Chapter 7
Query Auto-Completion

Liangda Li, Hongbo Deng, and Yi Chang

Abstract Search assist plays an important role in modern search engines to reduce users' search efforts and satisfy their information needs. Query auto-completion (QAC) is among one of the key search assist services, which help users type less while submitting a query. The QAC engine generally offers a list of suggested queries that start with a user's input as a prefix, and the list of suggestions is changed to match the updated input after the user types each keystroke. In this chapter, we formally introduce the definition of the QAC problem and present state-of-the-art QAC methods. More specifically, how the user's search intent can be predicted by exploring rich information, including temporal, contextual, personal, and underlying various search behaviors. We also describe the popular datasets and metrics that are utilized in evaluating the performance of QAC methods.

7.1 Problem Definition

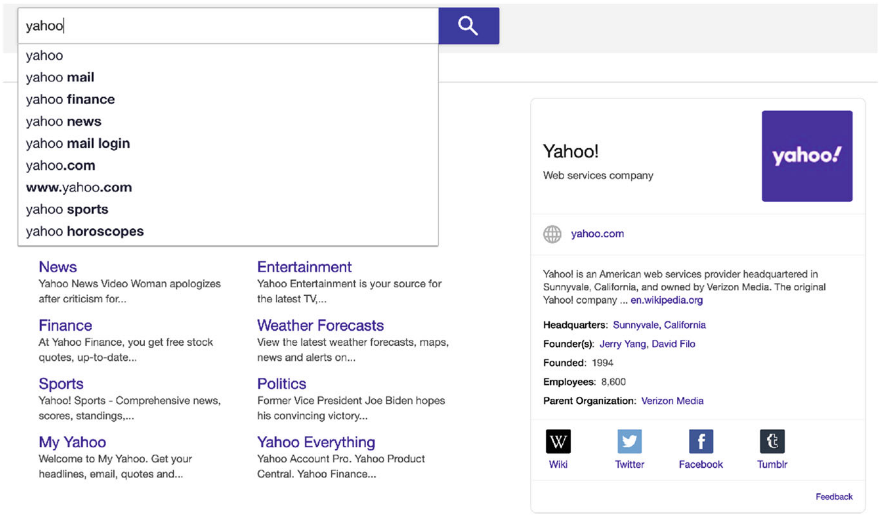
Query auto-completion (QAC) has been widely used in modern search engines to reduce users' efforts to submit a query by predicting the users' intended queries. The QAC engine generally offers a list of suggested queries that start with a user's input as a prefix, and the list of suggestions is changed to match the updated input after the user types each character. Suppose that a user is going to submit a query q to the search engine, and the user types the prefix of the query q of length i as $q[1..i]$ sequentially. The QAC engine will return the corresponding suggestion list

L. Li (✉)
Yahoo Research, Sunnyvale, CA, USA
e-mail: liangda@yahoo-inc.com

H. Deng
Alibaba Group, Zhejiang, China
e-mail: hbdeng@acm.org

Y. Chang
Jilin University, Jilin, China
e-mail: yichang@jlu.edu.cn

Y. Chang, H. Deng (eds.), *Query Understanding for Search Engines*,
The Information Retrieval Series 46, https://doi.org/10.1007/978-3-030-58334-7_7


yahoo|

yahoo
yahoo **mail**
yahoo **finance**
yahoo **news**
yahoo **mail login**
yahoo**.com**
www.yahoo**.com**
yahoo **sports**
yahoo **horoscopes**

News
Yahoo News Video Woman apologizes after criticism for...

Entertainment
Yahoo Entertainment is your source for the latest TV,...

Finance
At Yahoo Finance, you get free stock quotes, up-to-date...

Weather Forecasts
View the latest weather forecasts, maps, news and alerts on...

Sports
Yahoo! Sports - Comprehensive news, scores, standings,...

Politics
Former Vice President Joe Biden hopes his convincing victory...

My Yahoo
Welcome to My Yahoo. Get your headlines, email, quotes and...

Yahoo Everything
Yahoo Account Pro. Yahoo Product Central. Yahoo Finance...

Yahoo!
Web services company

yahoo.com

Yahoo! is an American web services provider headquartered in Sunnyvale, California, and owned by Verizon Media. The original Yahoo! company ... en.wikipedia.org

Headquarters: Sunnyvale, California
Founder(s): Jerry Yang, David Filo
Founded: 1994
Employees: 8,600
Parent Organization: Verizon Media

Wiki Twitter Facebook Tumblr

Feedback

Fig. 7.1 Example of query auto-completion

after the user types each character until user clicks the suggestion q from the list or presses return, ending the interaction with the QAC engine. Figure 7.1 shows an example of the QAC service from the Yahoo search engine.

In the following, we give a formal introduction of the query auto-completion (QAC) problem. Let p denote the prefix entered by a user u, and $C(p)$ denote the set of query completions that start with the prefix p, the output of a QAC method is $R(p)$, a ranking of a subset of queries from $C(p)$. Provided that the actual search intent of user u is query q, and a loss function $L(q, R(p))$ to measure how likely user u will click query q from the selected order query set $R(p)$. (Obviously, if $q \notin R(p)$, there is no click chance.) The target of a QAC method is to optimize the loss function $L(q, R(p))$ as:

$$\hat{R}(p) = \min_{R(p) \subset C(p)} L(q, R(p)) \tag{7.1}$$

Notice that $R(p)$ is an ordered set, different $R(p)$ can have the exact same set of queries with different rankings.

Typical loss functions prefer the q to be ranked as top position as possible in the ranking list $R(p)$, since normally a user prefers his/her intent query to be ranked as higher position as possible.

7.2 Evaluation Metrics for QAC

To evaluate the effectiveness of QAC methods, two main categories of metrics have been developed and explored: (1) metrics that focus on the quality of ranking and (2) metrics that focus on how user's effort in using QAC is saved.

