# Query Autocompletion

Knowledge Discovery and Data Mining 2 VU (706.715)
Summer Term 2019

Karmen Gostiša
karmen.gostisa@student.tugraz.at

## ABSTRACT

This paper describes the implementation and evaluation of a query autocompletion (QAC) mechanism, also known as autocomplete suggestion. Given a prefix containing one or more characters entered in a search box, the system returns a list of suggestions to complete the prefix to a full query. The paper starts with a brief introduction explaining motivation and historical background of the topic. Afterwards it describes the practical approach, going from choosing and parsing dataset to creating QAC. Furthermore, evaluation approach is presented and results of developed and baseline QAC systems are provided. In the end conclusions are drawn.

## 1. INTRODUCTION

### 1.1 Motivation

Word completion has been around as a feature of word editors and command shells for almost a half of century. It is a convenient feature that offers the user a list of words after one or more letters have been typed. The most typical usage is in the field of information retrieval, that provides a list of suggested queries when users begin to enter their query in the search box. User's incomplete input is often known as a *query prefix* and suggested queries as *query completions*. Query autocompletion (QAC) mechanism helps formulating queries when users do not have a clear "image" of a query in advance. It also helps avoiding spelling mistakes and saves time. However, QAC is not only used as a web search engine but also in other services for search tasks, for example, YouTube provides QAC for searching videos; Facebook has a QAC that is employed for finding friends, events, pages and other content; and Twitter for searching tweets [1].

### 1.2 Historical background

The original idea of QAC comes from more than half a century ago, when Longuet-Higgins and Ortony [2] discussed a method to help decrease the number of keystrokes needed to complete a word, specifically commands entered by developers. In that manner, QAC is actually a form of *word prediction*: when first letter or letters of a word are written, a word predictor lists possible words of choices that can be selected; or in some other form, the most likely following words are listed.

Early work on word prediction aimed both on evaluating the cognitive benefits and algorithmic development. It was found that time saved by reduction of keystrokes through word prediction was often compensated by time spent for going through the list of predicted words and selecting the desired one [3]. The optimal and reasonable balance between keystroke saving and cognitive load was displaying five predicted words in a vertical form [4]. Algorithmic work focused on predicting characters, completing words or combinations of the two. Character predictors reduce time spent on typing input by making more likely letters faster to select. Word completing systems offer words based on the user input. Combined approaches do both, for example, the *Reactive Keyboard*, is a prototypical example of word completion in early 1990s. Its word completion was in the beginning based only on a standard dictionary but then extended to adaptive modelling that produced word completions based on previously entered text. Similarly, user's previously entered text [5] and long-term search history [6] have been taken into account, although not implemented into a word completion system. Word completion techniques mostly use a special tree structure used to match the input and its completions, that is similar to the one some QAC techniques use. Both kind of techniques incorporate simple lexical models based on $n$-grams, extracted directly from collection of texts or query log [7].

In the early 2000s, Raskin [8] discussed the use of word prediction in which users get instant feedback as they enter a query. Such word prediction is also called an *incremental search* or *real-time suggestions* and it is actually a traditional search interface, consisting of three steps: submitting a query, system computing a result page, user receiving the result page. Since then, QAC has been used by web browsers, web sites, operating systems, databases, email clients and search engines.

## 2. PRACTICAL APPROACH

QAC was implemented using Solr, a search platform built on Apache Lucene [9]. This platform was chosen because it is open-source, reliable, scalable and fault tolerant, providing full-text indexing, search and auto-suggest feature. Implementation steps are described in the following Subsections.

## 2.1 Choosing dataset

To build an English based QAC, a proper dataset has to be chosen as its starting point. We used a Simple English Wikipedia, containing 117,527 articles. The dump [10] is available in BZ2 archive format and its extraction yields a single XML file with the size of 1.13GB. We chose this dataset because of its diversity on topics, scientific as well as unscientific ones. An English dictionary, for example, would not be a good choice for QAC as it is only a collection of English words.

## 2.2 Parsing dataset file

The XML file was first processed using a command provided by Gensim [11] that streams through all the XML articles, decompressing and extracting plain text from articles and their sections, putting it in JSON format. Afterwards, we wrote a script to process plain text data, namely removing redundant characters such as asterisks, single quotes and triple equality sign. A new final JSON file was produced, containing clean data fields describing Wiki's article: `ID`, `title` and `content`. Following is the first JSON line from the dataset (with shortened `content` field):

```
{
"id": 1,
"title": "April",
"content": "Introduction April is the 4th month
of the year, and comes between March and May.
It is one of four months to have 30 days. April
always begins on the same day of week as July,
and additionally, January in leap years..."
}
```

## 2.3 Creating a collection and indexing

JSON data file created in the previous step was added to Solr's collection, representing Simple English Wikipedia dataset. A very important document that was modified in this step, was XML schema that stores the details about the fields and field types Solr is expected to understand. Four fields were added: `ID`, `title`, `content` and `titleAndContent`. The last one is a copy field, a field that has two sources (title and content) and is used for indexing.

