The Importance of Proper Weighting Methods

Chris Buckley

Department of Computer Science
Cornell University
Ithaca, NY 14853

ABSTRACT

The importance of good weighting methods in information retrieval — methods that stress the most useful features of a document or query representative — is examined. Evidence is presented that good weighting methods are more important than the feature selection process and it is suggested that the two need to go hand-in-hand in order to be effective. The paper concludes with a method for learning a good weight for a term based upon the characteristics of that term.

1. INTRODUCTION

Other than experimental results, the first part of this paper contains little new material. Instead, it's an attempt to demonstrate the relative importance and difficulties involved in the common information retrieval task of forming documents and query representatives and weighting features. This is the sort of thing that tends to get passed by word of mouth if at all, and never gets published. However, there is a tremendous revival of interest in information retrieval; thus this attempt to help all those new people just starting in experimental information retrieval.

A common approach in many areas of natural language processing is to

1. Find "features" of a natural language excerpt
2. Determine the relative importance of those features within the excerpt
3. Submit the weighted features to some task-appropriate decision procedure

This presentation focuses on the second sub-task above: the process of weighting features of a natural language representation. Features here could be things like single word occurrences, phrase occurrences, other relationships between words, occurrence of a word in a title, part-of-speech of a word, automatically or manually assigned categories of a document, citations of a document, and so on. The particular overall task addressed here is that of information retrieval – finding textual documents (from a large set of documents) that are relevant to a user's information need. Weighting features is something that many information retrieval systems seem to regard as being of minor importance as compared to finding the features in the first place; but the experiments described here suggest that weighting is considerably more important than additional feature selection.

This is not an argument that feature selection is unimportant, but that development of feature selection and methods of weighting those features need to proceed hand-in-hand if there is to be hope of improving performance. There have been many papers (and innumerable unpublished negative result experiments) where authors have devoted tremendous resources and intellectual insights into finding good features to help represent a document, but then weighted those features in a haphazard fashion and ended up with little or no improvement. This makes it extremely difficult for a reader to judge the worthiness of a feature approach, especially since the weighting methods are very often not described in detail.

Long term, the best weighting methods will obviously be those that can adapt weights as more information becomes available. Unfortunately, in information retrieval it is very difficult to learn anything useful from one query that will be applicable to the next. In the routing or relevance feedback environments, weights can be learned for a query and then applied to that same query. But in general there is not enough overlap in vocabulary (and uses of vocabulary) between queries to learn much about the usefulness of particular words. The second half of this paper discusses an approach that learns the important characteristics of a good term. Those characteristics can then be used to properly weight all terms.

Several sets of experiments are described, with each set using different types of information to determine the weights of features. All experiments were done with the SMART information retrieval system, most using the TREC/TIPSTER collections of documents, queries, and relevance judgements. Each run is evaluated using the "11-point recall-precision average" evaluation method that was standard at the TREC 1 conference.

The basic SMART approach is a completely automatic indexing of the full text of both queries and documents. Common meaningless words (like 'the' or 'about') are removed, and all remaining words are stemmed to a root form. Term weights are assigned to each unique word (or other feature) in a vector by the statistical/learning processes described below. The final form of a representative for a document (or query) is a vector

$$D_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,n})$$

where D_i represents a document (or query) text and $w_{i,k}$ is a term weight of term T_k attached to document D_i. The similarity between a query and document is set to the inner-product of the query vector and document vector; the information retrieval system as a whole will return those documents with the highest similarity to the query.

2. AD-HOC WEIGHTS

Document or query weights can be based on any number of factors; two would be statistical occurrence information and a history of how well this feature (or other similar features) have performed in the past. In many situations, it's impossible to obtain history information and thus initial weights are often based purely on statistical information. A major class of statistical weighting schemes is examined below, showing that there is an enormous performance range within the class. Then the process of adding additional features to a document or query representative is examined in the context of these weighting schemes. These are issues that are somewhat subtle and are often overlooked.

2.1. Tf * Idf Weights

Over the past 25 years, one class of term weights has proven itself to be useful over a wide variety of collections. This is the class of tf*idf (term frequency times inverse document frequency) weights [1, 6, 7], that assigns weight w_{ik} to term T_k in document D_i in proportion to the frequency of occurrence of the term in D_i, and in inverse proportion to the number of documents to which the term is assigned. The weights in the document are then normalized by the length of the document, so that long documents are not automatically favored over short documents. While there have been some post-facto theoretical justifications for some of the tf*idf weight variants, the fact remains that they are used because they work well, rather than any theoretical reason.