7.2.1 Ranking Metrics

Since the output of a QAC method is a ranking of limited number of selected query candidates given the current prefix to best satisfy user's search intent, a good QAC method is supposed to rank the query that better satisfies user's intent in higher positions. As for the search intent judgment, different strategies were used in the literature:

- Using user's final submitted query in a QAC session. For instance, if a user clicks "facebook" among the queries in the suggestion list, "facebook" is regarded as the query that satisfies the user's real search intent. This is the most popular evidence used for the relevance judgment.
- Using user's submitted query's frequency within the most recent time slot. Such relevance judgment prefers query ranking suggestions that represent the search trend of general users instead of the unique search intent of individual users.
- Using manual judgments for each suggestion [3]. The major drawbacks of this strategy are that: (1) it requires a large amount of human resources for conducting the judgment, while the size of the data is usually limited to thousands of examples only; (2) the correctness of the judgment is usually not guaranteed, which can result in strong noise to the model training considering the limited data size of the editorial data.
- Using the quality of the search results retrieved by each suggested query [21]. Such relevance judgments benefit a search engine user who does not have a clear search intent before starting a QAC session, by suggesting him/her the most promising queries (with the best quality search results). However, the quality of search results is out of the control of a QAC engine, and a suggestion with better search results does not necessarily meet user's real search intent. Such a measurement tends to recommend user popular queries, thus fails to satisfy users who are searching tail queries which have a limited number of high quality search results, or the search engine itself performs poorly in indexing the high quality search results for them.

With the search intent judgment, traditional ranking metrics in information retrieval(IR) have been widely employed to measure the performance of QAC methods. Below, we list some popularly used measures.

- Mean reciprocal rank (MRR): This is a statistical measure that evaluates processes predicting a list of possible responses to a sample of queries. MRR is the most popular evaluation metric in measuring QAC performance [1, 19, 26],

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}, \tag{7.2}$$

where Q is the set of correct corresponds which, in our case, the query q a user finally submitted, and rank_q denotes the rank of the query q in the suggested query list. This evaluation setup assumes that items placed towards the top of a ranked list receive more attention and are therefore more useful to a search engine user.

Since most existing QAC works conducted experiments on normal QAC logs, which contained the query suggestion list under the last prefix of a QAC session only, MRR is calculated as the average reciprocal rank (RR) score of the last keystroke of each QAC session. Variations of MRR include:

- MRR@All: As introduced above, the normal MRR score only pays attention to the ranking of query suggestions under the last keystroke of a QAC session. However, in a real QAC scenario, a user is very likely to make the click at a shorter keystroke if his/her intended query is already shown at a reasonable position under that keystroke. Thus QAC methods that target to optimize the normal MRR score may fail to improve the query suggestion ranking at shorter keystrokes as well, while such an improvement can significantly save user's QAC action effort.

  Recently, the availability of high-resolution QAC data enabled the measurement of the quality of query suggestions at shorter keystrokes. A variation of the normal MRR score is proposed, named MRR@All, to calculate the average reciprocal rank (RR) score of all keystrokes, instead of the last keystroke only. Compared with the normal MRR score, such a variation prefers QAC methods that are able to infer user's real search intent as early as possible in a QAC session. To differentiate this variation from the normal MRR score, the normal score is named MRR@Last in those QAC works.
- Weighted mean reciprocal rank (wMRR): The normal MRR score assigns an equal weight to the last keystroke of each QAC session. However, one thing that is worth attention is that the effort of typing a specific prefix can also be different. For instance, if the user input is the letter "z," since there is only limited number of words that start with "z," the number of candidates to suggesting and ranking is also limited, which makes it a relatively easier task for a QAC model than the letter with larger number of candidates, such as "d." Thus, a weighted version of MRR, named weighted mean reciprocal rank (wMRR) [1] is proposed to each prefix based on the number of query suggestions available.

• Success Rate at top K (SR@K): This metric calculates the average ratio of the query that satisfied user's search intent can be found within the top K positions of the predicted query suggestion list. It is widely used for tasks that have only one ground truth among all candidates[10].

The major difference between ranking metrics in web document ranking and QAC problems is that, the judgment of the query that satisfies user's real search intent is relatively easier than the relevance judgment of web documents given the search query. A user's search intent in one QAC session is most likely the query that he/she finally submitted, while the relevance of web documents can hardly be determined by a user's click or dwell time on them. In learning to ranking tasks, editorial judgments of query-document relevance are very critical in measuring the performance, while QAC metrics rarely rely on the editorial effort. Such an advantage enables the collection of a large-scale golden evaluation dataset for the QAC tasks.

7.2.2 User Assist Metrics

Since the intuition of QAC is to assist search engine users' query formulation and save their interaction efforts, a good QAC method is supposed to reduce the cost of users' interaction with the search engine. Below we list some popularly used measures.

- Minimum Keystroke Length (MKS) [9]: It measures the number of actions a user has to take to submit a target query. This metric can be understood as a simulation of a search engine user's behavior during a QAC session. The user action taken into consideration includes both the letter typing and Down Arrow key pressing to reach the position of the target query. For instance, for the target query of a user that is located at the i-th position at the j-th keystroke, the number of actions will be calculated as $i + j$. Among all the potential positions in which the target query appears, the minimal number of actions needed will be counted as the value of MKS.

  A variation of the MKS metric is penalized Minimum Keystroke Length (PMKS), which considers an additional action, user's view of each suggestion for correctness. A penalty value of 0.1 is added for showing each suggestion, i.e., the latter keystroke a target query locates at, the larger penalty value is added. Such a variation can be view as an encouragement of users to make selections at shorter keystrokes.
- e-Saved and p-Saved [15]: p-Saved is proposed to compute the expected QAC usage as:

$$\text{pSaved}(q) = \sum_{i=1}^{|q|} \sum_{j} I(S_{ij}) P(S_{ij} = 1) = \sum_{i=1}^{|q|} \sum_{j} P(S_{ij} = 1) \tag{7.3}$$

where $P(S_{ij} = 1)$ measures the probability that a user ends the current QAC session at the j-th position under the i-th keystroke. And $I(S_{ij}) = 1$ when user actually used the corresponding query suggestion (at the j-th position under the i-th keystroke). This metric can be understood as the probability that a user actually uses the QAC engine rather than typing the target query on his/her own.

Conversely, e-Saved is proposed to measure the amount of effort saved in terms of keystrokes as:

$$\text{eSaved}(q) = \sum_{i=1}^{|q|}(1 - \frac{i}{|q|}) \sum_{j} P(S_{ij} = 1) \tag{7.4}$$

This metric is actually calculating the expected ratio of characters a user can skip inputting until his/her query is submitted. It prefers the improvements in the query suggestions for longer queries in particular, since a user usually prefers the effort saving benefit from a QAC engine when submitting long queries than the benefit when submitting short queries.

