Improving Search Quality with Automatic Ranking Evaluation and Tuning

Larícia Cavalcante[1,2], Ullayne Lima[1,2], Luciano Barbosa[1], Ana Luiza Gomes[2], Éden Santana[1], Thiago Martins[1]

[1]Centro de Informática, Universidade Federal Pernambuco, Recife – PE – Brazil

[2] VTEX, Rio de Janeiro – RJ – Brazil

{laricia.mota, ullayne.lima, ana.motta}@vtex.com.br

{luciano, eeas, tasm2}@cin.ufpe.br

Abstract. *Search is a common feature available in document-based applications. It allows users to find information of interest easier. Two essential aspects for building an effective search is to evaluate the ranking quality and be able to efficiently tune it based on this evaluation. In this paper, we present our Automatic Ranking Tuning and Evaluation System (ARTES) that measures the ranking performance based on users' clicks on search resulting pages and automatically tunes the search ranking function by applying a Bayesian Optimization algorithm. Our system is integrated with Elasticsearch, a widely-used search engine, which provides the search functionality. The whole solution is currently used by our customer support platform to help users effectively find relevant information, as our experimental evaluation confirms.*

1. Introduction

Document retrieval is a key activity in our daily digital experience because it allows users to find relevant content in large document collections easier. Examples of this task are search on emails, social media messages, personal computer content (desktop search) and web pages (web search). Regarding web usage, for instance, it is estimated that more than 90% of online activity starts with a web search[1].

Many document retrieval engines (e.g., Elasticsearch[2], Solr[3] and Terrier[4]) are available to ease the development of the search functionality over a collection of documents. It might happen, though, that when one builds this functionality, the search quality is overlooked or not properly assessed, and the selection and adjustment of the ranking score function in the engine is made ad-hoc. In this work, we tackle these issues by building a solution, which we name Automatic Ranking Tuning and Evaluation System (ARTES), that: (1) evaluates the ranking quality using a well-established information retrieval metric (NDCG [Baeza-Yates et al. 1999]); and (2) efficiently tunes the ranking function by applying a Bayesian Optimization algorithm (Tree-structured Parzen Estimator [Bergstra et al. 2011]). We use Elasticsearch (ES) as our search engine since ES has

[1]https://www.imforza.com/blog/8-seo-stats-that-are-hard-to-ignore/
[2]www.elastic.co
[3]https://lucene.apache.org/solr/
[4]http://terrier.org/

been the most popular search tool in the market for many years[5]. We integrate, hence, ARTES with ES allowing users to find relevant information available in our customer support platform. Although ARTES has been implemented in this specific configuration, it is generic enough to work in different instances of optimization algorithms and search engines.

In the experimental evaluation, we assess the search quality of our solution and compare it with other strategies. The results show that ARTES can provide better search results than some baselines and a model without tuning.

2. Background

Our customer support software is a system developed to be a content resource to help users throughout the usage of our platform. It contains several guides, tutorials and articles in order to support our customers. Writing these articles is a primordial task to keep the platform's support information up-to-date. This application is integrated with a Content Management System (CMS) to store and deliver the documents. In particular, we use Contentful[6] as CMS.

The document structure is composed of the following fields: (a) *id*: a unique identifier for the document; (b) *title*: the document's title; (c) *locale*: the language of the content; (d) *URL*: the document's URL; (e) *text*: the whole content of the document; and (f) *contentType*: the category of the document. The values of *contentType* are: (1) announcement: feature release announcements; (2) Frequently Asked Questions: frequently asked question about any feature in the platform; (3) Known Issue: known issues of the platform; (4) Track Article: a document belonging to a set of articles with step-by-step instructions; and (5) Tutorial: an article describing a feature or module of the platform.

Given that the system is used across many different countries, it is necessary to ensure that the support content is available in three different languages: Portuguese, Spanish and English. We use Elasticsearch (ES) as our search engine, which indexes each document on CMS in its respective index language. We provide, therefore, search in those 3 languages. We do not perform any stemming or stopword removal in the documents to index them.