Table 1 presents the evaluation results of running a number of tf*idf variants for query weighting against a number of variants for document weighting (the runs presented here are only a small subset of the variants actually run). All of these runs use the same set of features (single terms), the only differences are in the term weights. The exact variants used aren't important; what is important is the range of results. Disregarding one extremely poor document weighting, the range of results is from 0.1057 to 0.2249. Thus a good choice of weights may gain a system over 100%. As points of comparison, the best official TREC run was 0.2171 (a system incorporating a very large amount of user knowledge to determine features) and the median TREC run in this category was 0.1595. The best run (DOCWT = lnc, QWT = ltc), is about 24% better than the most generally used tf*idf run (DOCWT = QWT = ntc).

24%is a substantial difference in performance, in a field where historically an improvement of 10% is considered quite good. The magnitude of performance improvement due to considering additional features such as syntactic phrases, titles and parts of speech is generally quite small (0 - 10%). Adding features and using good weights can of course be done at the same time; but the fact that somewhat subtle differences in weighting strategy can overwhelm the effect due to additional features is worrisome. This means the experimenter must be very careful when adding features that they do not change the appropriateness of the weighting strategy.

2.2. Adding New Features

Suppose an experimenter has determined a good weighting strategy for a basic set of features used to describe a query or document and now wishes to extend the set of features. In the standard tf*idf, cosine-normalized class of weights, it is not as simple as it may first appear. The obvious first step, making sure the weights before normalization of the new set of features and the old set are commensurate, is normally straightforward. But then problems occur because of the cosine normalization. For example, suppose there were two documents in a collection, one of them much longer then the other:

- $D_1 = (w_{1,1}, w_{1,2}, w_{1,3})$
- $D_2 = (w_{2,1}, w_{2,2}, \ldots w_{2,100})$

Now suppose the new approach adds a reasonably constant five features onto each document representative. (Examples of such features might be title words, or categories the document is in.) If the new features are just added on to the list of old features, and then the weights of the features are normalized by the total length of the document, then there are definite problems. Not only does the weight of the added features vary according to the length of the document (that could very well be what is wanted), but the weight of the old features have

changed. A query that does not take advantage of the new features will suddenly find it much more difficult to retrieve short documents like D_1. D_1 is now much longer than it was, and therefore the values of $w_{1,k}$ have all decreased because of normalization.

Similarly, if the number of new added features tends to be much more for longer documents than short (for example, a very loose definition of phrase), a query composed of only old features will tend to favor short documents more than long (at least, more than it did originally). Since the original weighting scheme was a supposedly good one, these added features will hurt performance on the original feature portion of the similarity. The similarity on the added feature portion might help, but it will be difficult to judge how much.

These normalization effects can be very major effects. Using a loose definition of phrase on CACM (a small test collection), adding phrases in the natural fashion above will hurt performance by 12%. However, if the phrases are added in such a way that the weights of the original single terms are not affected by normalization, then the addition of phrases improves performance by 9%.

One standard approach when investigating the usefulness of adding features is to ensure that the weights of the old features remain unchanged throughout the investigation. In this way, the contribution of the new features can be isolated and studied separately at the similarity level. [Note that if this is done, the addition of new features may mean the re-addition of old features, if the weights of some old features are supposed to be modified.] This is the approach we've taken, for instance with the weighting of phrases in TREC. The single term information and the phrase information are kept separate within a document vector. Each of the separate subvectors is normalized by the length of the single term sub-vector. In this way, the weights of all terms are kept commensurate with each other, and the similarity due to the original single terms is kept unchanged.

The investigation of weighting strategies for additional features is not a simple task, even if separation of old features and new features is done. For example, Joel Fagan in his excellent study of syntactic and statistical phrases[2], spent over 8 months looking at weighting strategies. But if it's not designed into the experiment from the beginning, it will be almost impossible.

2.3. Relevance Feedback

One opportunity for good term weighting occurs in the routing environment. Here, a query is assumed to represent a continuing information need, and there have been a number of documents already seen for each query, some subset of which has been judged relevant. With this wealth of document features and information available, the official TREC routing run that proved to be the most effective was one that took the original query terms and assigned weights based on probability of occurrence in relevant and non-relevant documents[3, 5]. Once again, weighting, rather than feature selection, worked very well. (However, in this case the feature selection process did not directly adversely affect the weighting process. Instead, it was mostly the case that the additional features from relevant documents were simply not chosen or weighted optimally.)