7.3 QAC Logs

Most of the research works on QAC built models based on the normal search query log. Traditionally, the search query log only includes the user ID, the timestamp, the submitted query and its associated search results. While the content of submitted queries in the log lays the foundation of search intent prediction for in general during the QAC process, other information like the timestamp, the submitted query and its associated search results provid more precise evidence for the search intent prediction given a certain user under a certain scenario. Typical public query log that is widely used in existing QAC works includes: the AOL dataset [24], the MSN dataset [8], and the SogouQ dataset.[1]

Those normal search query logs do not contain the sequential keystrokes (prefixes) users typed in the search box, as well as their corresponding QAC suggestions. In order to better analyze and understand real users' behaviors, a high-resolution QAC log is introduced and analyzed in [19], which records users' interactions with a QAC engine at each keystroke and associated system respond in an entire QAC process. For each submitted query, there is only one record in a traditional search query log. However, in the high-resolution QAC log, each submitted query is associated with a **QAC session**, which is defined to begin with the first keystroke a user typed in the search box towards the final submitted query. The recorded information in each QAC session includes each keystroke a user has

[1]http://www.sogou.com/labs/dl/q.html.

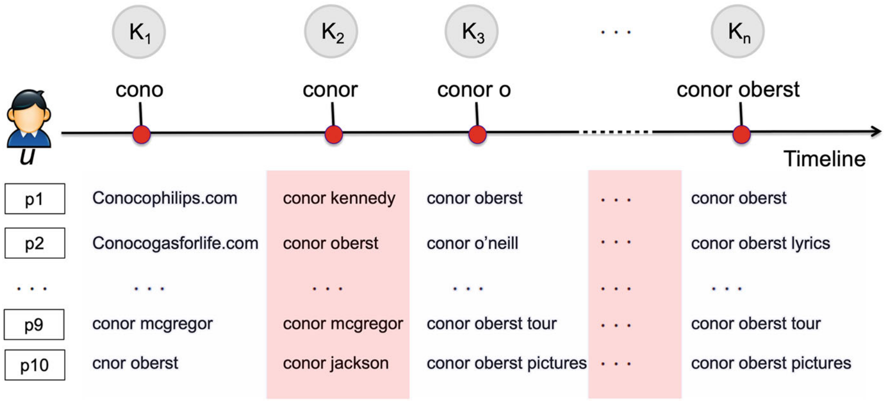


Fig. 7.2 High-resolution QAC log

entered, the timestamp of a keystroke, the corresponding top 10 suggested queries to a prefix, the anonymous user ID, and the final clicked query.

Formally, a **QAC session** contains S keystrokes and each keystroke has a suggested query list of length D as shown in Fig. 7.2.[2] A QAC session ends at the keystroke where the user clicks a query in the suggested query list, or when the prefix at that keystroke is exactly the query the user enters into the search engine. Among the $S \times D$ *slot*s in each QAC session, where each slot q_{ij} is indexed by the i-th position at the j-th keystroke, a user clicks at most one of them, although the user's intended query may appear in many slots.

7.4 QAC Methods

The basic idea to solve the QAC problem is taking the general interest and all users in a search engine and recommend users the most popular queries in search history. A normal query auto-completion engine usually makes an assumption that what a user searched in history is most likely to imply his/her current search intent and maintain a list of all candidate queries with their frequencies. However, such prediction only works under very limited scenario and fails to consider the variations across different users, time slots, etc. In the following, we discuss how different types of information are utilized by existing QAC works. Those information can be generally categorized into: temporal information, contextual information, personal information, user's interaction in QAC, and user's interaction besides QAC.

[2]In real-world search engines, $D = 4$ 4 for Baidu and Google, $D = 8$ for Bing, $D = 10$ for Yahoo.

7.4.1 Time-Sensitive QAC

Temporal information plays an important role in QAC, since search engine user's interest changes from time to time. Significant temporal factors that can result in user's search intent change include:

- User's own interest change along the daily time. Both "star wars" and "star trek" are famous movie/drama series started from many years ago. A user can be very devoted to "star trek" last year and divert his/her attention to "star wars" this year. Since both queries are high-frequency queries existed in the query log for many years, it is hard to decide which query should be ranked in a higher position under the prefix "star." QAC methods need to learn such knowledge from user's most recent query log and provide the appropriate recommendation.
- Periodic events that users participate in regularly. Search engine users can have some periodic interest in certain types of queries (like travel, shopping) that are closely related to seasonal events, such as weekend, yearend, holiday, anniversary, etc. Those queries are usually very different from user's submitted queries in regular days and unable to be predicted from user's recent history. Under this scenario, QAC methods need to utilize user's history at the same/similar seasonal events occurred previously to make the prediction.
- Breaking news that catch up users' attention. User's search intent may also follow the breaking news that happen from time to time. Queries related to those breaking news are likely never recorded in the query log before. QAC methods need to detect the trending queries in the most recent time period and promote those queries in query suggestion lists.

Most popular completion (MPC) is proposed by [1] to rank candidate queries based on their frequencies in the historical query log. This method is a quite straightforward utilization of some basic temporal features and can be regarded as an approximate maximum likelihood estimator as:

$$\text{MPC} = \text{argmax}_{q \in C(p)} \omega(q), \quad \text{where} \quad \omega(q) = \frac{f(q)}{\sum_{q_i \in Q} f(q_i)} \tag{7.5}$$

where $C(p)$ denotes the set of query completions that start with the prefix p, and $f(q)$ denotes the frequency of query q in the query log Q.

The main drawback of MPC is that it assumed user's interest is stable within the range of the collected historical query logs. However, as pointed out in previous paragraphs, user's interest changes from time to time and can be influenced by various types of temporal signals. Thus it makes us difficult to find a certain time window which can be used to predict user's current search intent.

Based on MPC, Shokouhi and Radinsky [27] proposed a time-sensitive QAC ranking model (TS), which replaced the real frequency of candidate queries utilized in MPC with forecasted scores computed by time-series modeling of historical query logs. The score of each candidate at time t is calculated based on its predicted frequency through time-series models that designed to detect the trending queries. This time-sensitive QAC ranking model is formalized as:

$$\text{TS}(p, t) = \text{argmax}_{q \in C(p)} \omega(q|t), \quad \text{where} \quad \omega(q|t) = \frac{\hat{f}_t(q)}{\sum_{q_i \in Q} \hat{f}_t(q_i)} \tag{7.6}$$

where p is the input prefix, $C(p)$ denotes the set of query completions that start with the prefix p, and $\hat{f}_t(q)$ denotes the estimated frequency of query q at time t in the query log Q.

