



Learning to Rank for Query Auto-completion in the Legal Domain

Marcos Aurélio Domingues   [State University of Maringá and Federal University of Amazonas and Jusbrasil | madomingues@uem.br]

Lucas Rocha  [Federal University of Amazonas and Jusbrasil | lucas.rocha@jusbrasil.com.br]

Edleno Silva de Moura  [Federal University of Amazonas and Jusbrasil | edleno@icomp.ufam.edu.br]

Altigran Soares da Silva  [Federal University of Amazonas | alti@icomp.ufam.edu.br]

 Department of Informatics, State University of Maringá, Avenida Colombo, 5790, Jardim Universitário, Maringá, PR, 87020-900, Brazil.

Received: 19 March 2024 • Accepted: 15 April 2025 • Published: 06 June 2025

Abstract Most modern Web search engines implement query auto-completion (QAC) to facilitate faster user query input by predicting users' intended query. This is the case of Jusbrasil, Brazil's most prominent and widely used legal search engine platform. Query auto-completion is typically performed in two steps: matching and ranking. Matching refers to the selection of candidate query from a suggestions dataset. Ranking sorts the matching results according to a score function that attempts to select the top most relevant suggestions for the user. In this paper, our main goal is to explore the effectiveness of learning to rank algorithms on the ranking step for query auto-completion in the legal domain. In particular, we explore four learning to rank algorithms: LambdaMART, XGBoost, RankSVM and Genetic Programming. LambdaMART is widely used in query auto-completion. On the other hand, as far as we know, this is the first time that the RankSVM and XGBoost are used for this task. Additionally, we propose the use of Genetic Programming as a lightweight and viable alternative for query auto-completion. One difficulty for exploring learning to rank algorithms in query auto-completion is the lack of fine-grained training and test datasets, since learning to rank algorithms rely on a large number of features. To bridge this gap, and also to foster research on this area, we propose two datasets with different types of features for query auto-completion in the legal domain. The datasets were created by collecting data from several data sources from Jusbrasil, including contextual features from search query logs, enriched with additional features extracted from other data sources like auto-completion log, document content and metadata available at Jusbrasil. Then, we show that learning to rank is effective for query auto-completion in the legal domain by answering four main research questions: 1) How each feature, specially the novel ones proposed in our work, impact the rankings in query auto-completion?; 2) How effective is learning to rank with respect to the Most Popular Completion (MPC), a ranking algorithm widely adopted as baseline in the literature?; 3) Among the four alternatives experimented, which learning to rank algorithm is more effective in the legal domain?; and 4) How effective is learning to rank with respect to ranking models based on BERT and ColBERT? Finally, we conduct an online A/B test at Jusbrasil.

Keywords: Query auto-completion, Learning to rank, Contextual and additional features, LambdaMART, RankSVM, XGBoost, Genetic Programming, BERT, ColBERT

1 Introduction

Query auto-completion (QAC) is one of the most important component of modern search engines like Jusbrasil¹, Brazil's most prominent and widely used legal search engine platform. A query auto-completion system plays a key role in assisting users to express their information need more efficiently by suggesting full queries after the user has typed a prefix of a few characters [Cai and de Rijke, 2016]. As illustrated in Figure 1, while the user is typing in the search box, the query auto-completion component offers a list of query completions that might be useful for the user. By using this component, the user does not need to type the whole query, and in some cases the auto-completion suggestions may help the user to formulate a more accurate query.

Query auto-completion tasks is often performed in two

steps: matching and ranking. Matching refers to the selection of candidate query suggestions according to the exact or approximate match between a given prefix and the full queries in a suggestions dataset. More recent works focus on using language models to generate the query suggestions [Wang *et al.*, 2020]. The second step, i.e. ranking, sorts the matching results according to a score function that attempts to select the top most relevant suggestions for the user. This paper focuses on the ranking step.

Ranking in query auto-completion has been intensively studied in the literature in recent years. A detailed survey by Cai and de Rijke [2016] classifies ranking in query auto-completion into two main categories: heuristic approaches and learning-based approaches. Heuristic approaches use fixed algorithms to compute a final score for each query. These approaches use only a few features to compute the relevance of the candidate query suggestions for a given prefix.

¹<https://www.jusbrasil.com.br>

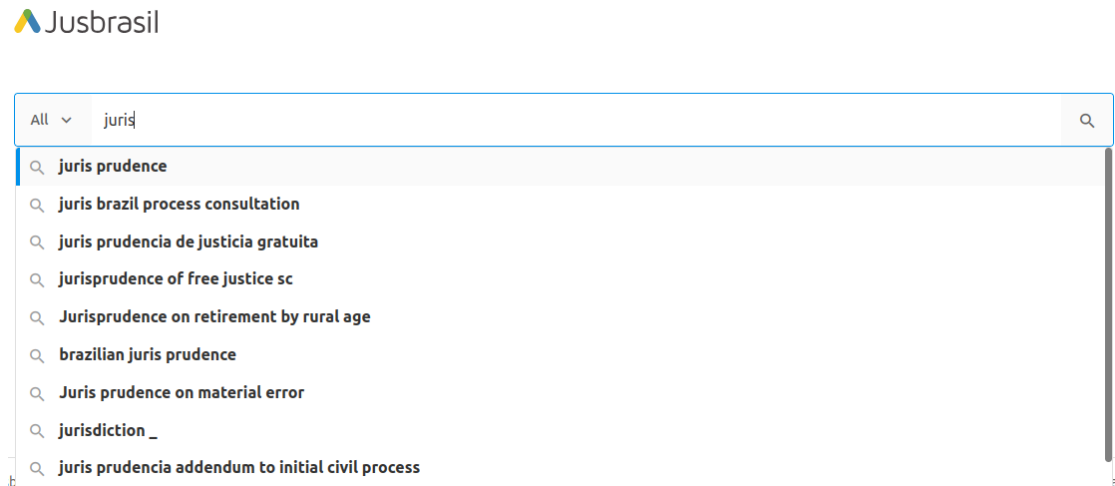


Figure 1. Illustration of QAC in Jusbrasil search engine for the query “juris”.

On the other hand, learning-based approaches apply learning to rank techniques, which have been extensively studied in the information retrieval literature [Liu, 2009]. They rely on a large number of features to compute the relevance for the candidate query suggestions, and generally outperform heuristic approaches [Cai and de Rijke, 2016].

In this paper, we aim to explore the effectiveness of learning to rank algorithms on the ranking step for query auto-completion in the legal domain. In particular, we explore four learning to rank algorithms: LambdaMART, RankSVM, XGBoost and Genetic Programming. One difficulty for exploring learning to rank algorithms in query auto-completion is the lack of fine-grained training and test datasets. To bridge this gap, and also to foster research on this area, we introduce two datasets with different types of features for query auto-completion in the legal domain. Then, we show that learning to rank is effective for query auto-completion in the legal domain by answering four main research questions:

1. How each feature, specially the novel ones proposed in our work, impact the rankings in query auto-completion?
2. How effective is learning to rank with respect to the Most Popular Completion (MPC) [Bar-Yossef and Kraus, 2011], a ranking algorithm widely adopted as baseline in the literature?
3. Among the four alternatives experimented, which learning to rank algorithm is more effective in the legal domain?
4. How effective is learning to rank with respect to ranking models based on BERT [Nogueira et al., 2019] and ColBERT [Khattab and Zaharia, 2020]?

With the first question, we intend to evaluate the impact that each ranking feature can bring to query auto-completion in the legal domain. Our goal with the second question is to see whether the learning to rank algorithms can outperform the MPC, which is a naive approach based on the candidate queries past popularity, widely used in the literature as a baseline for query auto-completion. In the third question, we analyze which learning to rank algorithm, i.e. LambdaMART, RankSVM, XGBoost and Genetic Programming,

provides the best result for ranking in query auto-completion. Finally, in the last question, we analyze whether learning to rank algorithms provide better results than ranking methods based on Large Language Models (LLMs), such as adaptations for BERT (Bidirectional Transformers for Language Understanding) to use it in ranking and ColBERT.

Additionally to the empirical evaluation, we also conduct an online A/B test at Jusbrasil.

The remainder of this paper is organized as follows. Section 2 discusses learning to rank algorithms, data and features for query auto-completion. In Section 3, we define ranking in query auto-completion, and present the datasets and algorithms explored in our work. We discuss the results obtained from our empirical evaluation in Section 4. In Section 5, we show the results obtained with an A/B test that we conducted at Jusbrasil search platform. Finally, Section 6 presents the conclusions and future directions for this work.

2 Related Work

In this section, we discuss previous work related to learning to rank algorithms for query auto-completion, and the data and features used by such algorithms.

Cai and de Rijke [2016] presented a detailed query auto-completion survey which classify the methods adopted by these systems into two categories: heuristic approaches and learning-based approaches. Heuristic approaches are adopted to compute a ranking score for each query [Bar-Yossef and Kraus, 2011; Shokouhi and Radinsky, 2012; Di Santo et al., 2015; Jiang et al., 2018a]. On the other hand, learning-based approaches handle the query auto-completion as a learn to ranking problem [Shokouhi, 2013; Jiang et al., 2014; Jiang and Cheng, 2016; Jiang et al., 2018b; Kannadasan and Aslanyan, 2019], and rely on the research from this field [Liu, 2009] to produce optimized ranking functions. According to Cai and de Rijke [2016], learning-based approaches usually outperform the heuristic approaches.

A more recent survey is presented by Tahery and Farzi [2020], where learning-based approaches are extended to consider different types of learning algorithms instead of only learning to rank. There, we can see algorithms based

on time series analysis [Shokouhi and Radinsky, 2012; Whiting and Jose, 2014; Cai *et al.*, 2016], neural networks [Mitra, 2015; Mitra and Craswell, 2015; Park and Chiba, 2017; Fiorini and Lu, 2018; Jaech and Ostendorf, 2018; Wang *et al.*, 2018; Jiang *et al.*, 2018c; Kim, 2019; Wang *et al.*, 2020], probabilistic approaches [Zhang *et al.*, 2015; Cai and Chen, 2017; Li *et al.*, 2017a; Maxwell *et al.*, 2017; Wang *et al.*, 2017], vector clustering [Bar-Yossef and Kraus, 2011; Shao *et al.*, 2018; Hu *et al.*, 2018], and learning to rank [Shokouhi, 2013; Jiang *et al.*, 2014; Mitra and Craswell, 2015; Cai and de Rijke, 2016; Jiang and Cheng, 2016; Park and Chiba, 2017; Fiorini and Lu, 2018; Jiang *et al.*, 2018b; Kannadasan and Aslanyan, 2019]. Besides that, we can also find in the literature algorithms based on click modeling [Li *et al.*, 2014, 2015] and counterfactual learning [Block *et al.*, 2022]. Due to their effectiveness in the ranking task, we have adopted learning to rank algorithms in our study.

With respect to learning to rank algorithms, the query auto-completion literature [Shokouhi, 2013; Jiang *et al.*, 2014; Mitra and Craswell, 2015; Cai and de Rijke, 2016; Jiang and Cheng, 2016; Park and Chiba, 2017; Fiorini and Lu, 2018; Jiang *et al.*, 2018b; Kannadasan and Aslanyan, 2019] has focused on the well known LambdaMART algorithm [Burgess, 2010], which is one of the most effective learning to rank algorithms. LambdaMART transforms a ranking into a pairwise classification or regression problem. The algorithm considers a pair of items at a single time, coming up with a viable ordering of those items before initiating the final order of the entire rank. In our work, we also investigate the use of three other learning to rank algorithms that were not applied in previous query auto-completion scenario: RankSVM [Joachims, 2002], XGBoost [Chen and Guestrin, 2016] and Genetic Programming [Koza, 1994]. We evaluate how the RankSVM, XGBoost and Genetic Programming algorithms behave in query auto-completion. RankSVM is a commonly used pairwise ranking algorithm for search engines [Joachims, 2002]. XGBoost is an implementation of the gradient boosting framework that uses an ensemble of decision trees to make predictions [Chen and Guestrin, 2016]. Finally, Genetic Programming is a type of evolutionary algorithm that automatically evolves computer programs to solve a problem. It uses the principles of natural selection and genetics to evolve a population of computer programs towards an optimal solution, and has been adopted by several authors in learn to rank studies [da Costa Carvalho *et al.*, 2012; Yeh and Lin, 2017].

