Ranking Document Similarity and Determining Optimal Query Length in a Cover Density Approach

Michael Clougher

Whiting School of Engineering, Johns Hopkins University

December 2020

Abstract

We discuss our findings from our implementation of the cover density ranking approach to determining document similarity. This technique was developed specifically with the goal of optimizing similarity to queries with relatively few terms. To that end, we also conduct an experiment to determine an optimal length for the query itself. Our overall results confirm the efficacy of cover density ranking for short queries relative to the more widely adapted technique of vector-cosine scoring. In addition, we find that the length of a query affects the accuracy of cover density much less than other popular methods.

1 Introduction

A document's cover density ranking for a query relies firstly on query terms being located near one another in the target document, as well as the number of times these terms are found. For a particular document, we assign a number to each term that appears based on its position with the document. For example, in the following document we see each position as denoted by a superscript:

Leonardo^1 DiCaprio^2 has^3 finally^4 won^5 his^6 first^7 Oscars^8 award^9 for^{10} survival^{11} epic^{12}, The^{13} Revenant^{14} after^{15} six^{16} nominations^{17}. The^{18} Oscars^{19} were^{20} hosted^{21} this^{22} year^{23} by^{24} Chris^{25} Rock^{26}.

Note that in our implementation we shall mark term positions after preprocessing, while the following document includes positions for all words. A *span* for a document represents a beginning and end set of document positions in which every query term appears between those two positions, inclusively. For example, for the given query "Leaonardo DiCaprio Oscars", we find two spans in the above document, [1, 8] and [1, 19]. A *cover* represents all of the spans that do not contain another span. In our example, [1, 19] contains [1, 8], and so [1, 19] is not a cover for this query. The above document has just one cover for the query: [1, 8], though documents could contain multiple covers. We can determine the covers that each document contains for a query and determine a score for each document from which we can derive a ranking of documents. Note that for a query such as “Chris Rock comedy”, the above document has no cover since the word “comedy” does not appear anywhere in ther document. Thus this document would not be included in the rankings for such a query. Documents scores are calculated by summing the scores of every cover contained. An individual cover of length K is scored by dividing this length by $q - p + 1$, where $q - p$ represents the length of the range of the cover. The equations are summarized below:

$$DocumentScore = \sum_{j=1}^{n} CoverScore(p_j, q_j)$$

where

$$CoverScore(p, q) = \begin{cases} \frac{K}{q-p+1}, & \text{if } q - p + 1 > K \\ 1, & \text{otherwise} \end{cases}$$

To evaluate how well our cover density ranking approach performs, we shall compare results obtained against a vector-cosine implementation. In this model, both documents and queries can be thought of as vectors in a multidimensional space. The dimensions of each vector are represented by the terms that each document or query contains. There are various weighting schemes that can be applied to each dimension, but the most common is TF-IDF, in which the term frequency (TF) of each vector dimension is multiplied by the inverse document frequency (IDF), a measure of how rare a term is for a collection of documents. This weighting method is the one chosen for use in our implementation. Once we have determined vector directions and weights, we can find the cosine similarity between the angles of the vectors and use this score to determine document similarity from which we can rank documents.

2 Related Work

Wilkinson et al. (1995) have shown that results obtained from very short queries are often undesirable according to the users. Query lengths of 2-10 terms were examined across well established similarity measures and results were particularly poor among the shortest query lengths. The fewer the terms that are provided, the more a user would expect to see every term appear in the retrieved document, regardless of inverse document frequencies for each term. A similar result was also found in Rose and Stevens (1996).

Clarke et al. (1999) proposed the cover density ranking approach as described earlier to deal with such queries that were relatively short in length. This approach does not consider a term’s inverse document frequency or average document length, instead we are only concerned with the proximity and cooccurrence of the terms. They found ranking documents by cover density to perform favorably relative to more established methods.

3 Methodology

3.1 Cover Density Application

To preprocess our corpus, we removed punctuation and any words containing a numerical digit. For the sake of computational efficiency, we considered any word of length 3 characters or fewer to be a stopword to be removed. We also utilized the *PorterStemmer* package from the *nltk* Python library to stem our words.

In implementing our cover density model we used two datasets, TIME and FIRE. The TIME dataset is a sample of about 400 articles extracted from TIME magazine. FIRE is a substantially larger dataset of news articles. Each corpus came with a set of queries and relevance judgements as determined by a human which are useful for determining performance of any information retrieval system. However, cover density is designed to be most effective at determining document relevance as judged against a relatively short query. A typical query most suited for cover density will be about 3-7 words in length, yet the queries provided for each dataset here were often substantially longer. Therefore, queries were created manually for each set of provided queries. These manually generated queries were 3-7 words in length, (without stopwords), and were formulated with the intention to choose the most relevant words from the original query and attempt to simulate "normal" user behavior in a web-based search engine application.