In practice, TS utilized the single exponential smoothing method [11] to predict the frequency of query q at time t based on the real frequency at the last time slot $t - 1$, and a smoothed frequency at the time slot $t - 2$.

$$\hat{f}_t = \bar{f}_{t-1} = \lambda * f_t + (1 - \lambda) * \bar{y}_{t-2} \tag{7.7}$$

where f_{t-1} and $\bar{f}_{t-1}$ denote the real observed and smoothed values for the query frequency at time slot $t - 1$, $\hat{f}_t$ is the estimated frequency of the query at the current time slot t, and λ is a trade-off parameter in the range of [0, 1]. Notice that the smoothed value $\bar{f}_{t-1}$ at the last time slot $t - 1$ is used as the predicted value $\hat{f}_t$ at the current time slot t.

Although this single exponential smoothing can produce reasonable forecasts for stationary time-series, it is proved to perform poorly in capturing the trending queries. Double exponential smoothing methods [11] are proposed to address this issue by extending the previous model with a trend variable involved.

$$\hat{f}_t = \bar{f}_{t-1} + F_{t-1} \tag{7.8}$$

$$\bar{f}_{t-1} = \lambda_1 * f_{t-1} + (1 - \lambda_1) * (\bar{f}_{t-2} + F_{t-2}) \tag{7.9}$$

$$F_{t-1} = \lambda_2 * (\bar{f}_{t-1} - \bar{f}_{t-2}) + (1 - \lambda_2) * F_{t-2} \tag{7.10}$$

Here, parameter F_{t-1} models the linear trend of time-series at time $t - 1$, f_t and $\bar{f}_t$ represent the real and smoothed frequency at time t. λ_1 and λ_2 are smoothing parameters.

In addition to the double exponential smoothing, triple exponential smoothing (or HoltWinters smoothing) [11] goes one step further to model the periodical queries as:

$$\hat{f}_t = (\bar{f}_{t-1} + F_{t-1}) * S_{t-T} \tag{7.11}$$

$$\bar{y}_{t-1} = \lambda_1 * (f_{t-1} - S_{t-1-T}) + (1 - \lambda_1) * (\bar{f}_{t-2} + F_{t-2}) \tag{7.12}$$

$$F_{t-1} = \lambda_2 * (\bar{f}_{t-1} - \bar{f}_{t-2}) + (1 - \lambda_2) * F_{t-2} \tag{7.13}$$

$$S_{t-1} = \lambda_3 * (f_{t-1} - \bar{f}_{t-1}) + (1 - \lambda_3) * S_{t-1-T} \tag{7.14}$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{7.15}$$

where λ_1, λ_2, and λ_3 are free smoothing parameters in [0, 1], S_{t-1} captures the periodicity of query at time $t-1$, and T denotes the length of periodic cycle.

Another solution based on time-series analysis is a time-sensitive QAC method proposed by Cai et al. [7], which attempted to detect both cyclically and instantly frequent queries. This method estimated the current query frequency as a linear combination of its periodicity score and trending score. It not only inherited the merits of time-series analysis for long-term observations of query popularity, but also considered recent variations in query frequency. In specific, it predicted the frequency of a query q at time slot t through:

$$\hat{f}_t(q,\lambda) = \lambda * \hat{f}_t(q)_{trend} + (1-\lambda) * \hat{f}_t(q)_{peri} \tag{7.16}$$

where $\hat{f}_t(q)_{trend}$ tries to capture the trending of query q, and $\hat{f}_t(q)_{peri}$ tries to capture the periodicity of query q. This method sets $\lambda = 1$ for aperiodic queries and $0 \leq \lambda < 1$ for periodic queries.

The term $\hat{f}_t(q)_{trend}$ is formulated as a linear combination of the trending queries during the most recent N days:

$$\hat{f}_t(q)_{trend} = \sum_{i=1}^{N} \omega_i * \hat{f}_t(q,i)_{trend} \tag{7.17}$$

Here ω_i is a time decay weight while constrained by the condition that $\sum_i \omega_i = 1$.

The trending prediction for each day i is calculated based on the first order derivative of the frequency of query q within time slot t:

$$\hat{f}_t(q,i)_{trend} = f_{t-1-\mathrm{TD}(i)}(q) + \int_{t-1-\mathrm{TD}(i)}^{t} \frac{\partial C(q,t)}{\partial t} dt \tag{7.18}$$

Here $f_{t-\mathrm{TD}(i)}(q)$ is actual the frequency of query q at day i, while $C(q,t)$ denotes the frequency of query q within time slot t.

The term $\hat{f}_t(q)_{peri}$ is formulated as the smoothing term that averages the query frequency of the most recent M preceding time slots $t_p = t - 1 * T_q, \ldots, t - M * T_q$ in the query log as:

$$\hat{f}_t(q)_{peri} = \frac{1}{M} \sum_{m=1}^{M} f_{t-m*T_q}(q) \tag{7.19}$$

Here T_q is the length of periodic cycle of query q.

The temporal information recorded in the QAC log is not limited to be utilized in the query frequency estimation. Recent works [17] also made use of this information to reveal the relationship between user's click behaviors in QAC logs, such as the click position. Different search engine users can have different preferences in the positions to click during the QAC process. For instance, some users prefer to make clicks at lower positions under shorter prefixes, while others prefer higher positions under longer prefixes. For the same user, such kind of preference may also change with respect to time. Thus in learning a user's preference of click position, it is reasonable to assign higher weights to the recent historical click positions. To quantify the degree of the influence between click events from the temporal aspect, this method employed the following formula:

$$\kappa(t_l - t) \tag{7.20}$$

where t is the timestamp when a user makes the current click, t_l is the timestamp when the l-th historical click event occurs, and $\kappa(t_l - t)$ represents a time decay effect.

7.4.2 Context-Sensitive QAC

Context-sensitive QAC methods take the context which a search engine user has input into consideration in user search intent prediction. Different from the normal query frequency based QAC methods, which predict the probability that whether a candidate queries will be issued by a user based on the exact same query recorded in historical query logs, context-aware QAC methods make the prediction based on the submission of other queries that share a certain relationship with the predicted candidate query. Such relationship can be:

- With similar content. Queries that shared similar content are very likely to reflect the same or similar user search intent. Thus besides the original query, other queries with similar content can provide additional evidence in the search intent prediction. For instance, "star wars" and "star wars the old republic" can both tell a user's interest in the movie/drama "star wars," and the frequency of both queries can be very high. Such kind of information is especially useful in predicting user's search intent under short keystrokes, such as "st" in this case, since the frequency of the single query "star wars" is not able to represent user's real relative degree of interest in this movie/drama, when compared with other queries which also started with "st" but do not have so many high-frequency queries with similar content.
- Belong to the same category. Such information can be helpful in revealing a user's interest when little information is given, for instance, only one keystroke is entered. One typical example is that, if most queries submitted by a user are shopping queries, it is very likely that he/she will click "amazon" rather than "aol" under the keystroke 'a'.
- Co-occurred frequently in the query log. If two consecutive queries "hollywood" and "beverly hills" are issued by the same user, then the previous query "hollywood" can also be viewed as the context of query "beverly hills." Generally, two consecutive queries issued many times by different users are more likely

to have a strong correlation between each other. It makes more sense to take into account the explicit temporal information of query sequences exhibited by many different users in the whole-query logs. The basic intuition is that if two consecutive or temporally close queries are issued many times by the same user or many other users, it is more likely that these two queries are semantically related to each other. Those queries are very likely to form a search task, which target to accomplish a single search intent goal (travel in western los angeles in the previous example). Thus the co-occurrence of those queries can happen frequently in the future across different users that conduct the same search task.