The quality of the results provided by a learning to rank algorithm depends on its input ranking features. According to Cai and de Rijke [2016], ranking features can be categorized into 3 types: time-sensitive features, demographic features and contextual features. Time-sensitive features are based on query popularity and can change over time. Demographic features, such as location, are typically limited and hard to access. Contextual features rely on the user's previous activity and are easy to extract from the search query logs. For example, the number of terms or the number of clicks in a query can be seen as contextual features. In the literature, several learning to rank works use contextual features [Shokouhi, 2013; Jiang *et al.*, 2014; Jiang and Cheng, 2016; Jiang *et al.*, 2018b; Kannadasan and Aslanyan, 2019].

Besides the features described by Cai and de Rijke [2016], query auto-completion systems may also take advantage of other sources of query suggestions, such as using generative models to automatically creating query suggestions [Kang *et al.*, 2021]. In our work, besides contextual features from search query logs, we enrich our data with additional features extracted from two other data sources, the auto-completion log and structured information present in legal documents. To the best of our knowledge, the information extracted from auto-completion log was previously considered only by Li *et al.* [2017b], which compared several alternative ranking approaches for query auto-completion, studying their performances on a commercial medical search engine.

Our work differs from their work in the following aspects. First, they have experimented each feature as an individual ranker but not considered the combination of them. We here study how to combine features, studying and comparing the performance of four distinct learning to rank approaches. Second, we investigate the use of query auto-completion in a legal documents search service, while they presented a study using a medical search service. Third, we create two public datasets which combine contextual features from search query log with additional features from other data sources. By providing public datasets to the community, we expect to foster research on this area. Fourth, we explore the use of structured information extracted from legal documents as sources of query suggestions, and also propose and study the performance of features related to such suggestions to be used at ranking time. As far as we know, the new sources of query suggestions and the new features associated to them were not previously proposed in the literature. Finally, we conduct an online A/B test at Jusbrasil search platform.

3 Ranking in Query Auto-completion

Formally, a ranking step in query auto-completion takes as input a prefix p of length l typed by the user in the search engine box and a list of candidate queries Q_p to rank. A prefix p is a sequence of characters belonging to the query the user is about to submit. It can be a letter, a word, or even a sequence of letters or words. The candidate queries Q_p is a list of possible completions for the current prefix. The aim of the ranking step is to rank the candidate queries list Q_p for each given prefix p , in an attempt to assign top positions to candidates that have more chance of being selected by the user to express his/her intent.

For the task of ranking the candidate queries Q_p , there exist several algorithms in the literature [Tahery and Farzi, 2020]. The first algorithm proposed for this task, called Most Popular Completion (MPC), is a naive approach based on the candidate queries past popularity [Bar-Yossef and Kraus, 2011]:

$$MPC(q) = \frac{f(q)}{\sum_{q_i \in Q} f(q_i)}, \quad (1)$$

where $f(q)$ denotes the frequency of a query $q \in Q_p$ in the query search log Q , and the denominator of this equation is a normalizer by the total sum of query frequencies. This algo-

rithm is used as baseline in the majority of the experiments presented in the literature.

In our study, we go to a different direction by exploring learning to rank algorithms, in particular, LambdaMART, RankSVM, XGBoost and Genetic Programming algorithms. For this purpose, we use the JusBrasilQAC datasets, which were collected from Jusbrasil for query auto-completion. The datasets contain a list of candidate queries for each prefix with their respective ranking features. They also contain relevance feedback about each candidate query for each given prefix present on them. Thus, we can compare ranking algorithms over the same list of candidate queries. The datasets and the learning to rank algorithms explored in our work are presented in the following sections.

3.1 The JusBrasilQAC Datasets

As already mentioned, one difficulty for exploring learning to rank algorithms in query auto-completion is the lack of fine-grained datasets. We propose two datasets with different types of features for query auto-completion in the legal domain²: JusBrasilQAC-Click and JusBrasilQAC-Submission. Both datasets were created by collecting data from different data sources (i.e. query log, auto-completion log, document content and metadata) from Jusbrasil during the period ranging from 2022-05-01 to 2023-06-01.

In JusBrasilQAC-Click dataset, each record is represented by a tuple containing a prefix query, a candidate query, the set of features associated to the pair, and the relevance of the candidate to the given prefix. The relevance is estimated by counting the number of clicks in the candidate query when the prefix was typed by users in the past. This dataset contains 284,519 records and 17,938 unique prefixes. Similarly, the JusBrasilQAC-Submission contains the same elements, but in this case the relevance is computed by counting the number of times the candidate query was selected by users in Jusbrasil's search platform, and the number of times the user submits the candidate query by completely typing it in the search box. This second dataset contains 194,460 records and 12,433 unique prefixes. In both datasets, each prefix provides at least five candidate queries.

The datasets contain features extracted from 3 distinct sources: from the access logs, from the document body, and from metadata information containing entities extracted from the documents. The metadata of each document contains information about entities found on it or related to it, including defendants names, plaintiffs names, court names, names of laws, and main topics of law related to the document, among others. As we show here, the quality of these ranking features is a deciding factor for the success of applying learning to rank in both datasets presented. We also adopted features exclusively related to the prefix and candidate query pairs content.

Both datasets contain the following features extracted from the search and from the query auto-completion logs:

- *MPC*: number of occurrences of the candidate query in the search log. This is a moderate signal of relevancy and it is wide available on the datasets;
- *dgc*: number representing the DCG [Järvelin and Kekäläinen, 2002] of the candidate query in the search log. Each query result in the company is evaluated using an online evaluation process based on user feedbacks, and we had access to the DCG of each query produced by this evaluation process. Thus, we take the previous DCG for queries that have been submitted at least once and adopted zero for the other queries. Note that we have decided to adopt DCG for two main reasons. First, a previous study that evaluated DCG vs NDCG metrics and their correlation with user satisfaction indicates that DCG is better for web search services [Al-Maskari et al., 2007]. The second reason is that we do not have the ideal ranking, since we do not have a complete list of relevant answers in our evaluation process. This feature indicates the quality of the documents retrieved by the suggested query and it is also wide available on the datasets;
- *totalClicks*: total number of times that the candidate suggestion query was selected from the suggestions in the query auto-completion log, no matter that prefix was typed by the user. This is a moderate signal of relevancy based on past clicked suggestions and it is not easy to collect from the search platform, since a small number of users interact with the suggested queries;
- *clickSuggestionOnPrefix*: number of times that the suggestion query was clicked as a suggestion to a specific prefix typed by the user in the query auto-completion logs. This can be a strong signal of relevancy, since it directly relates a prefix with a suggestion. However, it is also not easy to collect such a feature from the search platform, since a small number of users interact with the suggested queries and it is limited by the prefix.

Both datasets also contain the following features extracted from the Jusbrasil documents textual content:

- *averageWordDocCount*: average number of occurrences for each non stop word present on the candidate query in the Jusbrasil documents. This feature is an indicative that the suggestion is correctly spelled, and it is available for all the candidate queries;
- *minTFFCount*: minimum number of occurrences among the words of the candidate query in the Jusbrasil documents. It is also an indicative that the suggestion is correctly spelled, and it is also available for all the candidate queries.

Besides processing the legal documents as plain text, we also used metadata available in the documents containing information about the entities related to a legal process, including defendant and plaintiff names. We observed that such entity names usually appear as parts of the query or even as the whole query in the search log. Since these entities are recognized by their names in legal processes, we decided to include each of them as a possible query suggestion in the dataset. Further, we included the following features to represent them in our ranking model:

²The company may provide access to these datasets for researchers under the restrictions of Brazilian law of user data protection. Instructions for use and proper future citation of these datasets will be provided by email (datasets@jusbrasil.com.br).

- *averageWordOccurrencesInEntities*: average number of occurrences for each word of the candidate query in the metadata of documents. This would be a moderate relevancy signal, however, it is not available on candidate queries without entities;
- *frequencyOfEntity*: the number of documents where the entity appears, for the suggestions that are extracted from entities found in the dataset. This would be a moderate relevancy signal.

Finally, we also have features exclusively related to the prefix and candidate query pairs content:

- *editDistance*: the minimum edit distance value computed between the prefix and suffixes of interest in the candidate query. We adopted as suffixes of interest all the suffixes of the candidate string at positions that start the words. Edit distance between the prefix already typed by the user and a suffix of the candidate is computed by taking only the initial characters of the suffix, up to the size of the string. The edit distance is referred to as prefix edit distance in the literature [Ferreira et al., 2022]. This feature is used by ordering methods to relate the prefix and the candidate queries, and it is available for all the candidate queries;
- *prefixSize*: number of characters of the prefix. This feature could be used by the ordering methods for find. This feature should be a way for ordering methods to relate the prefix and suggestions, and this is available for all the suggestions.

Figure 2 shows an example of the JusBrasilQAC-Click dataset, where each line represents a pair prefix/candidate query, its features, and the relevance signal (i.e. number of clicks). The JusBrasilQAC-Submission has the same structure with a different relevance signal. It is worth to say that the query auto-completion component from JusBrasil is error tolerant [Xiao et al., 2013], which means that the candidate queries generated by it might not always have a exact match with the prefix provided by the user.

3.2 Learning to Rank for Query Auto-completion

Learning to rank consists in the application of machine learning techniques in the construction of ranking models for systems such as document retrieval, collaborative filtering, sentiment analysis, online advertising, and query auto-completion [Liu, 2009].

In query auto-completion, learning to rank algorithms receive as input training data which consists of lists of candidate queries Q_p for each prefix p , with Q_p results receiving relevance signals used to induce their partial order. For convenience, the lists are represented as prefix/candidate query pairs which contain a set of ranking features. The goal of a learning to rank algorithm is to construct a ranking model to rank new, unseen lists in a similar way to rankings in the training data.

In our work, we experiment with four learning to rank algorithms: LambdaMART, RankSVM, XGBoost and Genetic Programming. These algorithms were chosen due to

their performance and facility to be deployed in query auto-completion components. According to Guo et al. [2020], LambdaMART is still considered a state of the art algorithm for learning to rank problems. RankSVM and XGBoost are competitive algorithms with respect to the LambdaMART, and can handle large-scale data. Finally, we choose the Genetic Programming algorithm, which is a competitive, lightweight and viable alternative to the others when the set of features is small. All four algorithms are described in the next sections.

3.2.1 LambdaMART Ranker

LambdaMART [Burgess, 2010] is a powerful and effective algorithm for ranking problems that has been widely used in industry and academia. Its ability to optimize for ranking metrics, along with its regularization techniques and early stopping criteria, make it a popular choice for search engines, recommendation systems, and other ranking applications. For example, it is the most used learning to ranking algorithm in query auto-completion [Shokouhi, 2013; Jiang et al., 2014; Mitra and Craswell, 2015; Cai and de Rijke, 2016; Jiang and Cheng, 2016; Park and Chiba, 2017; Fiorini and Lu, 2018; Jiang et al., 2018b; Kannadasan and Aslanyan, 2019]. In this section, we explain how LambdaMART can be used in query auto-completion.