## 2.4 Configuring an autocomplete component

Solr provides Suggester, a component for automatic suggestions for query terms. Its main parameters are:

- **Dictionary implementation:** specifying how terms are stored in the suggestion dictionary;
- **Lookup implementation:** specifying how terms are found in the suggestion dictionary;
- **Field:** a field from the index to use as the basis of suggestion terms;
- **Analyzer:** specifying what tokenizers and filters are applied at index-time and query-time.

Some dictionary and lookup implementation require additional parameters to be configured.

Our solution uses a standard document dictionary containing terms taken from the index. The lookup strategy looks at the last $N$ tokens (configurable parameter) plus the prefix of the final token the user is typing, to predict the most likely next token. Returned suggestions are $n$-gram tokens. Analyzer carries out the same analysis both at index and query-time as follows:

- **Stopword filter:** Following stopwords taken from Lucene's StopAnalyzer were removed: a, an, and, are, as, at, be, but, by, for, if, in, into, is, it, no, not, of, on, or, such, that, the, their, then, there, these, they, this, to, was, will, with.
- **Standard tokenizer:** Text was split into tokens, treating whitespace and punctuation as delimiters.
- **Lowercase filter**.

## 2.5 Implementing the web application

To see how proposed solution worked in practice, a web application was developed, containing short description and a search bar. As we typed in the bar, 20 auto-suggestions were shown. In the backend, typing in the field invoked a function that requested data from the Solr server. AJAX was used in order to enable updating suggestions menu without reloading the whole page.

## 3. EVALUATION AND RESULTS

In this Section we first discuss our evaluation approach. Next, several rank-based metrics used for QAC evaluation are described. Finally, we present the results.

## 3.1 Approach

QAC system is good if it returns user's intended query at the top of the list, even when only short input has been provided. A lot of QAC research is based on a *query log* that contains records with at least three main components: a submitted query, a user ID and a timestamp. The first can be used to generate query prefixes, while the other two are used for extracting information about user and time. There are not many publicly available query logs as sharing personal queries may lead to personal information leakage so to evaluate QAC developed in this project, top 100 Google search queries in the US (as of April 2019) [12] were used instead. For each query, a random query prefix was generated. Random generator produced a random offset number so that prefix was at least three characters long and not longer than a half of a final query rounded up. For example, possible prefix queries generated for the query “weather” are “wea” and “weat”. Generated prefix was then given to QAC and 20 suggestions were retrieved.

To get a better understanding of how good developed QAC is, the best option is to compare it with other existing QAC mechanisms. In our case that was Google Suggest.

## 3.2 Metrics

### *Mean Reciprocal Rank (MRR)*

A standard measure to evaluate the effectiveness of QAC rankings has become Mean Reciprocal Rank (MRR) [13]. It is associated with a user model where the user only wishes to see one relevant document, in our case one relevant query completion. Representing the average of reciprocal ranks, it is computed as follows:

$$MRR = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{rank_i} \qquad (1)$$

$rank_i$ refers to a rank position of the $i$-th final query submitted by user. For example, user enters query prefix “know” and the system retrieves queries completions in next order: *knowledge*, *knowledge discovery*, *know your meme*. User chooses the third one, marking *know your meme* as the final query so the reciprocal rank for this query is $\frac{1}{3}$. Fig. 1 depicts MRR value in dependency of an average rank position of queries.

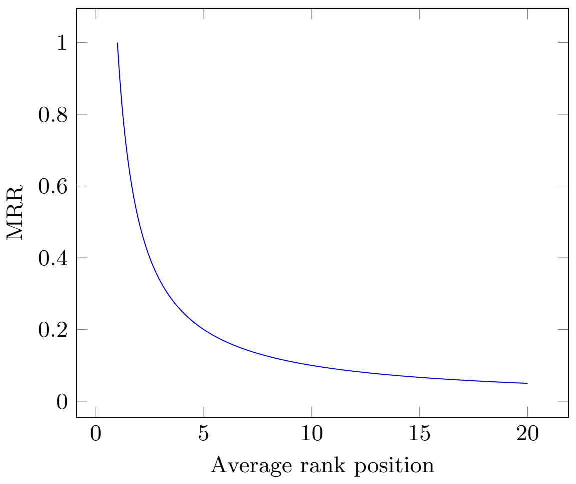

**Figure 1: A chart depicting MRR depending on average rank position of queries.**

### *Success rate at top K (SR@K)*

Although MRR is great for evaluating QAC results where only few top items are relevant, another metric is introduced, i.e. *success rate at top K (SR@K)*. This metric presents the average ratio of the final query that is found in the top $K$ query completion suggestions. Two values were chosen for $K$, namely 5 and 10.