The NearestCompletion method [1] utilized users' recent queries as the context of the user input. This method did a good job in predicting user's search intent when matching the context of the user.

NearestCompletion described a context-sensitive extension of the Maximum Likelihood Estimator, which tried to predict the candidate query q that started with prefix x whose presentation vector v_q has the highest cosine similarity to the search context representation v_C:

$$\text{NearestCompletion}(p, C) = \text{argmax}_{q \in C(p)} \frac{< v_q, v_C >}{\|v_q\| \cdot \|v_C\|}. \tag{7.21}$$

Here $C(p)$ is the set of candidate queries starting with prefix p.

The context representations in NearestCompletion are based on the query representations. Given $v_{q_1}, \ldots, v_{q_t}$ as the corresponding vectors of context $C = q_1, \ldots, q_t$. The context vector v_C is formulated as a linear combination of the query vectors $v_C = \sum_{i=1}^{t} \omega_i v_{q_i}$, with weights $\omega_1, \ldots, \omega_t \geq 0$. Those weights described the degree of the influence from the historical query as context to the current search intent of a user. They are required to be time decayed, since the more recent submitted queries are more likely to be relevant to the current query. Popular weight functions that satisfy this condition include: recent-query-only (w_t = 1 and w_i = 0 for all $i < t$), linear decay ($w_i = 1/(t - i + 1)$), logarithmic decay ($w_i = 1/(1 + \log(t - i + 1))$), and exponential decay ($w_i = 1/e_{t-i}$).

Notice that using the output of NearestCompletion alone for a QAC task is not working well for when a new user joins or a user's current search intent is not relevant to the context collected for the user. In practice, this work used a linear combination of the score from the NearestCompletion function and the MPC function introduced above as the final score for the query candidates ranking in QAC.

Cai et al. [7] utilized two different types of context for search intent prediction. One is the set of queries in the current search session, denoted as Q_s, the other is the set of historical queries issued by user u, denoted as Q_u. This method calculated the scores of the candidates $q_c \in S(p)$ through a linear combination of similarity scores $\text{Score}(Q_s, q_c)$ and $\text{Score}(Q_u, q_c)$ as follows:

$$\text{Pscore}(q_c) = \omega * \text{Score}(Q_s, q_c) + (1 - \omega) * \text{Score}(Q_u, q_c) \tag{7.22}$$

here ω weights the above two components.

To compute the similarity scores, this method used n-gram to represent each query, thus enabling the proposed similarity score to capture syntactic reformulations. Moreover, to overcome the problem that the query vocabulary is too sparse to capture semantic relationships, it treated a user's preceding queries Q_s in the current session and Q_u in the historical log as context to personalize QAC where the similarity is measured at the character level.

Jiang et al. [13] studied user's reformulation behaviors in QAC based on the context information. Three types of context based features are designed to describe the reformulation behaviors of search engine users by capturing how users modify their preceding queries in a query session, including:

- Term-level features: for instance, term keeping—$|S(q_{t-1}) \cap S(q_t)|$, which describes the number of shared terms by the query issued at time slot t and the previous query at time slot $t-1$.
- Query-level features: for instance, average cosine similarity—$\frac{1}{t-1}\sum_{i=1}^{t-1} \text{sim}_{\cos}(q_i, q_t)$, which calculates the content similarity between the queries issued at time slot t and all previous historical queries issued within the same query session.
- Session-level features: for instance, ratio of effective terms $|C_{\text{eff}}(q_t)|/|S(q_t)|$, which is the ratio of the number of clicks on the search results of query q_t divided by the number of terms in query q_t.

Such contextual features that capture user's reformulation behaviors are proved to be an effective additional signal to the regular context features introduced above.

Li et al. [17] designed a set of contextual features that describe the relationship between the content of a historical query q' and the current suggested query q, to quantify the degree of the influence between click events from the context aspect. These features count the number of appearances of a certain pattern involving both the historical query q' and the current suggestion q in a certain time range formulated as:

$$x(p)(t, \Delta t) = \#\{p \in [t - \Delta t, t)\} \tag{7.23}$$

where p represents a certain defined pattern, $[t - \Delta t, t)$ is the time interval from some ancient timestamp to the current timestamp. Table 7.1 shows several patterns adopted in this work, which is inspired by the features proposed in [25].

As shown in Table 7.1, those contextual features generally originate from the co-occurrence of two queries in the query sequence submitted by search engine users and reflect pairwise relationship. A feature vector $\mathbf{x}_{q',q}(t)$ is formed for each query-pair (q', q) at any given timestamp t as

$$\mathbf{x}_{q',q}(t) = \{x(p)(t, \Delta t) | p \in \mathcal{P}_{q',q}, \Delta t > 0\} \tag{7.24}$$

where $\mathcal{P}_{q',q}$ refers to the set of patterns involving the pair of queries $\{q', q\}$. Thus for each timestamp t, a unique set of feature vectors $\{\mathbf{x}_{q',q}(t)\}$ imply how a historical

Table 7.1 Patterns in constructing contextual features

Pattern p	Description
$q' \rightarrow q$	Query q is submitted just after the submission of query q'
$q \longleftrightarrow q'$	Query q and q' are submitted in adjacent
$q' \xrightarrow{(v)} q$	Query q is submitted after the submission of query q', and v queries have been submitted in between
$q \overset{(v)}{\longleftrightarrow} q'$	v queries are submitted between the submission of query q and q'

click event (on query q') influences the current click event (on query q) from the contextual aspect.