The basic idea of LambdaMART is to train an ensemble of weak learners by using MART and approximate Newton step and linearly combining their predictions into a stronger and more effective learner [Burgess, 2010]. LambdaMART tunes the gradients of the regression trees based on a gradient-based optimization method. The gradient function is calculated according to the selected evaluation metric of effectiveness, and this metric is optimized directly in the training process of a ranking model.

A key concept of the LambdaMART algorithm is the gradient function λ which is defined as a smooth approximation to the gradient of a target cost with respect to the score of a candidate query. The gradient function quantifies the adjusted direction (up or down) and the force of a “to-be-sorted” candidate query in the next iteration. The two candidate queries of any given pair have the equal gradient values but opposite moving directions. The gradient of the candidate query in the positive direction pushes it toward the higher rank position of the sorted list, while the gradient of the candidate query in the negative direction pushes it toward the lower rank position of the sorted list.

LambdaMART algorithm optimizes the gradient λ_i of each candidate query $q_i \in Q_p$ for each prefix p . For a given prefix, if the relevance signal r_i between q_i and p is higher, and the ranked position of q_i is more closed to the bottom of the ranked list, then the positive value of λ_i indicates a push toward the top of the ranked list, and a bigger value of λ_i represents a stronger force. A similar reasoning applies to the opposite case.

The algorithm integrates the evaluation criteria of information retrieval (such as *DCG* [Järvelin and Kekäläinen, 2002]) into the gradient computation. The gradient λ_i for each candidate query q_i is obtained by the summation of all λ_{ij} over all pairs of $\langle q_i, q_j \rangle$ that q_i participates in for prefix p ; that is,

```

prefix,candidateQuery,MPC,dcg,totalClicks,clickSuggestionOnPrefix,averageWordDocCount,minTFFCount,
averageWordOccurrencesInEntities,frequencyOfEntity,editDistance,prefixSize,relevanceSignal
defesa impto,legitima defesa impronuncia,42,0.405624,1,0,2087.9008,23603,0.0,0,1,11,0
defesa impto,execucao fiscal defesa por negativa geral,21,0.077126,1,0,6019.9044,42503348,0.0,0,2,11,0
defesa impto,defesa importunacao sexual,104,0.0,1,1,2068047,99539,0.0,0,0,11,3
defesa impto,cerceamento de defesa aposentadoria especial,410,0.564475,1,0,4595958,18066145,0.0,0,2,11,0
defesa impto,carta de defesa mob inss,15,1.089279,2,0,3569.7356,249435,0.0,0,2,11,0
defesa lei,defesa de multa de transito,1880,1.039836,3,0,5602.9064,125071289,0.0,0,2,11,0
defesa lei,legitima defesa de terceiro configurada,894,1.550174,1,0,2548.1464,10819603,0.0,0,2,11,0
defesa lei,principio da ampla defesa e do contraditorio,557,1.490295,2,0,3442.5336,27803840,0.0,0,2,11,0
defesa lei,inovacao da tese de defesa em sede recursal,1758,2.486146,1,0,3672192,6898620,0.0,0,2,11,0
defesa lei,legitima defesa lei maria da penha,32,2.207287,5,0,7691.4344,1943267,0.0,0,0,11,0
defesa lei,defesa de multa de transito lei seca,55,1.586761,1,0,6895.2528,822119,0.0,0,2,11,0
defesa lei,ampla defesa e contraditorio,1575,2.463492,5,0,3036.4688,27803840,3036.4688,7,2,11,0
defesa lei,modelo defesa lei seca,36,1.484488,1,0,6093.9332,822119,0.0,0,0,11,1
defesa lei,manifestacao sobre defesa e documentos,997,0.915875,1,0,8530896,116305380,0.0,0,2,11,0
defesa lei,defesa em crimes sexuais,1926,2.010215,1,0,2309.3504,503089,0.0,0,2,11,0
defesa lei,defesa lei maria da penha,81,1.434868,2,0,9583.2992,8838473,0.0,0,0,11,0
defesa mar,excludente da legitima defesa art do cp,116,0.0,1,0,6014974,3280128,0.0,0,1,10,0
defesa mar,defesa procon,799,2.443998,2,0,3098122,1823443,0.0,0,2,10,0
defesa mar,defesa previa,29,1.941406,5,0,3181.7182,30838345,0.0,0,2,10,0
defesa mar,defesa previa multa,578,2.581296,3,0,4379394,30838345,0.0,0,2,10,0
defesa mar,defesa preliminar lei de drogas,464,1.484003,1,0,6760656,28285058,0.0,0,2,10,0
defesa mar,defesa maria da penha lesao corporal,46,0.956438,1,1,4201586,203456,0.0,0,0,10,3
defesa mar,defesa maria da penha ameaca,22,1.135934,3,2,5324.0216,8838473,0.0,0,0,10,1

```

Figure 2. Example of the JusBrasilQAC-Click dataset.

λ_i is computed as follows:

$$\lambda_i = \sum_{j:\{i,j\} \in I} \lambda_{ij} - \sum_{j:\{j,i\} \in I} \lambda_{ij}. \quad (2)$$

In Equation 2, λ_{ij} denotes the gradient of the pair $\langle q_i, q_j \rangle$ and is calculated as follows:

$$\lambda_{ij} = -\frac{\beta}{1 + e^{\beta \times (s_i - s_j)}} \times |\Delta M_{ij}|, \quad (3)$$

where β is a shape parameter for the sigmoid function, s_i and s_j represent the score assigned to q_i and q_j by the ranking model, respectively, and ΔM_{ij} represents the change on the effectiveness measure M by swapping the two queries q_i and q_j at the rank positions i and j accordingly (while keeping the rank positions of all other queries unchanged). The effectiveness evaluation criteria can be measured by any common information retrieval metric such as *DCG* [Järvelin and Kekäläinen, 2002]). ΔM_{ij} is calculated as follows:

$$\Delta M_{ij} = M_p - M_p^*, \quad (4)$$

where M_p denotes the effectiveness of the prefix p for a ranked list of all documents with respect to p , and M_p^* denotes the effectiveness of p after swapping q_i and q_j at the rank positions i and j for the ranked list. Thus, LambdaMART algorithm incorporates an effectiveness evaluation criteria into the gradient function in the training process of the ranking models, and this optimizes both the loss function and the effectiveness evaluation criteria.

After several learning trials, we tune the LambdaMART with 500 trees and a learning rate of 0.001 across all experiments reported in Section 4.

3.2.2 RankSVM Ranker

RankSVM is a commonly used pairwise ranking algorithm for search engines [Joachims, 2002]. In our work, we have adopted it for the query auto-completion scenario.

For the query auto-completion problem with n prefixes and a list of candidate queries Q_p for each prefix p , features $x_i \in \mathbb{R}^m$ are extracted from the pair prefix/candidate query (p_i, Q_{p_i}) and label $y_i \in \mathbb{R}$ is the relevance signal of the Q_{p_i} to the prefix p_i . The training data is a set of signal-prefix-instance tuples (y_i, p_i, x_i) . Let \mathcal{P} denote the set of preference pairs. If $(i, j) \in \mathcal{P}$, Q_{p_i} and Q_{p_j} are in the same prefix ($p_i = p_j$) and Q_{p_i} is preferred over Q_{p_j} ($y_i > y_j$). Thus, the goal of RankSVM is to get a ranking function:

$$f(x) = w^T x, \quad (5)$$

such that $\forall (i, j), f(x_i) > f(x_j) = w^T x_i > w^T x_j$, and $w \in \mathbb{R}^m$.

The RankSVM algorithm has a good generalization due to the margin-maximization property. According to Yu and Kim [2012], the margin is defined as the closest distance between two data points when the data points project to the ranking vector w :

$$d = \min \frac{w^T (x_i - x_j)}{\|w\|}, \forall (i, j) \in \mathcal{P}. \quad (6)$$

Maximizing the margin is good because data point pairs with small margins represent very uncertain ranking decisions. RankSVM can guarantee to find a ranking vector w with the maximum margin [Yu and Kim, 2012].

In our experiments (Section 4), we ran the RankSVM algorithm with the epsilon insensitive loss function, and a maximum of 1000 iterations with a tolerance for the stopping criteria of 0.0001.

3.2.3 XGBoost Ranker

XGBoost [Chen and Guestrin, 2016], which stands for Extreme Gradient Boosting, is a popular machine learning algorithm used for regression, classification and ranking problems. It has gained widespread popularity due to its scalability, efficiency, and accuracy. For this reason, we have decided to use it for query auto-completion. To the best of

our knowledge, this is the first time that such an algorithm is used for query auto-completion.

The XGBoost algorithm is a decision tree ensemble based on gradient boosting that builds an additive expansion of the objective function by minimizing a loss function. Boosting algorithms combine weak learners, i.e. learners slightly better than random, into a strong learner in an iterative way [Freund and Schapire, 1999]. Gradient boosting is a boosting-like algorithm for regression [Friedman, 2001].

In our work, we define the XGBoost algorithm as follows. Let $D = \{(x_i, y_i) : i = 1..n, x_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}$ be an input dataset with m features and n examples. In query auto-completions, an example is a pair prefix/candidate query. The goal of XGBoost is to find an approximation $\hat{F}(x)$ of the function $F^*(x)$, which maps instances x to their output values y , by minimizing the expected value of a given loss function $L(y, \hat{F}(x))$. The algorithm builds the additive approximation $\hat{F}(x)$ as a weighted sum of functions:

$$\hat{F}(x) = \sum_{k=1}^K \rho_k h_k(x_k), \quad (7)$$

where ρ_k is the weight of the k^{th} function $h_k(x_k)$, and x_k is a subset of the input variables. The K functions $h_k(x_k)$ are the models (i.e. the decision trees) of the ensemble.

XGBoost solves the Equation 7 by finding the best set of functions that minimize the following loss function:

$$L_{xgb} = \sum_{i=1}^n L(y_i, \hat{F}(x_i)) + \sum_{k=1}^K \Omega(h_k), \quad (8)$$

where L represents the loss function which is the difference between the predicted $\hat{F}(x_i)$ and the actual y_i outputs, and Ω is a measure of complexity of the model. This measure assists to avoiding overfitting of the model and is calculated as follows:

$$\Omega(h_k) = \gamma T + \frac{1}{2} \lambda \|w\|^2, \quad (9)$$

where T is the number of leaves of the tree and w are the output scores of the leaves.

The loss function can be integrated into the split criterion of decision trees leading to a pre-pruning strategy. Higher values of γ result in simpler trees. The value of γ controls the minimum loss reduction gain needed to split an internal node. An additional regularization parameter in XGBoost is shrinkage, which reduces the step size in the additive expansion. Finally, the complexity of the trees can also be limited using other strategies such as the depth of the trees. A secondary benefit of tree complexity reduction is that the models are trained faster and require less storage space.

After several learning trials, we tune the XGBoost with 100 trees and a learning rate of 0.1 across all experiments in Section 4.

3.2.4 Genetic Programming Ranker

Genetic Programming (GP) [Koza, 1990] is an extension of Genetic Algorithms (GA) for solving problems starting from a high-level statement. Instead of programming a model and

refine it for solving a particular problem, GP use an extensive search for automatically generate and find potential solutions.

The GP method has been successfully applied to ranking problems [Fan et al., 2004; Yeh et al., 2007; Yeh and Lin, 2017] by generating functions for evaluating the relevance of documents, allowing a set of documents to be ordered by their relevance. Since most of the query auto-completion systems apply a two steps framework for retrieval, and then ordering suggestions, GP can be applied on the second step as a learning to rank method.