3. Proposed Solution

As before mentioned, in this work, we propose a solution to evaluate and rank documents from a customer support site in a principal way. Figure 1 depicts an overview of its architecture. The Search Evaluator is responsible for evaluating the quality of search algorithms based on the click log search from previous interactions with the search system. This component is used by the Search Evaluation Report to provide analysis over the ranking configurations. The Search Tuning module searches for the best ranking configuration (ranking algorithm and weights). To retrieve the results, it queries the Search API provided by ElastiscSearch, and to evaluate the ranking quality of each configuration, uses the Search Evaluator. In the following, we provide further details of each component of our solution.

[5]https://db-engines.com/en/ranking
[6]https://www.contentful.com/

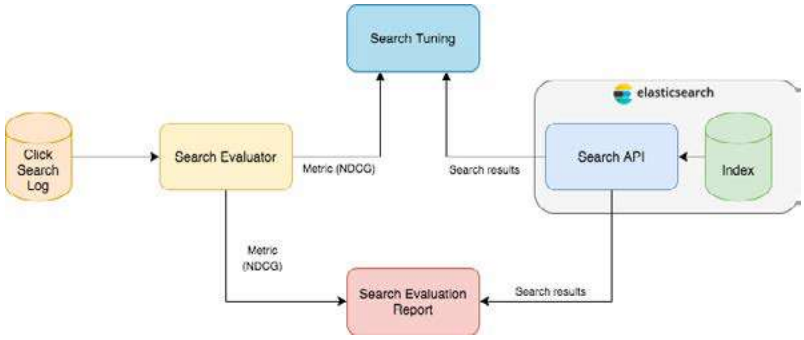

Figure 1. ARTES' System architecture.

3.1. Search Evaluator

To evaluate the search's ranking, the Search Evaluator collects document relevance information from the Click Search Log and uses it to calculate the ranking quality.

The interaction of the users with the search site is logged into the Click Search Log system. In our solution, we use Google Analytics[7] (GA) for this purpose. For each submitted query, GA provides the number of clicks on the documents inspected by the user right after the query, which we call destination pages. We consider a successful search when the user clicks on destination pages in the query's resulting list, and an unsuccessful search when she/he goes to other pages on the website. These two types of pages are easy to differentiate in the logs since the query's resulting pages have in their URL the document id, whereas URL of pages from unsuccessful searches do not.

To measure the quality of the ranking, we consider the information obtained from the successful searches. For that, we use the number of clicks on destination pages as a relevance signal. Our main assumption is: the more a resulting page is clicked by the users, the higher its relevance is. Since the number of clicks on destination pages varies considerably across queries, we compute the relevance score of a destination page for a given query q, a set of terms, by normalizing and discretizing its clicks according to the formula:

$$rel(t) = int(\frac{clicks(t)}{maxClicks} * 4) \quad (1)$$

where $clicks(t)$ is the number of clicks on the destination page t, $maxClicks$ is the number of clicks of the most clicked destination page of q, and int is the function that rounds down a number to the closest integer. The relevance score of documents ranges from 0 (lowest relevance) to 4 (highest relevance).

For a given query q and the relevance values of its destination pages, we measure the quality of the ranked results using Normalized Discounted Cumulative Gain (NDCG). The NDCG is the normalization of the Discounted Cumulative Gain measure. The DCG at a certain k is the sum of the relevance scores of the top-k documents in the search result:

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{log(i+1)} \quad (2)$$

where i is the position of the rank and rel_i is the relevance score of the document at this position. The highest value of DCG@k for q is called Ideal Discounted Cumulative

[7]http://www.google.com/analytics/

Gain at k (IDCG@k). The IDCG@k of q is calculated by sorting in descending order the relevance scores of the destination pages of q, which in our solution we extract from the user query log as previously mentioned. The NDCG@k is then calculated as:

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{3}$$

The NDCG measures, therefore, how similar the rank returned by a ranking algorithm (DCG@k) is close to the ideal rank (IDCG@k). An NDCG@k equals to 1 represents a perfect ranking.