7.4.3 Personalized QAC

Another type of useful signal in QAC is the user's personal information. Unlike the context information that can vary with respect to time, personal information described user's inherit characteristics that are mainly to differentiate one user from other users, or a group of users from other groups. Such personal information generally includes:

- bcookie. In query logs, bcookie is unusually used as the identifier of a single search engine user, although in real-world scenario one actual user can have multiple bcookies (such as owning multiple computers), and a bcookie can be shared by multiple users (family members shared computers or public computers). Learning distinct models based on the query log under different bcookies can increase the accuracy in user's search intent prediction, since the interest and search habit of users can be very different from each other. The major drawback of bcookie based model is that the number of QAC sessions completed by a single user is very limited. Obviously, it will fail when facing new bcookies. Moreover, for most normal search engine users who regularly submit tens of queries per day, the available QAC sessions for model learning are very limited. Thus, more general categorical personal information is also important, which can jointly utilize the query logs of users within the same category to benefit the model learning.
- gender. The gender of a search engine user can be a strong signal in predicting his/her interest. A male user is more likely to submit sports queries than shopping queries, and vice versa for a female user.
- age. The age of a search engine user is another strong signal. Teenagers usually prefer gaming, while older individuals care more about health.
- location. The location information is very useful in query suggestion, since a large percentage of queries submitted by search engine users are with local intent. For instance, under the prefix "amc," a user lived in sunnyvale is more likely to search for "amc cupertino" instead of "amc san Francisco." Notice that location

is not a typical type of personal information, since a user can move across cites, states, or even countries, especially for users who spend a great amount of time in traveling. However, for regular search engine users, their locations are usually stable.

In the Personalized QAC model proposed in [26], a number of demographic features are utilized, including users' age, gender, and zip-code information from their Microsoft Live profiles. This method divides users into five groups based on age: $\{<20, 21-30, 31-40, 41-50, >50\}$. For each user, the model made use of the frequency of query candidates that submitted by all other users fall into the same age groups as a feature. Similar features are also generated based on gender and zip-code information. Notice that the zip-codes are also collapsed into 10 regions according to the corresponding first digits, so as to reduce sparsity. Those demographic features are incorporated into a supervised learning framework for personalized ranking of query auto-completion.

Cai et al. [6] also conducted experiments to test the effectiveness of the demographic features in learning to rank algorithms such as Burges et al. [5], results showed that demographic features such as location are more effective than others in the QAC task. The SQA algorithm proposed in [20] studied how to utilize location information to solve the QAC task based on a native index structure combined with a spatial index. This method utilized the longitude/latitude information to describe a certain location and ranked candidate suggestions q given a certain prefix p based on the ranking score function as:

$$\text{RankScore}(q, p) = \alpha * \frac{\text{Dis}(q_{loc}, p_{loc})}{\text{DisMax}} + (1-\alpha) * \text{RelScore}(q, p) \tag{7.25}$$

where $\alpha \in (0, 1)$ is a parameter that balances the spatial proximity and the normal relevancy between the candidate suggestion q and prefix p. $\text{Dis}(q_{loc}, p_{loc})$ is the Euclidean distance between q_{loc}, the location descriptor of query q, and p_{loc}, the location when user typing the prefix p. DisMax is the potential max distance value used for normalization. $\text{RelScore}(q, p)$ is the normal relevance score for the query-prefix pair (q, p) calculated based on regular QAC features.

7.4.4 User Interactions in QAC

Rich user interactions can be observed along with each keystroke until a user clicks a suggestion or types the entire query manually. It becomes increasingly important to analyze and understand users' interactions with the QAC engine, so as to improve its performance. Figure 7.3 presents the general process that a search engine user interacts with the QAC engine in a QAC session.

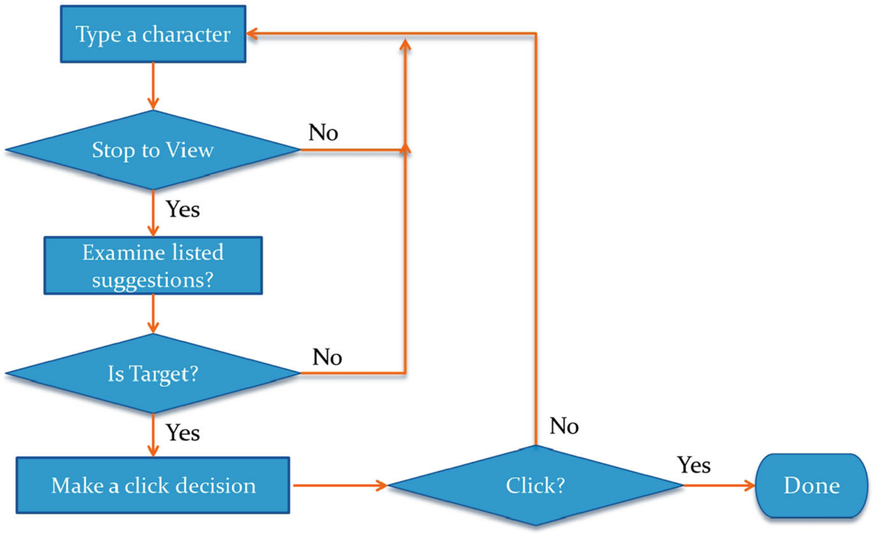


Fig. 7.3 How a search engine user interacts with the QAC engine in a QAC session

In the following, we list several typical user behaviors that can facilitate the understanding of the QAC process:

- **Click behavior.** User's click behavior is one of the key signals in understanding user's search intent, since the target of a QAC method is to increase user's click chance during a QAC session (as early as possible). There are mainly two types of information in user's click behaviors:
  - **Position bias.** One important type of click information is the click bias on vertical positions in QAC. Using the same set of QAC sessions, we have computed the distribution of clicks according to their positions in the final suggestion list and the final prefix length. Similar to the findings in the traditional click models, most of the clicks concentrate on top positions. Such vertical positional bias suggests that the relevance estimation of queries should be boosted if they are clicked on lower ranks. Compared to user's QAC behavior on PC, their clicks on mobile distribute more evenly within positions from 1 to 3. In addition, most of the clicks are located in long prefix, the click probability at short prefix length (1 and 2) is very low, suggesting that users tend to skip the suggested queries at the beginning.
  - **Click choice.** The click choice of a user can provide rich information in predicting user's search intent. The query candidates that have been suggested by the QAC engine but not clicked by a user during a QAC session have a small chance to meet user's search intent. A user is unlikely to look for the query "facebook" if he/she does not select it under prefix "f," since "facebook" is the top query suggestion.