The fundamental concept of GP is based on structuring potential solutions called “individuals” and defining functions similar to genetic operations to modify them. Initially, a population of N individuals is generated, then each solution is evaluated, producing a fitness value that gauges its efficacy in addressing the problem. The next step is apply the functions defined previously in the individuals and create a new population. Finally, each individual is evaluated and those with lower fitness are removed. This process is repeated until a stop criteria is matched.

In our work, since we are interested in generating a ranking function as output, the structure of the individuals are defined as equations trees, where each inner node is a basic arithmetic function $f = \{f_0, f_1, f_2, \dots, f_n\}$, such as addition, subtraction, multiplication and division, and the terminals nodes represent a feature value. By doing this, the structure could be compiled to a function that takes the suggestion features as input. The fitness of the individuals are defined as the mean of the DCG calculated for each pair prefix/candidate query.

In this case, the algorithm is trying to maximize the fitness of its individuals in order to generate better ranking functions. To tune the GP, we have used an initial population of 300 individuals, a 70% crossover rate, a 30% mutation rate, at most 10 of tree depth, tournament size of 3, and a ceiling of 40 generations. These values we obtained after several learning trials.

4 Offline Empirical Evaluation

In this section, we detail the empirical evaluation conducted in our work. In particular, we aim to answer four main research questions:

1. How each feature, specially the novel ones proposed in our work, impact the rankings in query auto-completion?
2. How effective is learning to rank with respect to the Most Popular Completion (MPC), a ranking algorithm widely adopted as baseline in the literature?
3. Among the four alternatives experimented, which learning to rank algorithm is more effective in the legal domain?
4. How effective is learning to rank with respect to ranking models based on BERT and ColBERT?

We start by describing the methodology and metrics used in our evaluation, and then, we discuss the main findings.

4.1 Methodology and Metrics

The empirical evaluation was carried out by using the 10-fold cross-validation methodology [Mitchell, 1997]. For query auto-completion, this methodology can be described as follows. The prefixes are split into 10 mutually exclusive partitions, in which 1 of these partitions is chosen as the testing set and the remaining are chosen as the training set. The testing set is chosen 10 times, without repeating the same partition. Finally, the results for the 10 partitions are averaged.

With respect to the evaluation metrics, we have adopted the Mean Reciprocal Rank (MRR) [Voorhees and Tice, 1999] and the Discounted Cumulative Gain (DCG) [Järvelin and Kekäläinen, 2002]. According to [Tahery and Farzi, 2020], more than 77% of the research works proposed in the literature have employed the MRR metric, so it has become a standard evaluation metric for query auto-completion. Additionally, we also evaluate our experiments employing the well known DCG metric. These metrics are defined as follows.

The Mean Reciprocal Rank is the expected reciprocal rank of the model on the testing dataset:

$$MRR = \frac{1}{|Q_T|} \sum_{q \in Q_T} \frac{1}{rank_q}, \quad (10)$$

where Q_T denotes the testing set, and $rank_q$ is the position of the ground truth q in the ranked auto-completion list.

The Discounted Cumulative Gain represents the usefulness or gain of the query based on its position in the ranked auto-completion list. The metric penalizes the relevance of the query logarithmically based on the position of the query in the ranked list. Notice that again we have decided to adopt DCG based on the conclusions presented on [Al-Maskari et al., 2007] which indicates that DCG better correlates with user satisfaction when compared with NDCG.

$$DCG = \sum_{i=1}^{|P_{Q_T}|} \frac{2^{rel_i} - 1}{\log_2(i + 1)}, \quad (11)$$

where i denotes the rank, $|P_{Q_T}|$ denotes the number of ranks to be generated for the testing set Q_T , and rel_i is the relevance signal of the query at rank i . In this paper, we compute the DCG metric for ranks of size from 1 (@1) to 12 (@12) candidate queries.

Finally, it is worth to say that all comparisons in our work were conducted by using the two sided paired t-test with a 95% confidence level [Mitchell, 1997].

4.2 Results

In this section, we discuss our main findings regarding the four research questions.

4.2.1 Analyzing the Impact of the Ranking Features

To answer our first question, and evaluate the impact that each ranking feature can bring to query auto-completion, we start by running a set of experiments comparing the results obtained with the learning to rank algorithms trained with all features against to the same learning algorithms trained by

removing one ranking feature each time, i.e. the feature that we will analyze in the respective experiment. The results are presented in Table 1, and Figures 3 and 4.

In Table 1, we can see the MRR values obtained with the four learning to rank algorithms (i.e. Genetic Programming, LambdaMART, RankSVM and XGBoost) trained with all features, and with minus one specific ranking feature each experiment run. There, we can see that the XGBoost in JusBrasilQAC-Click dataset, and the Genetic Programming in JusBrasilQAC-Submission dataset are the only algorithms that provided MRR values higher using all ranking features than by removing one feature. In other scenarios, we have better results by removing one feature. For JusBrasilQAC-Click dataset, the Genetic Programming obtained its best result by removing averageWordDocCount or frequencyOfEntity, with a gain of 6.12% with respect to the use of all features; the LambdaMART had its best result by removing the editDistance feature, obtaining a gain of 4.06%; and the RankSVM obtained its best result by removing minTF-FCount, with a gain of 26.16% with respect to the use of all features. However, the only gain that is statistically significant is the one obtained with the RankSVM algorithm. For JusBrasilQAC-Submission dataset, the LambdaMART obtained its best result by removing dcg, with a gain of 1.03% with respect to the use of all features; the RankSVM obtained a gain of 12.04% by removing minTF-FCount; and the XGBoost had its best result with a small gain of 0.15% by removing one of the following features: MPC, averageWordOccurrencesInEntities, frequencyOfEntity or minTF-FCount. Again, although we see some gains in the JusBrasilQAC-Submission dataset, they are not statistically significant.

On the other hand, other ranking features are very important to query auto-completion, and their removal can diminish the quality of the ranks. For example, by removing the clickSuggestionOnPrefix feature in the JusBrasilQAC-Click dataset, we obtained a reduction of -4.39% for Genetic Programming, -38.89% for LambdaMART, -33.15% for RankSVM, and -9.35% for XGBoost with respect to the use of all features. Similarly for JusBrasilQAC-Submission dataset, we obtained reductions of -2.32% for Genetic Programming (removing totalClicks), -4.15% for LambdaMART (removing editDistance), -26.28% for RankSVM (removing clickSuggestionOnPrefix) and -7.38% for XGBoost (removing editDistance). Most of these results are statistically significant as we can see in Table 1.

A similar comparison to measure the quality of the ranks by using the DCG metric is presented in Figures 3 and 4. In Figure 3, for the JusBrasilQAC-Click dataset, we have that the removal of features does not improve or diminish the results in most of the cases, according to the statistical t-test used to compare the removal of features with respect to the use of all features. The only exception is the removal of the clickSuggestionOnPrefix feature, which diminish the quality of the ranks for all algorithms. For the JusBrasilQAC-Submission dataset, although the Figure 4 shows some variations in the DCG values, they are not statistically significant.

Additionally, to better understand the impact of each feature in the learning process, we computed the feature importance using XGboost, method that is among the best in

Table 1. Comparing the MRR values obtained with learning to rank algorithms trained with all features, and removing one ranking feature, in the JusBrasilQAC-Click and JusBrasilQAC-Submission datasets. The highest values are in boldface, and the lowest ones are underlined. The symbol “*” indicates the values which are statistically significant with respect to the value for the allFeatures.

Algorithms	Features	JusBrasilQAC-Click		JusBrasilQAC-Submission	
		Mean	Std. Deviation	Mean	Std. Deviation
Genetic Programming	allFeatures	0.751	0.149	0.646	0.009
	-MPC	0.750	0.150	0.641	0.016
	-averageWordDocCount	0.797	0.011	0.638	0.016
	-averageWordOccurrencesInEntities	0.750	0.148	0.645	0.008
	-clickSuggestionOnPrefix	<u>0.718</u>	<u>0.014</u>	0.637	0.011
	-dcg	0.752	0.149	0.636*	0.010*
	-editDistance	0.776	0.013	0.596*	0.013*
	-frequencyOfEntity	0.797	0.011	0.642	0.010
	-minTFFCount	0.790	0.018	0.641	0.007
	-prefixSize	0.750	0.149	0.645	0.008
	-totalClicks	0.744	0.147	<u>0.631</u> *	<u>0.014</u> *
LambdaMART		Mean	Std. Deviation	Mean	Std. Deviation
	allFeatures	0.689	0.129	0.578	0.012
	-MPC	0.692	0.130	0.579	0.010
	-averageWordDocCount	0.696	0.132	0.579	0.011
	-averageWordOccurrencesInEntities	0.689	0.129	0.577	0.013
	-clickSuggestionOnPrefix	<u>0.421</u> *	<u>0.030</u> *	0.383*	0.014*
	-dcg	0.692	0.130	0.584	0.011
	-editDistance	0.717	0.140	<u>0.554</u> *	<u>0.010</u> *
	-frequencyOfEntity	0.689	0.129	0.579	0.012
	-minTFFCount	0.689	0.129	0.582	0.013
	-prefixSize	0.713	0.134	0.577	0.008
	-totalClicks	0.690	0.129	0.575	0.009
RankSVM		Mean	Std. Deviation	Mean	Std. Deviation
	allFeatures	0.558	0.110	0.548	0.070
	-MPC	0.499	0.157	0.480	0.075
	-averageWordDocCount	0.530	0.152	0.481	0.085
	-averageWordOccurrencesInEntities	0.486	0.168	0.462*	0.063*
	-clickSuggestionOnPrefix	<u>0.373</u> *	<u>0.094</u> *	<u>0.404</u> *	<u>0.056</u> *
	-dcg	0.500	0.154	0.501	0.091
	-editDistance	0.483	0.140	0.464*	0.064*
	-frequencyOfEntity	0.487	0.104	0.492	0.073
	-minTFFCount	0.704 *	0.044 *	0.614	0.074
	-prefixSize	0.457	0.149	0.489	0.087
	-totalClicks	0.547	0.071	0.472*	0.075*
XGBoost		Mean	Std. Deviation	Mean	Std. Deviation
	allFeatures	0.802	0.011	0.650	0.012
	-MPC	0.798	0.011	0.651	0.012
	-averageWordDocCount	0.801	0.012	0.649	0.013
	-averageWordOccurrencesInEntities	0.801	0.012	0.651	0.013
	-clickSuggestionOnPrefix	<u>0.727</u> *	<u>0.015</u> *	0.634*	0.012*
	-dcg	0.802	0.012	0.640	0.010
	-editDistance	0.776*	0.013*	<u>0.602</u> *	<u>0.014</u> *
	-frequencyOfEntity	0.801	0.012	0.651	0.012
	-minTFFCount	0.801	0.011	0.651	0.013
	-prefixSize	0.800	0.011	0.647	0.013
	-totalClicks	0.750	0.150	0.641	0.012

all experiments. The feature importance was evaluated according to the average gain of each feature while using to split nodes. Results are depicted in Figure 5, where we can see that some of the new features discussed here achieved high importance results. For example, the click-based feature clickSuggestionOnPrefix was the most important. This

was expected, since it represents a direct user feedback about the quality of suggestions previously presented to users. The editDistance was the second most important feature, which shows the importance of implementing approximate match in query auto-completion systems. This importance may be explained by the difficult that many users have to properly