An information retrieval system using cover density ranking requires a unique index architecture to be built. In our implementation we create a postings list where each "key" entry (in a Python dictionary structure) contains a term that we encounter in our corpus. Each term then has for its "value" a list of every position in the corpus in which that term appears. The position value is determined by considering the corpus to consist of the concatenation of every document in the corpus back to back. After processing the text, we can then determine each term's position by finding its position in the concatenated list. We also store a *listings* variable which holds the start and end positions in the corpus for each document id.

Once our index is created, we find a list of spans that are candidates to be covers. To accomplish this, we create a pointer for each query term pointing to the first entry in that query term's posting's list. Of these pointers, we find the earliest position and mark that we have "found" this query term at this position before advancing just this query term's pointer, (until reaching the end of the corpus). We then repeat this process of searching through the corpus by updating the earliest pointer one at a time and marking the latest position at which we have found this query term until all query pointers have reached the end of their respective postings lists. After each step of this process, we compare our current position with the previous position in the corpus and use our document listings to determine whether we have crossed a document boundary between the end of the previous document and the start of the current document. If we have crossed such a boundary, then we must reset the marker for each query term to indicate that these terms are "not found". Whenever we have marked a term as being most recently "found" at some position and all query terms have a "found" position, then we add a list of these positions to a list of spans for this query term over the current document. Once we have reached the end of the postings list for each query term, one final step is needed to extract the covers from the list of spans. For each document that has multiple spans, we check if any span $S1$ contains another span $S2$. If $min(S1) <= min(S2)$ and $max(S1) >= max(S2)$, then we know that $S1$ contains $S2$ and so $S1$ is not a cover. We are finally left with the set of covers for each document.

Once we have found these covers, scoring them is relatively simple according to the formula introduced earlier. Here, we use $k = 16$ as suggested by (Clarke, Cormack, & Burkowski, 1995). We note however that values of k in the range from 2-20 were all evaluated and found to have little to no impact on our results. Once each document is given a score from the combined sum of each of its covers, we are able to produce a ranked list of documents for each query. Note that due to how we define a cover, only documents which contain every term in the query can be given a score.

3.2 Query Length Optimization

We now describe the construction of our experiment for determining an optimal query length for a cover density query. To begin, we took the original queries associated with the TIME dataset and performed the same tokenization and stopword removal as before. For each query q_i in the processed query set, we then used the bootstrap method to sample k random words, (without replacement), from q_i for integers k in $[2, 10]$. This process was repeated 100 times so that different samples of words from each query q_i could be evaluated. If some query q_i was found to have fewer than k terms then this query was not included in our calculations for this particular value of k. After computing cover density scores for these queries. we were then left with 900 different rankings files so that there were 100 rankings for each value of k random words. Finally, to compare our results, we created an additional 900 different rankings files as computed by a vector-cosine method across the same queries.

4 Results

4.1 Cover Density Application

To evaluate our cover density implementation, we computed standardized TREC results for our list of manually created queries. For baseline values, we computed TREC results for two vector-cosine scoring models that were implemented

	MAP	P@10
Vector-Cosine (Original Queries)	0.5825	0.2585
Vector-Cosine (Cover Queries)	0.4804	0.2235
Cover Density	0.3122	0.1474

Table 1: TIME dataset results.

through Python's widely available *sklearn* library. The first vector-cosine model processed the original queries while the second processed the same manually generated queries that our cover density algorithm processed. We predict that the vector-cosine scoring method across the originally provided longer queries will obtain the most favorable results. However, what we are truly interested in is how our cover density implementation compares to the vector-cosine method across the same manually created short queries.

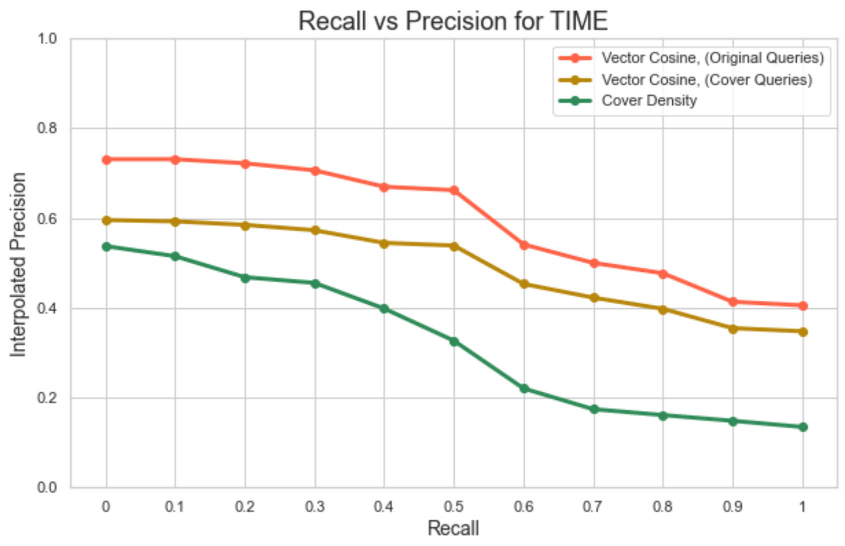

Figure 1: Interpolated precisions for various recall values for the TIME dataset.