### *Average overlap score (AOS)*

This metric measures a similarity score between two ranked lists. It upgrades the idea of set intersection with the concept of rank. In general, it determines the ratio of overlapping content at different depths. For example, two lists are compared in Table 1, showing fraction of elements overlapping at various depths. Average overlap score is computed as the average of the last column.

| Depth | List A | List B | Set intersection | Fraction |
|---|---|---|---|---|
| 1 | a | c | {} | 0 |
| 2 | a,b | c,b | {b} | 1/2 |
| 3 | a,b,c | c,b,f | {b,c} | 2/3 |
| 4 | a,b,c,d | c,b,f,d | {b,c,d} | 3/4 |
| 5 | a,b,c,d,e | c,b,f,d,g | {b,c,d} | 3/5 |

**Table 1: Fraction of elements from List A and B overlapping at various depths.**

This kind of approach naturally gives more importance to elements at higher ranks as a common element at lower depth contributes to all further set intersections. It is therefore good for assessing how similar two rank-based lists are.

## 3.3 Results

The results of previously described evaluation metrics are shown in Table 2.

| | Our solution | Google Suggest |
|---|---|---|
| MRR | 0.205 | 0.510 |
| SR@5 (%) | 29 | 61 |
| SR@10 (%) | 36 | 67 |
| AOS (%) | 13 | |

**Table 2: Evaluation metrics values for our solution and Google Suggest.**

Developed QAC provided MRR value 0.205, meaning that final queries were on average on 5th position. Google’s MRR equals to 0.510, meaning final queries were on average on 2nd position. Our solution put 29% of final queries in the top 5 suggestions and 36% in the top 10 suggestions. Success rates at top $K$ queries are higher on Google Suggest, being 61% for top 5 and 67% for top 10. The AOS for both lists is 13%.

## 4. CONCLUSION

This paper started with a short introduction of the query autocompletion topic, describing motivation and historical background. Afterwards, our QAC implementation was presented, along with the evaluation approach, metrics and results.

The results our solution achieved look very promising, especially being compared to Google Suggest, a very popular search system. Evaluation showed our QAC put final queries on average on 5th position, meanwhile Google Suggest put them on 2nd. Even though the 5th position is not an ideal one, it still comes very high in a list, enabling user to relatively fast find the desired query completion.

Several Solr's lookup implementations were tested during development, but the final decision was to only keep the best-performing one. Many other implementations also work in a different way as they provide suggestions based on the *whole* content of the field from the index, that was specified to use as the basis of suggestion terms. In our approach that field contains title of the article as well as its content. Providing both as suggestions would be irrational. We tried to use only title as the basis of suggestion terms but the results were not promising. By this, we learnt that Solr Suggester is good for providing suggestions based on short text field, for example, movie titles, people names, locations and similar.

The solution would also apply to a basic search scenario as once dataset is indexed, queries can be run using Solr Admin UI. This scenario, however, requires the user has recreated all our implementation steps and has his own instance of Solr up and running.

In summary, I gathered a lot of new insights in the field of Information retrieval. I learnt how to process large dataset, clean the data and prepare it in a useful form for solving query autocompletion problem. It was also the first time I used Apache Solr search platform and after initial struggles getting it up and running, I learnt a lot new about indexing and using Solr for search as well as query autocompletion.

## 5. REFERENCES

[1] F. Cai, M. De Rijke, *et al.*, "A survey of query auto completion in information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 10, no. 4, pp. 273–363, 2016.

[2] H. C. Longuet-Higgins and A. Ortony, "The adaptive memorization of sequences," *Machine Intelligence*, vol. 3, 1968.

[3] G. Vanderheiden and D. Kelso, "Comparative analysis of fixed-vocabulary communication acceleration techniques," *Augmentative and Alternative Communication*, vol. 3, no. 4, pp. 196–206, 1987.

[4] A. Swiffin, J. Arnott, J. A. Pickering, and A. Newell, "Adaptive and predictive techniques in a communication prosthesis," *Augmentative and Alternative Communication*, vol. 3, no. 4, pp. 181–191, 1987.

[5] Z. Bar-Yossef and N. Kraus, "Context-sensitive query auto-completion," in *Proceedings of the 20th international conference on World wide web*, pp. 107–116, ACM, 2011.

[6] F. Cai and M. de Rijke, "Selectively personalizing query auto-completion," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 993–996, ACM, 2016.

[7] F. Cai and M. de Rijke, "Learning from homologous queries and semantically related terms for query auto completion," *Information Processing & Management*, vol. 52, no. 4, pp. 628–643, 2016.

[8] J. Raskin, *The humane interface: new directions for designing interactive systems*. Addison-Wesley Professional, 2000.

[9] "Apache Solr 8.1.1." `https://lucene.apache.org/solr/`. [Online; accessed June 17, 2019].

[10] "Simple English Wikipedia dumps." `https://dumps.wikimedia.org/simplewiki/`. [Online; accessed June 17, 2019].

[11] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, (Valletta, Malta), pp. 45–50, ELRA, May 2010.

[12] "Top 100 Google search queries in the US (as of April 2019)." `https://ahrefs.com/blog/top-google-searches/`. [Online; accessed June 17, 2019].

[13] N. Craswell, *Mean Reciprocal Rank*, pp. 1703–1703. Boston, MA: Springer US, 2009.