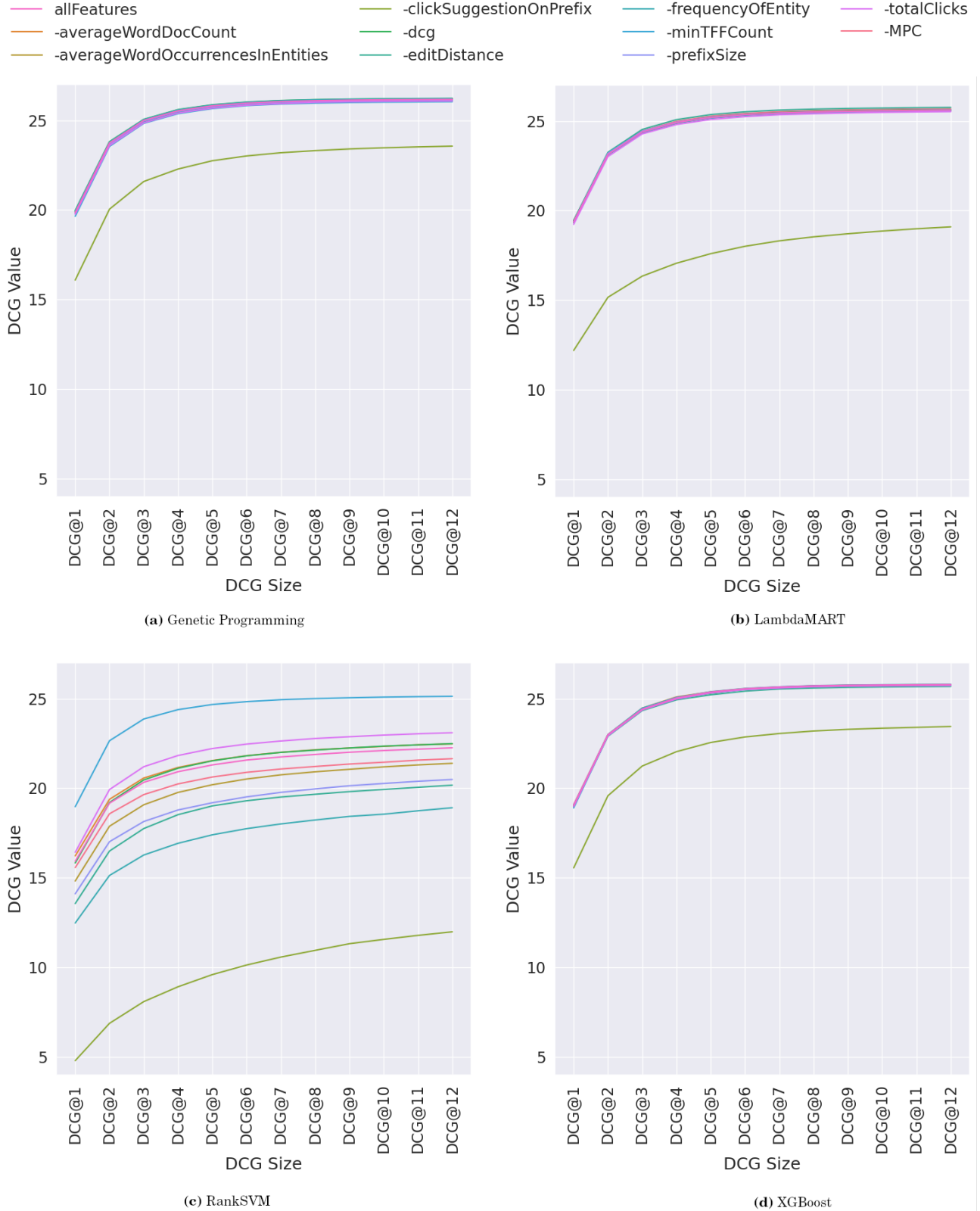


Figure 3. Comparing the DCG values obtained with learning to rank algorithms trained with all features, and removing one ranking feature each running time, in the JusBrasilQAC-Click dataset.

spelling some words. To give an example, the expression “habeas corpus”, which comes from Latin, appears in the query log with almost the same frequency as “habes corpus”. Notice also that importance results supports the conclusions found during the feature removals. The features clickSuggestionOnPrefix, editDistance, totalClicks and MPC perform

better than the other ones in our model.

In summary, the results show that our ranking features should be carefully analyzed before to be used, since they can affect positively or negatively the ranking of the candidate queries.

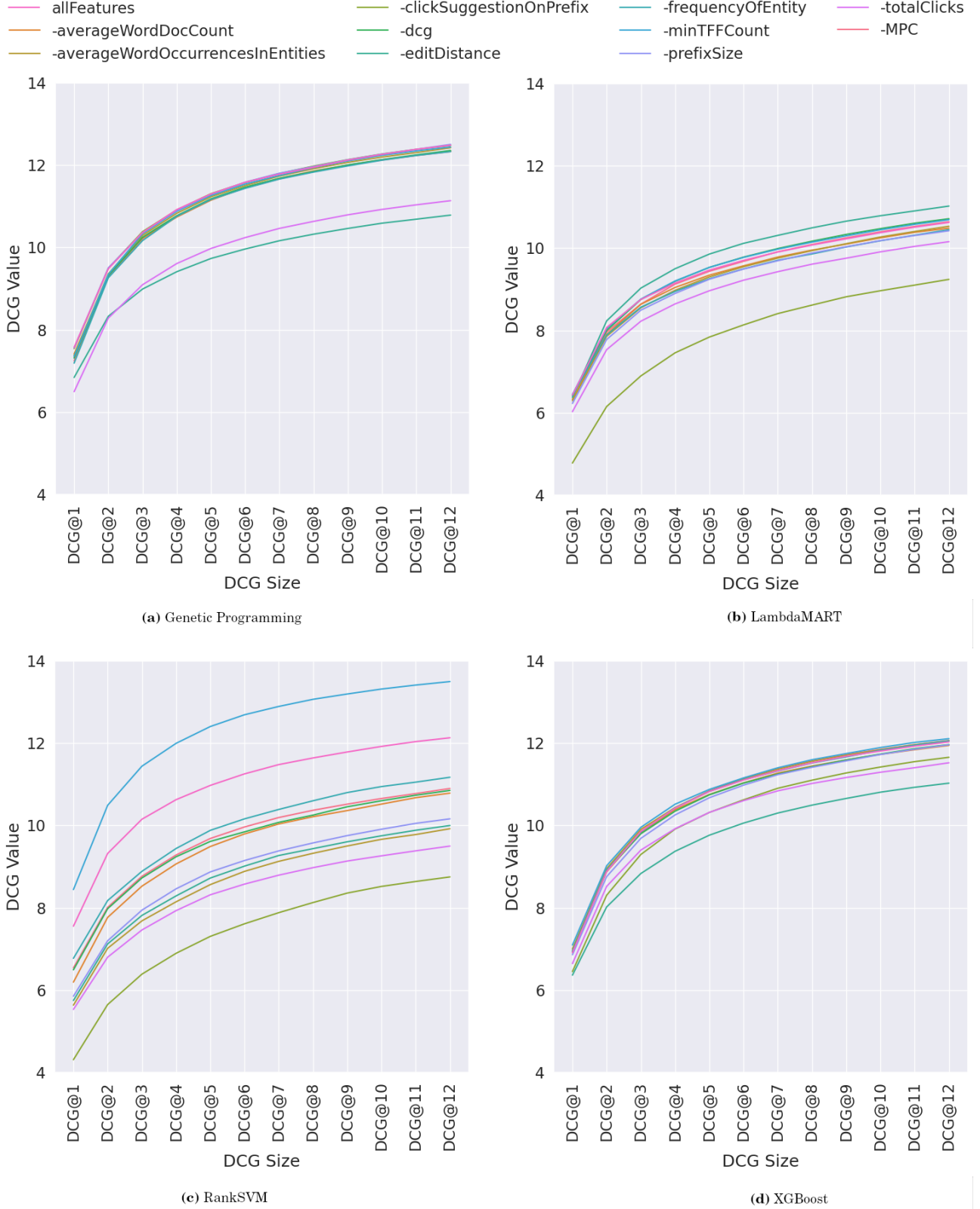


Figure 4. Comparing the DCG values obtained with learning to rank algorithms trained with all features, and removing one ranking feature each running time, in the JusBrasilQAC-Submission dataset.

4.2.2 Comparing Learning to Rank Against MPC Algorithm

In this section, we compare the four learning to rank algorithms against the MPC algorithm in order to answer our second research question. As already described in Section

3, the MPC was the first algorithm proposed for query auto-completion and it is used as baseline in the majority of the works presented in the literature. For this comparison, we trained the learning to rank algorithms with all features together.

In Figure 6, we can see that Genetic Programming, Lamb-

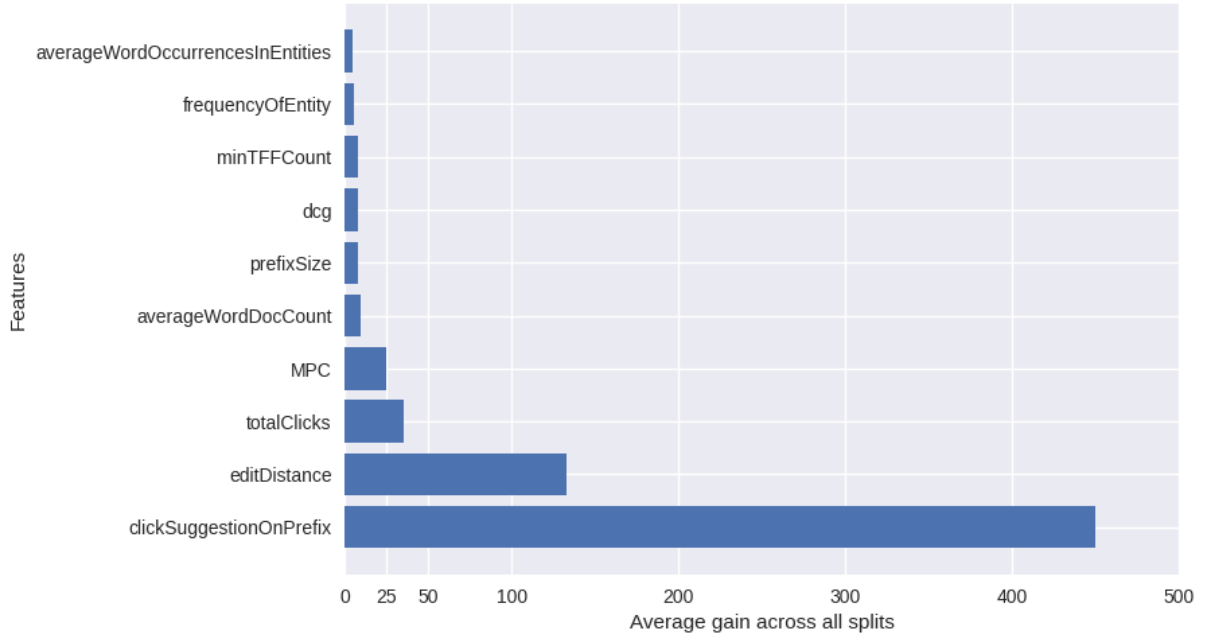


Figure 5. Comparing the average gain across all splits the which the feature is used in the XGBoost algorithm trained with all features in the JusBrasilQAC-Click dataset.

