prices**), document 'doc8', was ranked as the 7th in search no. 2 and then, the 6th in search no. 3 and so on until it settles at rank 2 in searches no. 6-9 and eventually reaching rank 1 in search no. 10. So, it seams that both mechanisms (query-term:++n and query-term:+n) achieve approximately the same results. This is due to the specific IDF values of the two query terms and the way we have chosen the value of parameter n in both experiments. For our specific

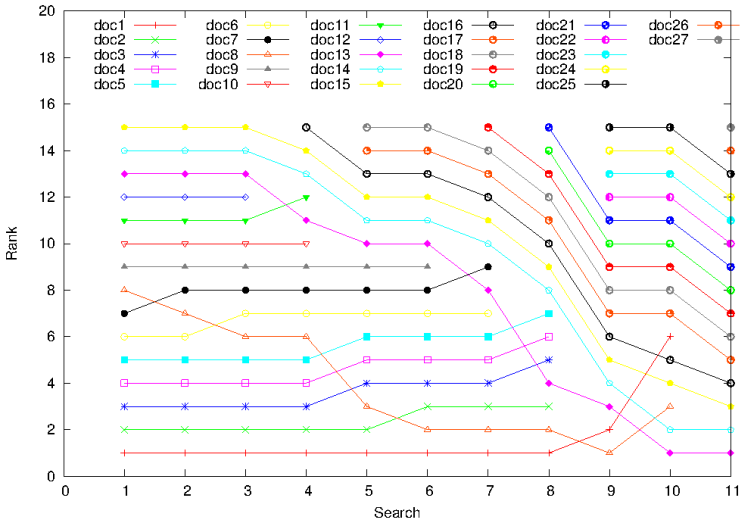

FIGURE 3. The rank of the returned documents in each of the 11 searches (a rank closer to 1 means a more relevant document)

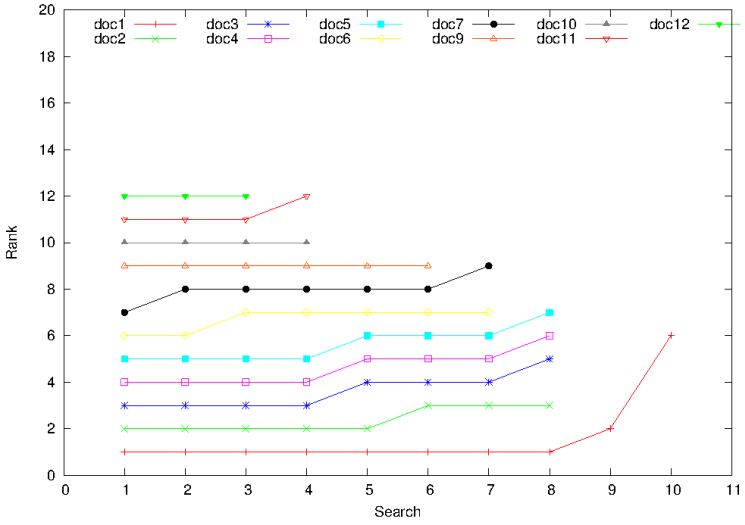

FIGURE 4. The rank of the returned documents which contain only the term 'coke' (a rank closer to 1 means a more relevant document)

query, the term **prices** has the IDF 2.52 and **coke** has the IDF 7.07. So, for the first experiment (i.e. query 'coke prices:++n' is used) in search no. 4 n is equal to 1.5 and the new, boosted weight of **prices** will be $2.52+1.5=4.02$. In the case of the second experiment (i.e. query 'coke prices:+n' is used), in search no. 4 n would be equal to 3 and so the new, boosted weight of **prices** would be $2.52+3\cdot 0.1\cdot(7.07-2.52)=3.885$