Implementation. We developed the Search Evaluator module in Python. It is composed of two components: data processing and API. The data processing reads the user logs from GA, converts the search queries to lowercase, removes special characters and punctuation, and calculates for each destination page of a query its relevance score, as described in Equation 1. This process creates the ground-truth data (GT), in which each query in the GT has a list of destination pages with their respective score. This information is then used to calculate the NDCG@k. The API, implemented in Flask[8], contains the entry point that, given the search results of a query, returns the NDCG at different levels of k.

3.2. Search Tuning

In this section, we describe the Search Tuning component responsible for selecting the best configuration of ranking. Our ranking is computed by sorting in descending order the matching score between a document d in the index, composed of fields, and a user query q. The document fields used in the matching are: title, text, URL and contentType. For the last one, we only consider whether the contentType of a document is a "tutorial" or "trackArticle", since these types are considered more relevant according to our business rules. Our matching score $s(q, d)$ is then:

$$s(q,d) = \sum_{i \in \{title, text, url\}} s_i(q,d) * w_i + \sum_{j \in \{tutorial, trackArticle\}} \mathbf{1}_j(d) * w_j \tag{4}$$

where $s_i(q, d)$ is the similarity score computed by a matching algorithm between the content of q and d for field i, w_i is the weight of i, $\mathbf{1}_j(d)$ is the result of the indicator function for d on the contentType value j (i.e., $\mathbf{1}_j(d)$ = 1 when the contenType of d is j, and 0 otherwise), and w_j is the weight of j.

Instead of manually defining a similarity model for the field matching score $s_i(q, d)$ and the weights, we search for the parameters of $s(q, d)$ that produces the best overall ranking. In our context, we measure the quality of $s(q, d)$ by calculating the $NDCG@k$ (Equation 3) of the ranking produced by $s(q, d)$. A naïve approach to find the best values of the parameters of $s(q, d)$ would be to perform a random or grid search over the values' space. This is though very costly due to the high number of possible combinations. Instead, we perform our search tuning applying Bayesian optimization (BO) [Shahriari et al. 2015]. Given an objective function f, BO samples f conditioned on previous samples of f to build a surrogate function f' trying to optimize f by f', which is less costly than f. In our context, f is $NDCG@k(x)$, where x is a set of values of the parameters of $s(q, d)$. BO aims, hence, to find the maximum value of f given a defined parameter search space. We chose the Tree-structured Parzen Estimator (TPE) as the BO algorithm.

[8]https://flask.palletsprojects.com

		K	
Ranking Algorithm	**Best Weights**	**10**	**20**
BM25	[3,2,3,5,1]	0.509	0.521
DFR	[5,3,4,4,1]	0.501	0.514
DFI	[3,2,3,4,1]	0.511	0.523
IB	[4,4,4,3,1]	0.513	0.521
LM Dirichlet	[5,1,5,2,1]	0.475	0.492
LM Jelinek-Mercer	[3,3,3,5,1]	0.508	0.522
BM25 (all fields, no weights)	-	0.475	0.488
Contentful	-	0.180	0.211
Google	-	0.197	0.195

Table 1. Results of the evaluated models. The order of weights is the following: [title, text, url, tutorial, trackArticle] for the models that use weights.

Implementation. As Figure 1 shows, we implemented our Search API component using Elasticsearch. The approach to setting weights to the title, text and URL fields is through multi-match query[9], boosting individual fields with the caret ($\wedge$) notation. Furthermore, to add weights when a document's content type is "tutorial" or "trackArticle", we employ the function score query[10]. It allows the definition of functions that modify the score of documents retrieved by a query. In our case, we create functions to check whether the contentType of the document is in the tutorial or trackArticle category, setting its weight accordingly. These weights are added to the matching score, as shown in Equation 4. To find the best values of the parameters (matching model and weights) of $s(q, d)$, we use the TPE implementation available on the Bayesian optimization framework Optuna [Akiba et al. 2019]. To choose the matching model in order to calculate $s_i(q, d)$, the search tuning process evaluates six well-established information retrieval models [Baeza-Yates et al. 1999], available on Elasticsearch: BM25, Divergence from randomness (DFR), Divergence from independence (DFI), Information based model (IB), LM Dirichlet (LMD) and LM Jelinek Mercer (LMJM).