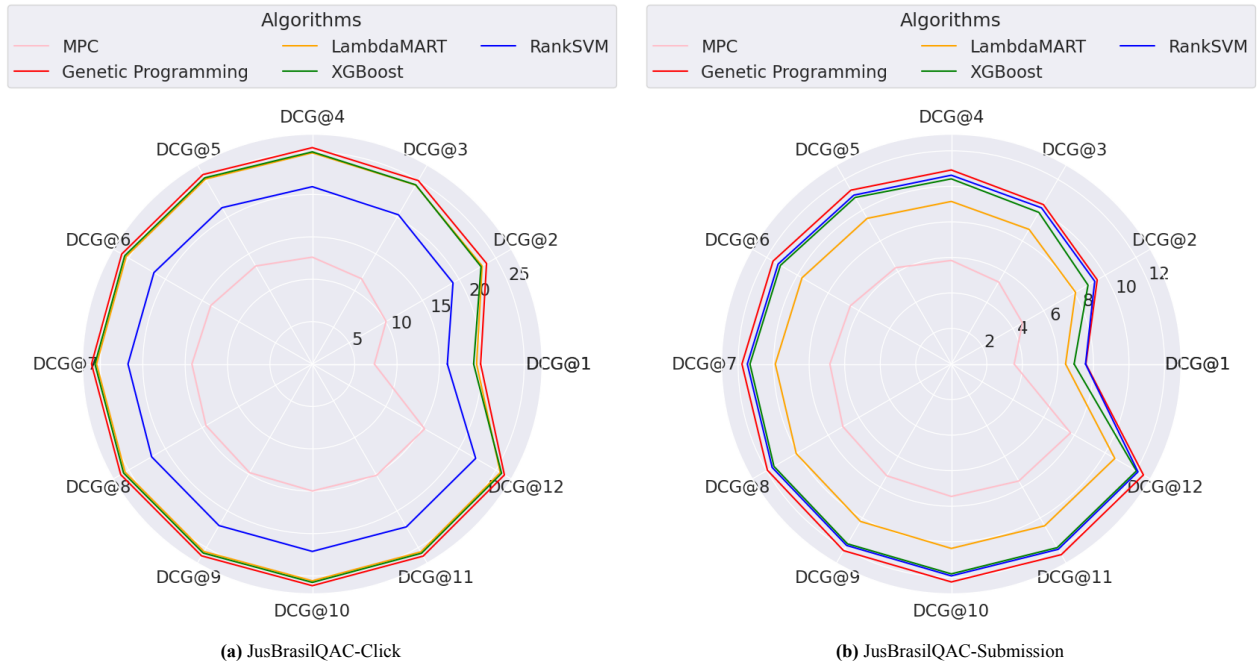


Figure 6. Comparing the learning to rank algorithms against to the MPC algorithm in terms of DCG for both datasets.

daMART, RankSVM and XGBoost algorithms provide better results than the MPC algorithm in both datasets. All these results are statistically significant according to the t-test applied in the comparison of each learning to rank algorithm against to the MPC. In Figure 6 (a), the rankings with 12 (@12) candidate queries generated by the learning to rank algorithms achieve improvements around 70% with respect to the MCP algorithm. A similar analysis for the JusBrasilQAC-Submission dataset (Figure 6 (b)) shows improvements of 61% with respect to the MCP algorithm. All these values are statistically significant.

Regarding the MRR metric (Table 2), we can also see that the four learning to rank algorithms provide better re-

sults than the MPC algorithm in both datasets. These results means the learning methods were able to bring the first relevant suggestions closer to the top in the ranking.

4.2.3 Analyzing the Learning to Rank Algorithms

We answer our third research question in this section by analyzing which learning to rank algorithm provides the best result for ranking in query auto-completion. Here, we also use the Figure 6 and Table 2 to carry out this analysis in terms of DCG and MRR metrics, but we compare only the learning to rank algorithms among them.

When looking to MRR results presented in Table 2, the

Table 2. Comparing the learning to rank algorithms against to the MPC algorithm in terms of MRR for both datasets. The symbol “*” indicates the values which are statistically significant with respect to the value for the MPC algorithm.

Algorithms	JusBrasilQAC-Click		JusBrasilQAC-Submission	
	Mean	Std. Deviation	Mean	Std. Deviation
MPC	0.367	0.104	0.348	0.016
Genetic Programming	0.751*	0.149*	0.646*	0.009*
LambdaMART	0.689*	0.129*	0.578*	0.012*
RankSVM	0.558*	0.110*	0.548*	0.070*
XGBoost	0.802*	0.011*	0.650*	0.012*

higher scores were achieved by XGBoost and Genetic Programming when applied to both experimented two collections, with a quite close performance in both cases, followed by LambdaMART and RankSVM with results slightly worse than XGBoost and Genetic Programming.

When considering the DCG results presented in Figure 6, we can see that Genetic Programming achieved the higher scores of DCG for both collections in the distinct top-k sizes experimented, but closely followed by the other experimented methods, which achieved almost similar performance. When looking to the scenario presented by both metrics and collections, we see that Genetic Programming and XGboost were among the best two alternatives for both collections, although closely followed by LambdaMART and RankSVM.

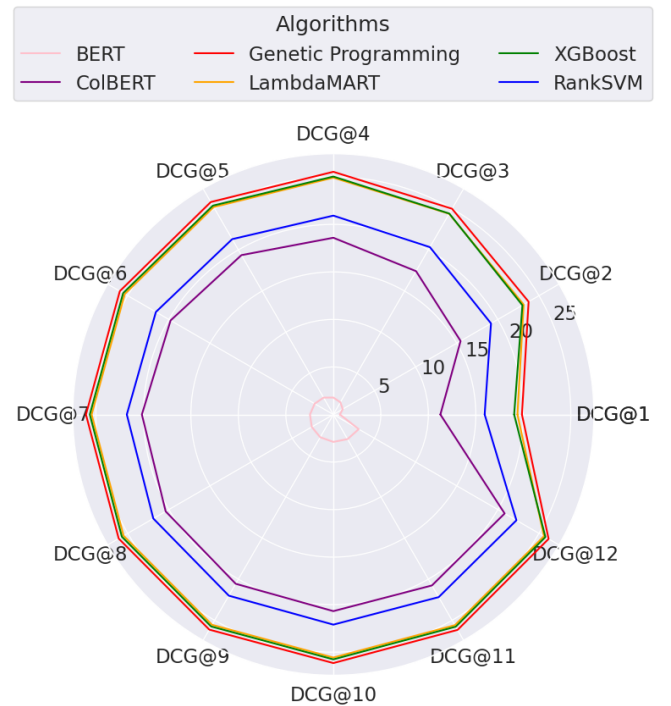
4.2.4 Comparing Learning to Rank Against to Ranking Models Using BERT and ColBERT

Finally, we answer our forth research question in this section by comparing the four learning to rank algorithms against ranking models based on BERT [Nogueira *et al.*, 2019] and the ColBERT [Khattab and Zaharia, 2020] for the JusBrasilQAC-Click dataset.

BERT (Bidirectional Encoder Representations from Transformers) is a language representation model that has impacted various natural language processing (NLP) tasks, including ranking [Lin *et al.*, 2021]. Using BERT to rank suggestions in auto-completion involves applying the model’s knowledge about the semantics of the language to assign relevance scores to query suggestions. The use of BERT for ranking query suggestions requires a fine tuning specially trained for this task. In these situations, its more interesting to adopt an extension of BERT specially designed for ranking tasks. We thus decided to also adopt a version of BERT adapted for ranking [Nogueira *et al.*, 2019] and ColBERT (Contextualized Late Interaction over BERT) in our experiments [Khattab and Zaharia, 2020]. ColBERT creates token based embeddings both from the prefix query and the query suggestions. The searcher module clusters embeddings of the query suggestions and each centroid becomes central in finding candidate suggestions when searching. The input prefix already typed by the user is similarly encoded and embedded, and ColBERT searches for the best suggestions applying a search on the dataset guided by the centroids for fast finding the best suggestions for the prefix query.

The results of our experiments are presented in Figure 7 and Table 3. With respect to the DCG metric, in Figure 7 we can see that the four learning to rank algorithms present better

results than the BERT and the ColBERT algorithms. The only exception is the RankSVM, which provides a similar performance with respect to the ColBERT, according to the statistical t-test.

**Figure 7.** Comparing the learning to rank algorithms against to BERT and ColBERT algorithms in terms of DCG for the JusBrasilQAC-Click dataset.

For the MRR metric, we can see in Table 3 that the four learning to rank algorithms also present better results than the BERT algorithm. With respect to the ColBERT algorithm, Genetic Programming and XGBoost provide superior ranking performance, RankSVM provides inferior performance, and LambdaMART provides a similar ranking performance, according to the statistical t-test.

As we have discussed, learning to rank has demonstrated high effectiveness for query auto-completion tasks, providing significant improvements over heuristic approaches such as the Most Popular Completion (MPC) algorithm. While BERT-based encoder representation models are powerful in capturing context and generating contextually rich completions, learning to rank frameworks such as LambdaMART, Genetic Programming, RankSVM, and XGBoost offer comparable or superior ranking performance, especially when tailored with features specific to the task. Furthermore, as highlighted in our study, the ability of learning to rank algorithms to integrate diverse feature sets, such as user interac-

Table 3. Comparing the learning to rank algorithms against to BERT and ColBERT algorithms in terms of MRR for the JusBrasilQAC-Click dataset. The symbol “*” indicates the values which are statistically significant with respect to the value for the BERT algorithm, and the symbol “†” indicates the values which are statistically significant with respect to the ColBERT algorithm.

Algorithms	Mean	Std. Deviation
BERT	0.113	0.110
ColBERT	0.642	0.013
Genetic Programming	0.751*†	0.149*†
LambdaMART	0.689*	0.129*
RankSVM	0.558*†	0.110*†
XGBoost	0.802*†	0.011*†

tion signals and document metadata, enhances the relevance of ranked suggestions. On the other hand, late interaction paradigms like those introduced in ColBERT allow for efficient query-document relevance evaluation without sacrificing effectiveness, making them competitive with BERT-based models [Khattab and Zaharia, 2020; Lin *et al.*, 2021].

Another important aspect of query auto-completion systems is that these systems need to be fast when processing queries, since it receives a query for each character typed by each user interacting with the search system, and it needs to provide an answer before the next character is typed. When comparing the four learning to rank methods experimented, the server with Genetic Programming was able to answer a query with 1.61ms on average, LambdaMART presented the second best performance, with 1.71ms, while XGBoost achieved 57.76ms, and RankSVM achieved 4163.24ms on average. With respect to the encoder representation models, ColBERT achieved 166.97ms, and BERT achieved 41826.84ms on average.

Thus, in the next section, we perform an A/B test to compare MPC and our approach using machine learning. Given the necessity of a good balance of quality of results and fast processing queries at query auto-completion problems, we have adopted the Genetic Programming algorithm, since it produced the best combination of quality and time performance. Genetic Programming was the fastest option and the second best in quality. As future work, we plan to further investigate more about the performance issues related to each method in order to see whether it is possible to improve them.

5 Online A/B Test

In addition to the offline evaluation, we also conduct two A/B tests to assess the impact of studied features when results are exposed to users in production. We adopted the Genetic Programming algorithm in the A/B tests, since processing the model was faster than processing the models produced by the other two methods. Besides, while we recognize that experimenting with the other models would be interesting, this was the only option available for the online A/B experiments, since the architecture of the system where the tests were executed was not designed for the other two models. Further, as Genetic Programming performed quite close to XGBoost in the experiments, still the online experiment with it worth for better understand the impact of the studied features in the experiments.

The A/B tests were conducted by sending 5% of the search traffic to the production system, and other 5% of traffic to the variant. The goal in each experiment is to measure the click rate on the suggestions presented by the query auto-completion component in the search box. The A/B test was deployed online for one month at Jusbrasil.