- **Skipping and viewing behavior.** Search engine users frequently skip several intermediate lists of candidates even though these lists contain their final submitted queries. A plausible explanation for the skipping behavior is that the user did not examine it due to some reasons, such as fast typing speed, too deep to look up the query, etc. The inference of whether a user has skipped to view a certain keystroke or a certain suggested query under that keystroke can be very helpful in predicting user's real search intent, since a query that a user does not click may also satisfy his/her intent if he/she skips that query or the corresponding keystroke due to the reasons mentioned as above.
- **Typing behavior.** Typing speed is an important signal that characterizes a search engine user. A user with fast typing speed is probably an expert user who has rich experience in using search engines, usually has a clear search intent, and is aware of what exact query to enter before starting a QAC session. Thus an expert user is less likely to use the assist from the QAC engine than a new search engine user.

Jiang et al. [13] employed user's click behaviors to model user's reformulation habit. This work designed session-level features based on both the timestamps of user's clicks in QAC. It calculated the average time duration between clicks as $\frac{1}{T-1}\sum_{i=1}^{T-1}(t_{i+1} - t_i)$, and the trends of time duration as $(t_T - t_{T-1})/\frac{1}{T-2}\sum_{i=1}^{T-2}(t_{i+1} - t_i)$, where t_i is the timestamp of the click that occurs in the i-th QAC session.

Li et al. [17] explored to learn the position bias in a user's click preference based on the positional information of historical QAC sessions from the same user. This work quantified the degree of the influence between the click events from the special slot aspect using the following formula:

$$\kappa(|\mathbf{p}_l - \mathbf{p}|) \tag{7.26}$$

where $\mathbf{p}$ is the slot where a user makes the current click, and $\mathbf{p}_l$ is the slot where the user makes the l-th historical click event, i.e., the click occurs at the l-th QAC session, and $\kappa(|\mathbf{p}_l - \mathbf{p}|)$ represents a decay effect from the slot discrepancy. Notice that $\mathbf{p} = (i, j)$ is a vector of length 2, its entries i and j denote the position and the keystroke, respectively.

TDCM [19] tried to utilize user's skipping behaviors and clicking position bias information to understand user's click choice during the QAC process. It defined a basic assumption for each type of user behaviors separately as below:

- SKIPPING BIAS ASSUMPTION: A query will not receive a click if the user did not stop and examine the suggested list of queries, regardless of the relevance of the query. This assumption explains why there are no clicks to intermediate prefix even though a relevant query is ranked at the top of the list, and all of the clicks are concentrated on the final prefix.
- VERTICAL POSITION BIAS ASSUMPTION: A query on higher rank tends to attract more clicks regardless of its relevance to the prefix.

Based on the above assumptions, TDCM proposed a Two-Dimensional Click Model to explain the observed clicks. This click model consists of a horizontal model (H Model) that explains the skipping behavior, a vertical model (D Model) that depicts the vertical examination behavior, and a relevance model (R Model) that measures the intrinsic relevance between the prefix and a suggested query.

In specific, TDCM formulated the probability of observing a click C in a session as:

$$P(C) = \sum_{H,D} P(C, H, D) \tag{7.27}$$

where $H = \{H_1, \ldots, H_n\}$, $D = \{D_1, \ldots, D_n\}$ is a set of hidden variables, respectively. Here, H_i denotes whether the user stops to examine the column i, and D_i denotes the depth of examination at column i. $C = \{C_1, \ldots, C_n\}$ is the click observation matrix in which only one click is observed: $C_{n,J} = 1$, n is the number of columns in the QAC session. This model followed the Cascade Model assumption as:

$$P(C_{n,J} = 1) = P(C_1 = 0, \ldots, C_{n1} = 0, C_{n,J} = 1, C_{n,j} = 0, j \neq J) \tag{7.28}$$

as well as the set of conditional probabilities as:

$$P(C_{ij} = 1 | H_i = 0) = 0 \tag{7.29}$$

$$P(C_{ij} = 1 | H_i = 1, D_i < j) = 0 \tag{7.30}$$

$$P(C_{ij} = 0 | H_i, D_i) = 1 - P(C_{ij} = 1 | H_i, D_i) \tag{7.31}$$

$$P(D_i > d | q_d : C_{n,d} = 1) = 0 \tag{7.32}$$

Among the above conditional probabilities, Eqs. (7.30) and (7.32) modeled the SKIPPING BIAS ASSUMPTION, and Eqs. (7.31) and (7.32) modeled the VERTICAL POSITION BIAS ASSUMPTION. Equation (7.32) stated that if a relevant query is ranked in depth d, the examination depth at the i-th column must not exceed d.

In the H model, TDCM attempted to capture user's skipping behavior via the following features: TypingSpeed: an expert user is less likely to use QAC than a slow user. CurrPosition: a user tends to examine the queries at the end of typing. IsWordBoundary: a user is more likely to look up queries at word boundaries. NbSuggQueries: it is more likely to be examined if the list of queries is short. ContentSim: a user may be more likely to examine the list if all queries are coherent in content. QueryIntent: a user tends to skip the list more when searching for navigational queries. Also, in the D model, TDCM utilized the positions a query candidate is ranked to measure the pure vertical position bias.

Zhang et al. [32] studied how user's click behavior can be utilized as the implicit negative feedback during user-QAC interactions. The key challenge is that this kind of implicit negative feedback can be strong or weak, and its strength cannot be directly observed. It utilized additional information such as dwell time and position to capture the confidence in using an unclicked suggestion as implicit negative feedback in search intent prediction.

If a user dwells on a suggestion list for a longer time, the user may have more time to carefully examine the suggested queries. Conversely, if a user dwells for a shorter time, the suggested queries will more likely be ignored; thus, even if these queries are unselected, whether the user favors them or not is unknown. Since different users may have different typing speeds, the inference of implicit negative feedback strength by dwell time should be personalized. This method represented implicit negative feedback from the user u to the query q at the k-th keystroke during the c-th QAC session in the QAC log by a feature vector $\mathbf{x}^{(k)}(u, q, c)$. The features utilized include: DwellT-M, the maximum dwell time when query q is suggested; DwellT, total dwell time where query q is suggested; WordBound, the number of the keystrokes at word boundaries when query q is suggested; SpaceChar, the number of the keystrokes at space characters when query q is suggested; OtherChar, the number of the keystrokes at non-alphanum char when query q is suggested; IsPrevQuery, 1 if query q is the immediately previous query; 0 otherwise; and Pos@i, the number of the keystrokes when query q is at position $i\,(i = 1, 2, \ldots, 10)$ of a suggestion list.