4. Experiments

Setup. The Search Evaluator generated the ground truth from click information between 01/04/2019 and 16/06/2020 from a GA log that has over 15 thousand entries. The set of search terms is composed of 300 queries: the 100 most frequent distinct ones for each of the three languages available in our system (English, Portuguese and Spanish). For each one of these queries, we only consider destination pages with more than 1 click. We use NDCG@10 and NDCG@20 to evaluate the ranking quality of all strategies, and NDCG@10 for the weight optimization of the models, as described in Section 3.2. These metrics were calculated using around 1400 documents for each language. The search space for each one of the weights is the integers in the interval [1-5]. On average, the BO algorithm executed a total of 637 combinations for each ranking algorithm. In addition to these models, we evaluate the BM25 algorithm searching using the 5 fields without any weights, and other two search engines: Google and Contentful (the engine used on our CMS, as mentioned in Section 2).

Results. Table 1 shows the results of the evaluated models computed for all 300 queries. It can be observed that our solution using weights, the first 6 lines, has outperformed the other approaches. However, comparing the ranking algorithms using weights, the NDCG values are very similar. To check whether their results regarding the distribution

[9]https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-multi-match-query.html
[10]https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-function-score-query.html

of NDCG@10 for their queries have the same distribution (null hypothesis), we executed the Friedman Test [Sprent and Smeeton 2016] with a significance level equals to 0.05. In the first comparison, we assessed all six ranking configurations that use weights and concluded that their distributions are different (p-value=0.001). Since the LM Dirichlet model obtained the lowest NDCG@10 and NDCG@20 values among the six models, we executed another Friedman Test removing it from the comparison. The resulting p-value (0.243) accepts the null hypothesis, meaning that the distributions of the five remaining models are similar. We also evaluate the impact of weight selection in the search quality. For that, we executed a statistical test (Wilcoxon [Sprent and Smeeton 2016]) to compare BM25 (best weights) with BM25 (all fields, no weights), with the alternative hypothesis considering that the median of the results of the first model is greater than the second one. The test result confirms that there is a statistical difference between the two configurations: p-value=7.735e-10. We can conclude from these numbers that our strategy of assessing the ranking and adjusting it using a Bayesian optimization, in fact, improves the quality of the search functionality over the documents on our customer support platform. Analyzing the sets of best weights in Table 1, we observe that the values of the weights depends on the algorithm, since each algorithm has a different set of weights. The field trackArticle, however, is the only exception, having the lowest value in all six algorithms. This is due to the low proportion of articles in this category in our article collection: trackArticle corresponds to less than 10% of the articles, whereas tutorial to more than 70%.

5. Conclusions

In this paper, we presented our solution that instead of informally or in an ad-hoc manner evaluating and choosing the ranking algorithm, it uses a well-established metric to assess the ranking quality and automatically selects the best ranking configuration using a Bayesian optimization algorithm. In addition to show conceptually how our system (ARTES) works and its integration with Elasticsearch, we also provide implementation details about it. The experimental evaluation shows ARTES can provide, for the set of most frequent queries, better ranking models than some search engines and a strategy with no weight optimization. A potential improvement to ARTES would be to add other information retrieval metrics such as mean average precision (MAP) and Precision@k to the Search Evaluator model.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD*, pages 2623–2631.

Baeza-Yates, R., Ribeiro-Neto, B., et al. (1999). *Modern information retrieval*, volume 463. ACM press New York.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *NeurIPS*, pages 2546–2554.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Sprent, P. and Smeeton, N. C. (2016). *Applied nonparametric statistical methods*. CRC press.