We chose to measure accuracy of query results by mean average precision (MAP), and precision at 10 (P@10). In addition, we have provided the interpolated precision graphs for chosen recall values. Table 1 shows our results against the TIME dataset. We see that, as expected, the vector-cosine model with long queries obtained the highest MAP and P@10 values. Vector-cosine with the short queries had the next highest results, while the cover density algorithm provided the lowest score for MAP and P@10 for the TIME dataset. Figure 1 also demonstrates that these relative rankings of algorithms held true for the interpolated precision value across all recall values.

For the FIRE dataset, Table 2 once again shows that longer queries are more

	MAP	P@10
Vector-Cosine (Original Queries)	0.2985	0.2900
Vector-Cosine (Cover Queries)	0.2024	0.2040
Cover Density	0.2115	0.2298

Table 2: FIRE dataset results.

effective in obtaining the most accurate results. However, we see that across this larger dataset, cover density ranking has provided a modest improvement over the vector-cosine method run across the same queries. We see from Figure 2 that this is not true for the interpolated precision across all recall levels, however, it is true for most of them.

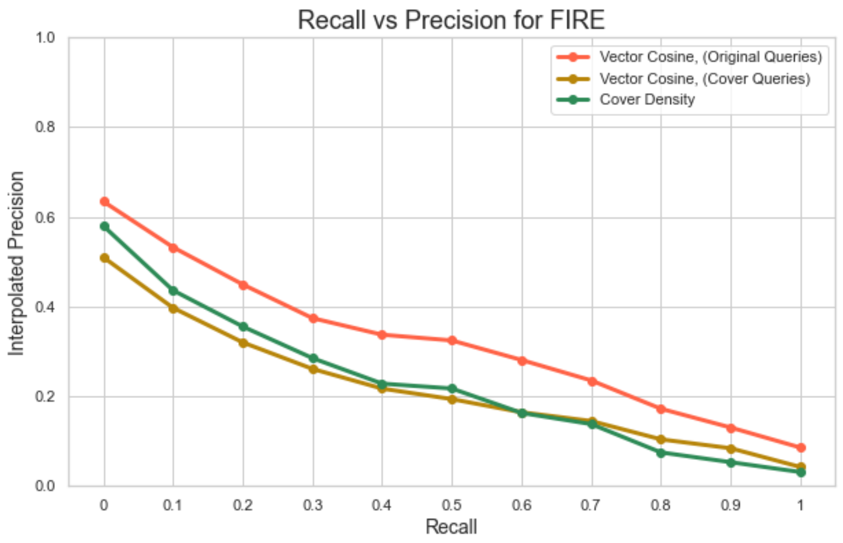

Figure 2: Interpolated precisions for various recall values for the FIRE dataset.

4.2 Query Length Optimization

To determine the results from our experiment, we computed the TREC mean average precision value for all 1800 of our rankings output files, (900 for cover density and 900 for vector-cosine). For each value of $k = [2, 10]$, we then determined the mean MAP of those 100 rankings files for each method employed. Results can be seen in Figure 3. Performances of cover density and vector-cosine are shown, as well as the baseline scores that were found for the TIME dataset for the cover density implementation as well as the vector-cosine method as computed across both the original queries and the manually generated queries of varying lengths.