In this run, using the RPI feedback model developed by Fuhr[3], relevance feedback information was used for computing the feedback query term weight q_i of a term as $p_i(1-r_i)/[r_i(1-p_i)]-1$ Here p_i is the average document term weight for relevant documents, and r_i is the corresponding factor for nonrelevant items. Only the terms occurring in the query were considered here, so no query expansion took place. Having derived these query term weights, the query was run against the document set. Let d_i denote the document term weight, then the similarity of a query to a document is computed by $S(q,d) = \sum(\log(q_i * d_i + 1))$

3. LEARNING WEIGHTS BY TERM FEATURES

The ad-hoc tf*idf weights above use only collection statistics to determine weights. However, if previous queries have been run on this collection, the results from those queries can be used to determine what term weighting factors are important for this collection. The final term weight is set to a linear combination of term weight factors, where the coefficient of each factor is set to minimize the squared error for the previous queries[4, 5]. The official TREC runs using this approach were nearly the top results; which was somewhat surprising given the very limited and inaccurate training information which was available.

This approach to learning solves the major problem of learning in an ad-hoc environment: the fact that there is insufficient information about individual terms to learn reasonable weights. Most document terms have not occurred in previous queries, and therefore there is no evidence that can be directly applied. Instead, the known relevance information determines the importance of features of each term. The particular features used in TREC 1 were combinations of the following term factors:

tf: within-document frequency of the term

logidf: log ((N+1)/n), where N is the number of documents in the collection and n is the number of documents containing the term

lognumterms: log (number of different terms of the document)

imaxtf: 1 / (maximum within-document frequency of a term in the document)

After using the relevance information, the final weight for a term in a TREC 1 document was

$$\begin{aligned} W(t_i) = \; & 0.00042293 + \\ & 0.00150083 * tf * logidf * imaxtf + \\ & -0.00150665 * tf * imaxtf + \\ & 0.00010465 * logidf + \\ & -0.00122627 * lognumterms * imaxtf. \end{aligned}$$

There is no reason why the choice of factors used in TREC 1 is optimal; slight variations had been used for an earlier experiment. Experimentation is progressing on the choice of factors, especially when dealing with both single terms and phrases. However, even so, the TREC 1 evaluation results were very good. If the minimal learning information used by this approach is available, the results suggest it should be preferred to the ad-hoc weighting schemes discussed earlier.

4. CONCLUSION

The sets of experiments described above focus on feature weighting and emphasize that feature weighting seems to be more important than feature selection. This is not to say that good feature selection is not needed for optimal performance, but these experiments suggest that good weighting is of equal importance. Feature selection is sexy, and weighting isn't, but optimal performance seems to demand that weighting schemes and feature selection need to be developed simultaneously.

References

1. Buckley, C. and Salton, G. and Allan, J., "Automatic Retrieval With Locality Information Using SMART." Proceedings of the First TREC Conference, 1993.
2. Fagan, J., *Experiments in Automatic Phrase Indexing for Document Retrieval: A Comparison of Syntactic and Nonsyntactic Methods*, Doctoral Dissertation, Cornell University, Report TR 87-868, Department of Computer Science, Ithaca, NY, 1987.
3. Fuhr, N., "Models for Retrieval with Probabilistic Indexing." Information Processing and Management 25(1), 1989, pp. 55-72.
4. Fuhr, N. and Buckley, C., "A Probabilistic Learning Approach for Document Indexing." ACM Transactions on Information Systems 9(3), 1991, pages 223–248.
5. Fuhr, N. and Buckley, C., "Optimizing Document Indexing and Search Term Weighting Based on Probabilistic Models" Proceedings of the First TREC Conference, 1993.
6. Salton, G. and Buckley, C., "Term Weighting Approaches in Automatic Text Retrieval." Information Processing and Management 24(5), 1988, pages 513–523.
7. Salton, G. and Yang, C.S., "On the Specification of Term Values in Automatic Indexing." Journal of Documentation 29(4), 1973, pages 351-372.

Query → / Doc	ntc	nnc	atc	btc	ltc	lnc
ntc	1813	1594	1834	1540	1908	1738
nnc	1818	1453	1916	1595	1993	1607
atc	1558	1473	1682	1437	1757	1499
anc	1892	1467	1908	1645	2000	1396
btc	1241	1179	1454	1231	1493	1237
bnc	1569	1130	1577	1421	1689	1057
ltc	1909	1815	1986	1726	2061	1843
lnc	2221	1857	2126	1887	2249	1716
nnn	0062	0051	0059	0067	0061	0050

Table 1: Comparison of tf * idf variants.
All weights expressed as triplets:
{tf contribution} {idf contribution} {normalization}

- tf:
  - n : Normal tf (ie, number of times term occurs in vector)
  - l : Log. 1.0 + ln (tf).
  - a : Augmented. normalized between 0.5 and 1.0 in each vector. 0.5+0.5*tf/MaxTfInVector
  - b : Binary (ie, always 1)
- idf:
  - n : None (ie, always 1)
  - t : Traditional (log ((N+1)/n)) where N is number of documents in collection and n is number of documents
- normalization:
  - n : None
  - c : Cosine.