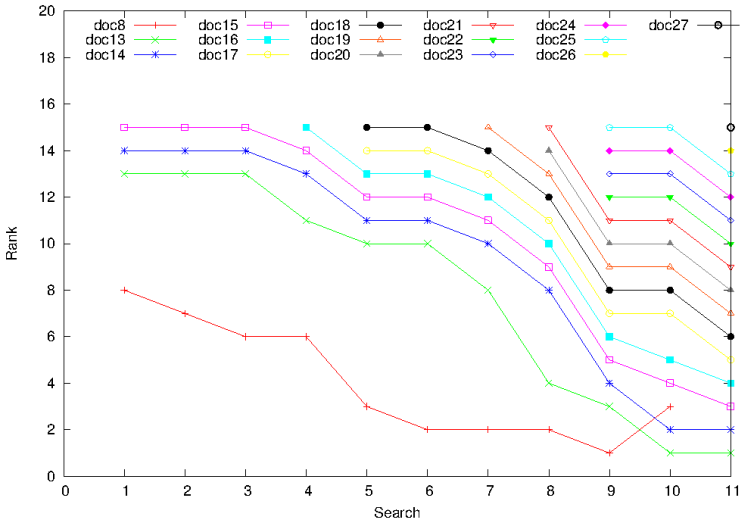

FIGURE 5. The rank of the returned documents containing the term 'prices' (a rank closer to 1 means a more relevant document)

which is very close to 4.02. This is why we see for example in figures 5 and 8 that document 'doc8' has approximately the same rank in the search operations with the same number (e.g. in search no. 2 'doc8' was ranked the 7th in both experiments, in search no. 3 'doc8' was ranked the 6th in both experiments). But if we have chosen a different sequence of values for parameter n in the two experiments we would see different results. Choosing the value of n is very subjective in nature and it is left to the human user.

5. Conclusions

In this paper we presented two mechanisms which allow the user to artificially increase the weight of a query term in order to improve the relevancy of the returned results in an IR system. These two mechanisms are useful when there is an important absolute difference between the IDF values of the query terms in the same query (i.e. a query term has an IDF much larger than the IDF of the other query terms) and in this case, documents containing this high IDF term would monopolize the list of returned results. We also evaluated these two mechanisms and showed that they can improve the results retrieved by an IR system for the case of specific user queries, but because the effects of these mechanisms are very dependent on the user input parameter (i.e. parameter n) we could not evaluate them fully in an objective manner.

References

[1] C.D. Manning, P. Raghavan, H. Schutze, *An introduction to Information Retrieval*, Cambridge University Press, 2009.

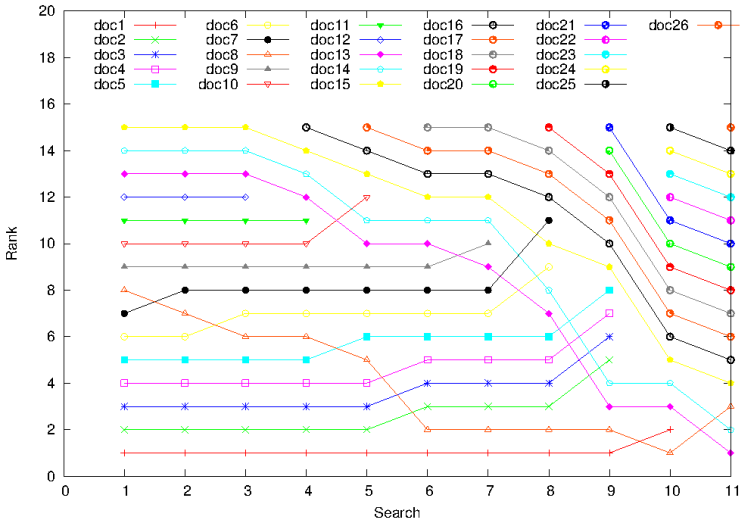

FIGURE 6. The rank of the returned documents in each of the 11 searches (a rank closer to 1 means a more relevant document)