In the first A/B test, the variant consists of two Genetic Programming functions trained by removing frequencyOfEntity from the set of ranking features, and using the number of clicks in the candidate query as relevance signal. This training was chosen to generate both functions since it provided the best result according to our empirical evaluation. One function is trained by removing only the frequencyOfEntity, and the other one is trained by removing the frequencyOfEntity and the clickSuggestionOnPrefix features. The later is also removed from the training of the second function because this feature is considered a strong relevance signal, and it is not wide available for all the candidate queries. Thus, if we have values for clickSuggestionOnPrefix, the variant uses the first function. Otherwise, it uses the second function.

The control in the first test was the MPC set of features, so we have an online evaluation to assert the online impact of using the best set of features among the one studied by us when compared to a system using only the features available on MPC.

The results show a click rate of 0.044% for the MPC (i.e. the control), and a click rate of 6.600% for the variant. This value represents an improvement of 14,900% of the variant with respect to the MPC. A significant improvement on the variant can also be seen on Precision (i.e. the fraction of the suggestions retrieved that are relevant to the users) in Table 4 and on MRR in Figure 8. The poor performance of MPC could be partially explained by the distribution of queries on Jusbrasil, a large part of which is on the long tail, thus reducing the effectiveness of strategies based solely on popularity. As we expected, the MPC was outperformed by our variant on metrics of user interaction.

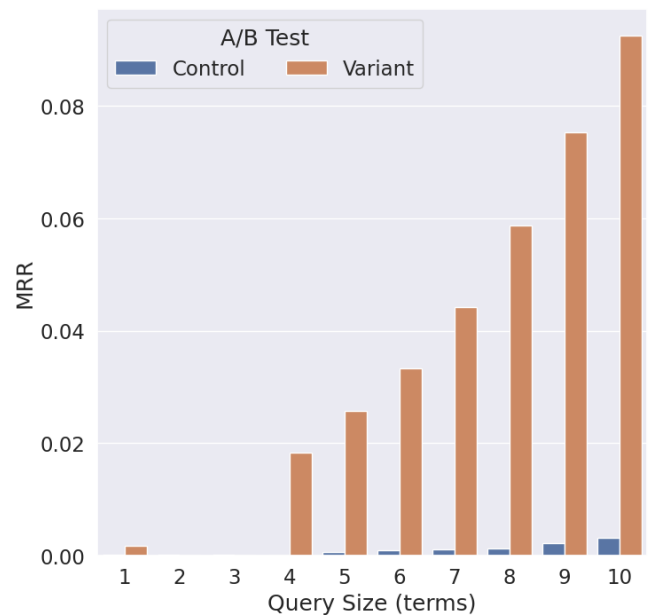


Figure 8. Comparing the MRR value between the control and the variant in the first A/B test.

Table 4. Comparing the Precision values (P@N) between the control and the variant in the first A/B test.

Metric	Test	Query Size (terms)									
		1	2	3	4	5	6	7	8	9	10
P@1	Control	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Variant	0.001	0.000	0.000	0.011	0.017	0.023	0.031	0.043	0.056	0.071
P@2	Control	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Variant	0.001	0.000	0.000	0.017	0.024	0.032	0.043	0.057	0.074	0.092
P@3	Control	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Variant	0.002	0.000	0.000	0.021	0.029	0.037	0.050	0.066	0.084	0.104
P@4	Control	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Variant	0.002	0.000	0.000	0.025	0.033	0.042	0.056	0.073	0.093	0.113
P@5	Control	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
	Variant	0.003	0.000	0.000	0.028	0.038	0.046	0.061	0.079	0.099	0.120
P@6	Control	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002
	Variant	0.004	0.000	0.000	0.031	0.041	0.050	0.065	0.084	0.105	0.126
P@7	Control	0.001	0.001	0.001	0.001	0.002	0.002	0.002	0.002	0.002	0.002
	Variant	0.004	0.000	0.000	0.033	0.043	0.053	0.068	0.088	0.110	0.131
P@8	Control	0.001	0.001	0.001	0.002	0.002	0.002	0.003	0.003	0.004	0.004
	Variant	0.005	0.000	0.000	0.034	0.045	0.056	0.071	0.091	0.114	0.136
P@9	Control	0.001	0.002	0.003	0.003	0.004	0.004	0.004	0.004	0.004	0.005
	Variant	0.005	0.000	0.000	0.036	0.047	0.058	0.073	0.094	0.117	0.139
P@10	Control	0.002	0.002	0.003	0.003	0.004	0.005	0.007	0.007	0.007	0.007
	Variant	0.005	0.000	0.000	0.037	0.049	0.060	0.076	0.097	0.121	0.142

Table 5. Comparing the Precision values (P@N) between the control and the variant in the second A/B test.

Metric	Test	Query Size (terms)									
		1	2	3	4	5	6	7	8	9	10
P@1	Control	0.001	0.000	0.000	0.012	0.018	0.023	0.033	0.043	0.056	0.069
	Variant	0.001	0.000	0.000	0.011	0.017	0.023	0.031	0.043	0.056	0.071
P@2	Control	0.002	0.000	0.000	0.019	0.028	0.033	0.045	0.058	0.076	0.092
	Variant	0.001	0.000	0.000	0.017	0.024	0.032	0.043	0.057	0.074	0.092
P@3	Control	0.002	0.000	0.000	0.023	0.033	0.039	0.052	0.067	0.086	0.105
	Variant	0.002	0.000	0.000	0.021	0.029	0.037	0.050	0.066	0.084	0.104
P@4	Control	0.003	0.000	0.000	0.027	0.038	0.044	0.058	0.073	0.094	0.114
	Variant	0.002	0.000	0.000	0.025	0.033	0.042	0.056	0.073	0.093	0.113
P@5	Control	0.003	0.000	0.000	0.030	0.041	0.048	0.063	0.078	0.100	0.121
	Variant	0.003	0.000	0.000	0.028	0.038	0.046	0.061	0.079	0.099	0.120
P@6	Control	0.003	0.000	0.000	0.032	0.044	0.050	0.066	0.082	0.104	0.126
	Variant	0.004	0.000	0.000	0.031	0.041	0.050	0.065	0.084	0.105	0.126
P@7	Control	0.004	0.000	0.000	0.034	0.046	0.053	0.069	0.086	0.109	0.130
	Variant	0.004	0.000	0.000	0.033	0.043	0.053	0.068	0.088	0.110	0.131
P@8	Control	0.004	0.000	0.000	0.036	0.048	0.055	0.072	0.089	0.113	0.135
	Variant	0.005	0.000	0.000	0.034	0.045	0.056	0.071	0.091	0.114	0.136
P@9	Control	0.005	0.000	0.000	0.038	0.050	0.057	0.074	0.092	0.116	0.139
	Variant	0.005	0.000	0.000	0.036	0.047	0.058	0.073	0.094	0.117	0.139
P@10	Control	0.005	0.000	0.000	0.039	0.052	0.059	0.077	0.095	0.119	0.142
	Variant	0.005	0.000	0.000	0.037	0.049	0.060	0.076	0.097	0.121	0.142

The second A/B test, uses the same variant adopted in the first, but the control uses the full set of features. In this second test, during the evaluation period, we obtained a click rate of 6.474% for the production system (i.e. the control) using the full set of features, and a click rate of 6.600% for the variant. The MRR in Figure 9 and Precision in Table 5 also have close values between these two variants, however our variant has a tendency to bring better results as the size of the prefix increase, and this behaviour could explain the reduction on the average prefix size from control to our

variant. These data demonstrate an improvement of the variant with respect to the control using the full set of features, and reinforces that results obtained in the offline experiment, which indicates results are better when removing the frequencyOfEntity from the set of features.

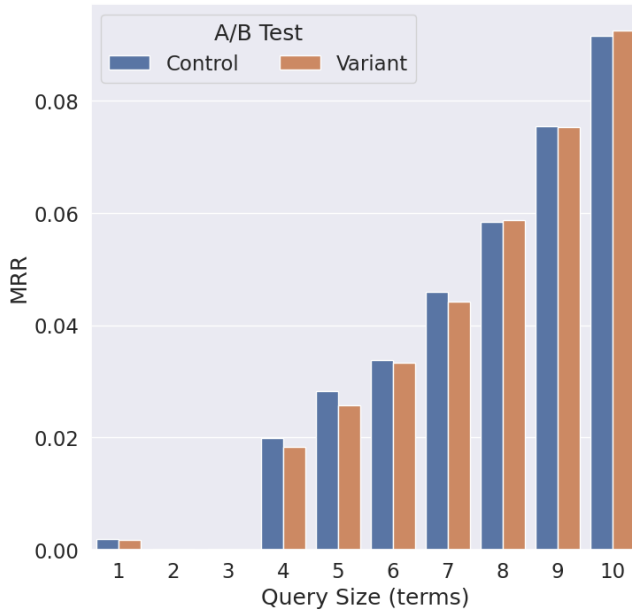


Figure 9. Comparing the MRR value between the control and the variant in the second A/B test.

6 Conclusion and Future Work

In this paper, we explored four learning to rank algorithms for query auto-completion in the legal domain: LambdaMART, RankSVM, XGBoost and Genetic Programming. Additionally, we also introduced two datasets with different types of features for query auto-completion in the legal domain to foster research on this area. Finally, we showed that learning to rank is effective for query auto-completion in the legal domain.

Among our findings, we highlight the following:

- The proposed ranking features from our datasets can improve the ranking query auto-completion in the legal domain;
- The four learning to ranking algorithms evaluated in our work provide better results than the widely used baseline MPC heuristic algorithm;
- The four learning to ranking algorithms present close quality of results;
- Although the ranking based on the ColBERT model seems promise, the learning to rank algorithms are still superior to it;
- The Genetic Programming algorithm, with our proposed features, can be a good candidate to be used in the query auto-completion component at Jusbrasil when considering both processing times and quality of results.

With respect to the limitations of our work, we can mention the difficulties in collecting query auto-completion datasets, in particular, for the legal domain. It is also expected that the features proposed in our work will be useful in other domains, but we were unable to validate this because we do not have datasets for such experiments. This fact comes from the difficulty of obtaining such datasets. Another limitation is that we were unable to include all learning to rank algorithms in the A/B tests. We aim to address the limitations of our work in the future.

For future work, we plan to generate our ranking features in other domains and evaluate them for query-auto completion. Moreover, we also plan to compare other learning to rank algorithms in the query-auto completion task to obtain more evidences about our findings. Further, the results achieved with ColBERT indicate that it can be used in a future work as a feature to be included in learning algorithms. The study of ColBERT to be used as a feature requires however further investigation about alternative ways of producing such feature offline to avoid time performance restrictions and the high computational costs related to the use of ColBERT. Finally, we also plan to run an A/B test to evaluate the query auto-completion component with other learning to rank algorithms.

Declarations

Acknowledgements

This research is partially supported by Jusbrasil; by the IPDEC Institute; by FAPEAM under the POSGRAD 2024/2025 Program and the NeuralBond Project (UNIVERSAL 2023 Proc. 01.02.016301.04300/2023-04); by the Coordination for the Improvement of Higher Education Personnel-Brazil (CAPES) financial code 001; and by CNPq under Project IAIA (406417/2022-9) and individual grants from CNPq to Altigran da Silva (307248/2019-4) and Edleno Moura (310573/2023-8). The authors sincerely thank the legal domain specialists from Jusbrasil for their invaluable input in this work.

Authors' Contributions

All authors contributed to the study conception and design. Data for the experimental research was provided by LR and EM. Material preparation, data preparation, experiments and analysis were performed by MD. The A/B test was conducted by LR. The article was written by all authors. All authors read and approved the article.

Competing interests

The authors declare that they do not have competing interests

Availability of data and materials

The company may provide access to the datasets for researchers under the restrictions of Brazilian law of user data protection. Instructions for use and proper future citation of the datasets will be provided by email (datasets@jusbrasil.com.br).