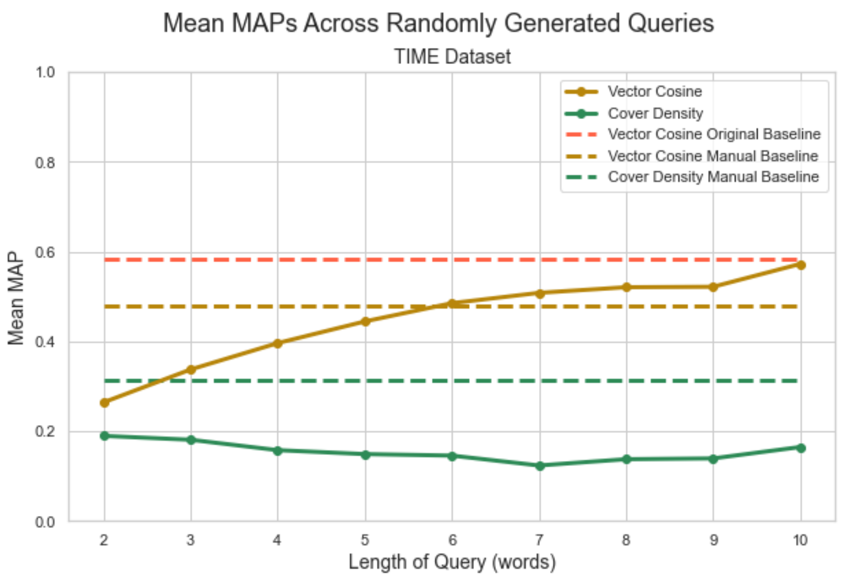

Figure 3: Results from query length optimization experiment.

For the cover density approach, we see little variation in performance across any length query, though $k = 2$ provided the strongest result. The vector-cosine method, however, clearly demonstrates improving performance as the length of the query increases. For query lengths shorter than 6 words, we see the mean MAP lower than the value obtained for our manually generated queries using this method on the TIME dataset. At query lengths of 10 words, we see that this performance has increased to nearly match the performance of our original queries.

5 Discussion

Our results confirm the findings from Clarke et al. that the cover density approach is well suited to evaluating relevance against relatively short queries, however, it should be noted that in our analysis this was only true in the FIRE dataset, a much larger corpus than the TIME dataset in which results were less favorable. The reasoning behind this warrants further investigation, however, one potential reason is that cover density does not employ IDF weighting. In our vector-cosine model, we may have been assigning higher weights to terms that are rare in the corpus that are actually not relevant to the query.

From our query length optimization experiment, we observed that the cover density ranking was much less sensitive to the length of query provided relative to vector-cosine. It was expected to see the vector-cosine model improve as the length of the query improved, and indeed as the number of random words in the query reached 10 we were often left with the original query itself. However, it is also quite important to remember here that in generating our random queries, we excluded those for which there were at least k unique terms in the processed query. Thus there were fewer and fewer queries from our original set contained at least k terms after processing. This means that as k grows, more and more documents will not contain a cover for the randomly generated query as it is required for every query term to be contained in a document in order for there to be a cover. Thus, while the length of the query may not have much direct impact on the quality of results that are retrieved, there will be fewer and fewer results retrieved. It would be interesting in future work to explore whether the recall scores for cover density were disproportionately impacted as well.

We also note that while the vector-cosine model eventually performed as well against the randomly generated queries as it did against the manually generated queries, the cover density approach scored for randomly generated queries fell consistently short of this algorithm's performance against the user generated queries. This suggests the relatively high degree of importance that well-formulated queries have on performance of a cover density approach compared to a vector-cosine implementation. We postulate that this effect can be explained by noting that cover density was created by keeping in mind how a user might generate a search. Users are more likely to formulate a query by choosing words that would appear closely together than could be expected from random chance. The impact that well formulated queries has on the performance of a cover density implementation also leads us to suggest that information retrieval engines that utilize cover density for computing document similarity would be particularly well suited to utilizing relevance feedback methods to improve results from subsequent queries. It would be worthwhile to observe whether a similar drop in performance would be observed across the larger FIRE dataset, in which cover density performed relatively well.

Finally, we note that since the precision of cover density models is highly dependent on our choice of query, it is quite possible that a different user would have retrieved better (or worse) results than what has been observed in this paper. This is important to consider for reproducibility purposes.

6 Conclusion

We conclude that a cover density model is capable of providing improvements over a more established method of determining document similarity such as vector-cosine. Additionally, the overall length of the query seems to have minimal impact on the precision of the documents retrieved in a cover density ranking approach, however, as the length of a query increases a more established method such as a vector-cosine model will likely yield more favorable results. Finally, it is also important to bear in mind that the accuracy of a cover density approach can be highly dependent on the quality of the query formulated; users should take care to carefully craft queries and consider implementing some form of relevance feedback in such an application as cover density ranking.

7 References

Clarke, C. L., Cormack, G. V., & Tudhope, E. A. (1999). Relevance ranking for one to three term queries. Retrieved 2020, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.1615&rep= rep1&type=pdf

Clarke, C. L., Cormack, G., & Burkowski, F. J. (1995). Shortest substring ranking. Fourth Text Retrieval Conference (TREC-4) Gaithersburg, MD, 295-304.

Rose, D., & Stevens, C. (1996). V-Twin: A lightweight engine for interactive use. Fifth Text Retrieval Conference (TREC-5) Gaithersburg, MD, 279-290.

Wilkinson, R., Zobel, J., & Sacks-Davis, R. (1995). Similarity Measures for Short Queries. Retrieved 2020, from http://citeseerx.ist.psu.edu/viewdoc/ summary?doi=10.1.1.54.1097