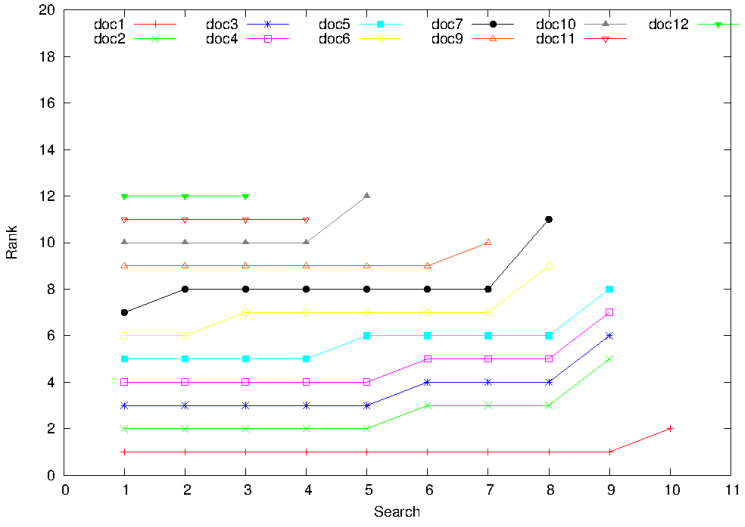

FIGURE 7. The rank of the returned documents which contain only the term 'coke' (a rank closer to 1 means a more relevant document)

[2] J. Pérez-Iglesias, J. Pérez-Aguera, V. Fresno, Y.Z. Feinstein, Integrating Probabilistic Model BM25/BM25F into Lucene, *CoRR* abs/0911.5046 (2009).
[3] Reuters Corpus Volume 1, `http://trec.nist.gov/data/reuters/reuters.html`
[4] V. Rijsbergen, C. Joost, *Information Retrieval*, 2nd edition, Butterworths, 1979.
[5] S. Robertson, H. Zaragoza, The Probabilistic Relevance Framework: BM25 and Beyond, *Foundations and Trends in Information Retrieval* **3** (2009), no. 4, 33–389.

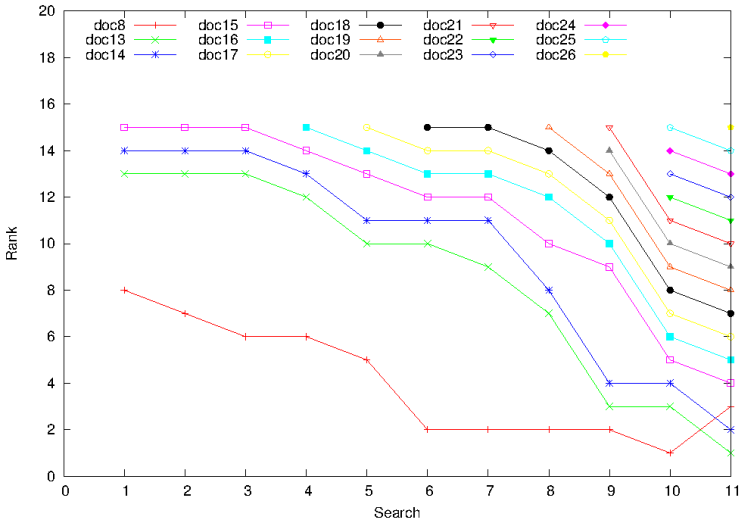

FIGURE 8. The rank of the returned documents which contain only the term 'prices' (a rank closer to 1 means a more relevant document)

[6] S.E. Robertson, K. Sparck Jones, Relevance weighting of search terms, *Journal of the American Society for Information Science* **27** (1977), no. 3, 129-146.

[7] G. Amati, C.J. van Rijsbergen, C. Joost, Probabilistic models of information retrieval based on measuring the divergence from randomness, ACM Transactions on Information Systems **20** (2002), no. 4, 357–389.

[8] S.E. Robertson, C. J. van Rijsbergen, M.F. Porter, Probabilistic models of indexing and searching in Information Retrieval Research, *SIGIR '80 Proceedings of the 3rd annual ACM conference on Research and development in information retrieval*, Cambridge (1980), 35–56.

[9] J. Lafferty, C. Zhai, Document language models, query models, and risk minimization for information retrieval, *ACM SIGIR Forum - SIGIR Test-of-Time Awardees 1978-2001*, **51** (2017), no. 2, 251–259.

(Ioan Badarinza, Adrian Sterca) BABES-BOLYAI UNIVERSITY, FACULTY OF MATHEMATICS AND COMPUTER SCIENCE, 1 M.KOGALNICEANU ST., 400084 CLUJ-NAPOCA, ROMANIA
E-mail address: ionutb@cs.ubbcluj.ro, forest@cs.ubbcluj.ro