References

- Al-Maskari, A., Sanderson, M., and Clough, P. (2007). The relationship between ir effectiveness measures and user satisfaction. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 773–774. DOI: 10.1145/1277741.1277902.

- Bar-Yossef, Z. and Kraus, N. (2011). Context-sensitive query auto-completion. In *Proceedings of the 20th International Conference on World Wide Web, WWW '11*, page 107–116, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/1963405.1963424.
- Block, A., Kidambi, R., Hill, D. N., Joachims, T., and Dhillon, I. S. (2022). Counterfactual learning to rank for utility-maximizing query autocompletion. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '22*, page 791–802, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3477495.3531958.
- Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, Microsoft Research. Available at: <https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/>.
- Cai, F. and Chen, H. (2017). Term-level semantic similarity helps time-aware term popularity based query completion. *J. Intell. Fuzzy Syst.*, 32(6):3999–4008. DOI: 10.3233/JIFS-151404.
- Cai, F. and de Rijke, M. (2016). Learning from homologous queries and semantically related terms for query auto completion. *Information Processing & Management*, 52(4):628–643. DOI: 10.1016/j.ipm.2015.12.008.
- Cai, F. and de Rijke, M. (2016). A survey of query auto completion in information retrieval. *Found. Trends Inf. Retr.*, 10(4):273–363. DOI: 10.1561/15000000055.
- Cai, F., Liang, S., and de Rijke, M. (2016). Prefix-adaptive and time-sensitive personalized query auto completion. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2452–2466. DOI: 10.1109/TKDE.2016.2568179.
- Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. DOI: 10.1145/2939672.2939785.
- da Costa Carvalho, A. L., Rossi, C., de Moura, E. S., da Silva, A. S., and Fernandes, D. (2012). Lepref: Learn to precompute evidence fusion for efficient query evaluation. *Journal of the American Society for Information Science and Technology*, 63(7):1383–1397. DOI: 10.1002/asi.22665.
- Di Santo, G., McCreadie, R., Macdonald, C., and Ounis, I. (2015). Comparing approaches for query autocompletion. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15*, page 775–778, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2766462.2767829.
- Fan, W., Gordon, M. D., and Pathak, P. (2004). A generic ranking function discovery framework by genetic programming for information retrieval. *Information Processing & Management*, 40(4):587–602. DOI: 10.1016/j.ipm.2003.08.001.
- Ferreira, B., de Moura, E. S., and Silva, A. d. (2022). Applying burst-tries for error-tolerant prefix search. *Information Retrieval Journal*, 25(4):481–518. DOI: 10.1007/s10791-022-09416-9.
- Fiorini, N. and Lu, Z. (2018). Personalized neural language models for real-world query auto completion. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 208–215, New Orleans - Louisiana. Association for Computational Linguistics. DOI: 10.18653/v1/N18-3026.
- Freund, Y. and Schapire, R. E. (1999). A short introduction to boosting. *Journal of Japanese Society of Artificial Intelligence*, 14(5):771 – 780. Available at: <https://www.yorku.ca/gisweb/eats4400/boost.pdf>.
- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232. DOI: 10.1214/aos/1013203451.
- Guo, J., Fan, Y., Pang, L., Yang, L., Ai, Q., Zamani, H., Wu, C., Croft, W. B., and Cheng, X. (2020). A deep look into neural ranking models for information retrieval. *Information Processing Management*, 57(6):102067. DOI: 10.1016/j.ipm.2019.102067.
- Hu, Y. H., Xiao, C., and Ishikawa, Y. (2018). Context-sensitive query auto-completion with knowledge base. In *The 10th Forum on Data Engineering and Information Management (the 16th Annual Meeting of Database Society of Japan)*, pages 1–5. Available at: <https://db-event.jp/jn/deim2018/data/papers/149.pdf>.
- Jaeck, A. and Ostendorf, M. (2018). Personalized language model for query auto-completion. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 700–705, Melbourne, Australia. Association for Computational Linguistics. DOI: 10.18653/v1/P18-2111.
- Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446. DOI: 10.1145/582415.582418.
- Jiang, D., Cai, F., and Chen, H. (2018a). Location-sensitive personalized query auto-completion. In *2018 10th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 01, pages 15–19. DOI: 10.1109/IHMSC.2018.00012.
- Jiang, D., Cai, F., and Chen, H. (2018b). Personalizing query auto-completion for multi-session tasks. In *2018 IEEE International Conference on Computer and Communication Engineering Technology (CCET)*, pages 203–207. DOI: 10.1109/CCET.2018.8542201.
- Jiang, D., Chen, W., Cai, F., and Chen, H. (2018c). Neural attentive personalization model for query auto-completion. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 725–730. DOI: 10.1109/IAEAC.2018.8577694.
- Jiang, J.-Y. and Cheng, P.-J. (2016). Classifying user search intents for query auto-completion. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval, ICTIR '16*, page 49–58, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2970398.2970400.
- Jiang, J.-Y., Ke, Y.-Y., Chien, P.-Y., and Cheng, P.-J. (2014). Learning user reformulation behavior for query auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR '14*, page 445–454, New York,

- NY, USA. Association for Computing Machinery. DOI: 10.1145/2600428.2609614.
- Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 133–142, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/775047.775067.
- Kang, Y. M., Liu, W., and Zhou, Y. (2021). Query-blazer: efficient query autocompletion framework. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 1020–1028. DOI: 10.1145/3437963.3441725.
- Kannadasan, M. R. and Aslanyan, G. (2019). Personalized query auto-completion through a lightweight representation of the user context. In *Proceedings of the SIGIR 2019 Workshop on eCommerce, co-located with the 42st International ACM SIGIR Conference on Research and Development in Information Retrieval, eCom@SIGIR 2019, Paris, France, July 25, 2019*, pages 1–8. DOI: 10.48550/arXiv.1905.01386.
- Khattab, O. and Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3397271.3401075.
- Kim, G. (2019). Subword language model for query auto-completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5022–5032, Hong Kong, China. Association for Computational Linguistics. DOI: 10.18653/v1/D19-1507.
- Koza, J. R. (1990). *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, volume 34. Stanford University, Department of Computer Science Stanford, CA. Available at: <http://infolab.stanford.edu/pub/cstr/reports/cs/tr/90/1314/CS-TR-90-1314.pdf>.
- Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4:87–112. DOI: 10.1007/BF00175355.
- Li, L., Deng, H., Chen, J., and Chang, Y. (2017a). Learning parametric models for context-aware query auto-completion via hawkes processes. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, page 131–139, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3018661.3018698.
- Li, L., Deng, H., Dong, A., Chang, Y., Baeza-Yates, R., and Zha, H. (2017b). Exploring query auto-completion and click logs for contextual-aware web search and query suggestion. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 539–548, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee. DOI: 10.1145/3038912.3052593.
- Li, L., Deng, H., Dong, A., Chang, Y., Zha, H., and Baeza-Yates, R. (2015). Analyzing user's sequential behavior in query auto-completion via markov processes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 123–132, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2766462.2767723.
- Li, Y., Dong, A., Wang, H., Deng, H., Chang, Y., and Zhai, C. (2014). A two-dimensional click model for query auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, page 455–464, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2600428.2609571.
- Lin, J., Nogueira, R. F., and Yates, A. (2021). *Pre-trained Transformers for Text Ranking: BERT and Beyond*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers. DOI: 10.2200/S01123ED1V01Y202108HLT053.
- Liu, T.-Y. (2009). Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331. DOI: 10.1561/15000000016.
- Maxwell, D., Bailey, P., and Hawking, D. (2017). Large-scale generative query autocompletion. In *Proceedings of the 22nd Australasian Document Computing Symposium*, ADCS 2017, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3166072.3166083.
- Mitchell, T. M. (1997). *Machine learning*, volume 1. McGraw-hill New York. Book.
- Mitra, B. (2015). Exploring session context using distributed representations of queries and reformulations. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 3–12, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2766462.2767702.
- Mitra, B. and Craswell, N. (2015). Query auto-completion for rare prefixes. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 1755–1758, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2806416.2806599.
- Nogueira, R. F., Yang, W., Cho, K., and Lin, J. (2019). Multi-stage document ranking with BERT. *CoRR*, abs/1910.14424. DOI: 10.48550/arXiv.1910.14424.
- Park, D. H. and Chiba, R. (2017). A neural language model for query auto-completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1189–1192, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3077136.3080758.
- Shao, T., Chen, H., and Chen, W. (2018). Query auto-completion based on word2vec semantic similarity. *Journal of Physics: Conference Series*, 1004(1):012018. DOI: 10.1088/1742-6596/1004/1/012018.
- Shokouhi, M. (2013). Learning to personalize query auto-completion. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, page 103–112, New York,

- NY, USA. Association for Computing Machinery. DOI: 10.1145/2484028.2484076.
- Shokouhi, M. and Radinsky, K. (2012). Time-sensitive query auto-completion. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, page 601–610, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2348283.2348364.
- Tahery, S. and Farzi, S. (2020). Customized query auto-completion and suggestion — a review. *Information Systems*, 87:101415. DOI: 10.1016/j.is.2019.101415.
- Voorhees, E. M. and Tice, D. M. (1999). The TREC-8 question answering track evaluation. In *Proceedings of The Eighth Text REtrieval Conference, TREC 1999, Gaithersburg, Maryland, USA, November 17-19, 1999*. Available at: <http://trec.nist.gov/pubs/trec8/papers/qa8.pdf>.
- Wang, P., Zhang, H., Mohan, V., Dhillon, I. S., and Kolter, J. Z. (2018). Realtime query completion via deep language models. In *The SIGIR 2018 Workshop On eCommerce co-located with the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2018)*, Ann Arbor, Michigan, USA, July 12, 2018. Available at: <https://openreview.net/forum?id=By3VrbbAb>.
- Wang, S., Guo, W., Gao, H., and Long, B. (2020). Efficient neural query auto completion. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2797–2804. DOI: 10.1145/3340531.3412701.
- Wang, Y., Ouyang, H., Deng, H., and Chang, Y. (2017). Learning online trends for interactive query auto-completion. *IEEE Trans. on Knowl. and Data Eng.*, 29(11):2442–2454. DOI: 10.1109/TKDE.2017.2738639.
- Whiting, S. and Jose, J. M. (2014). Recent and robust query auto-completion. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, page 971–982, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2566486.2568009.
- Xiao, C., Qin, J., Wang, W., Ishikawa, Y., Tsuda, K., and Sadakane, K. (2013). Efficient error-tolerant query autocompletion. *Proc. VLDB Endow.*, 6(6):373–384. DOI: 10.14778/2536336.2536339.
- Yeh, J.-Y. and Lin, J.-Y. (2017). Learning ranking functions for information retrieval using layered multi-population genetic programming. *Malaysian Journal of Computer Science*, 30(1):27–47. DOI: 10.22452/mjcs.vol30no1.3.
- Yeh, J.-Y., Lin, J.-Y., Ke, H.-R., and Yang, W.-P. (2007). Learning to rank for information retrieval using genetic programming. In *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval (LR4IR 2007)*. Citeseer. Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f83659a19e6373231a7609d13accf27555fd6ed8>.
- Yu, H. and Kim, S. (2012). *SVM Tutorial — Classification, Regression and Ranking*, pages 479–506. Springer Berlin Heidelberg, Berlin, Heidelberg. DOI: 10.1007/978-3-540-92910-9_15.
- Zhang, A., Goyal, A., Kong, W., Deng, H., Dong, A., Chang, Y., Gunter, C. A., and Han, J. (2015). Adaqac: Adaptive query auto-completion via implicit negative feedback. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 143–152, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2766462.2767697.