Then a generalized additive model, named AdaQAC, is proposed to predict the preference $p^{(k)}(u, q, c)$ for a query q of a user u at a keystroke k in the c-th QAC session:

$$p^{(k)}(u, q, c) = r^{(k)}(u, q, c) + \phi^{\top}(u)\mathbf{x}^{(k)}(u, q, c) \tag{7.33}$$

Here, the preference model $p^{(k)}(u, q, c)$ is able to reflect a user u's preference for a query q after the implicit negative feedback $\mathbf{x}^{(k)}(u, q, c)$ is expressed to q before the k-th keystroke in the c-th QAC session. With the associated feature weights $\phi(u)$ personalized for u, $\phi^{\top}(u)\mathbf{x}^{(k)}(u, q, c)$ encodes the strength of implicit negative feedback to q from u with personalization.

In addition to the above introduced QAC methods which modeled user's interaction at each keystroke independently, RBCM [16] made a further step to study the relationship between users' behaviors at different keystrokes, which includes:

1. **State transitions between skipping and viewing.** The study on high-resolution query log data revealed that a user may choose to either view or skip the suggestion list at each keystroke in a QAC session. Besides the above introduced factors that influence users' decisions on skipping or viewing, such as typing speed and whether the end of current prefix is at word boundary. This work believed that such decisions should also be influenced by their decisions on skipping or viewing at the previous keystroke. For instance, imagining a user u has 5 sequential skipping moves in one QAC session and 2 sequential skipping moves in another QAC session, the chance becomes higher for the same user to stop and view the suggestion list at the current keystroke after 5 sequential

skipping moves. Conversely, if the same user has already viewed too many keystrokes continuously but found no intended query, it becomes more likely that he/she may skip the next one;

2. **Users' real preference of suggestions**. This work claimed that, for each keystroke, the associated users' real preference is hard to be detected from the current suggested query list alone. The rankings of suggested query lists of latter keystrokes together with users' final click choices should also be utilized to re-rank the suggested queries in the list of the current keystroke. Intuitively, a clicked query, i.e., the user's intended query, should get a higher rank not only at the keystroke he/she makes the click, but also at previous keystrokes where this query appears, despite that it is not clicked at that time;
3. **User-specific cost between position clicking and typing**. Some users prefer typing than viewing and clicking, while others do not. Consequently, users' click choices are not only affected by their intent, but also by the position where the intended query is shown and their preference of clicking that position over typing the remaining keystrokes. For instance, a user that prefers clicking will probably click an intended query the first time it is shown to him/her, despite that it may be shown in a low position; while another user focuses on typing his/her intended query despite that the query already appears in the suggestion list, until it is ranked at the top position, or even worse, he/she will type the entire query manually without any intent to click the suggestions.

7.4.5 User Interactions Besides QAC

Besides the information recorded in the QAC log, user's behavior on other types of search logs can also be very useful in predicting user's search intent. One typical example is user's click log, which recorded user's click behavior on the returned web results after submitting a query.

Figure 7.4 shows a toy example of QAC and click logs that align in the timeline. We can observe that the QAC session of a query is followed by the click session of that query, and that click session is followed by another QAC session of the next query. Such sequential behaviors indicate the promising opportunity of exploring appropriate relationship between QAC and click logs. Although the user's behaviors on QAC and click logs are of different types, they imply the same underlying relationship between the user and his/her issued query, such as whether the issued query satisfies the user's intent, and how familiar the user is with the issued query or the domain that query belongs to. For instance, if a user is familiar with the issued query in QAC log, he/she may type the query very fast. Then in click log, if the SERP page provides many relevant results, the user may take a long time to click and check some relevant results in more details; however, if the SERP page does not provide relevant results, the user may reformulate a new query shortly which will start a new QAC session similar to previous query.

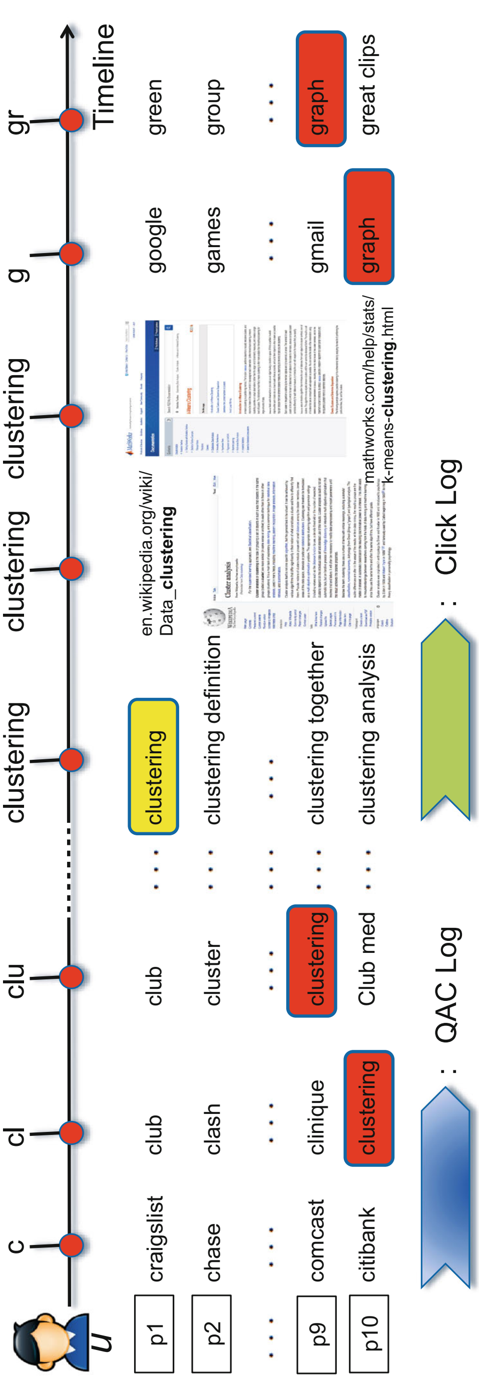


Fig. 7.4 A toy example of how QAC and click logs align in the timeline. Yellow tag highlights the query a user finally clicks, red tag highlights the user's intended query he/she does not click

Moreover, user's search behaviors on one type of log can be used as the contextual data for the other type of log across different query sessions, since users generally behave consistently in adjacent time slots. For instance, according to the click log, if a user's behaviors indicate he is very familiar with the current query, then similar behaviors will be likely observed in the QAC session of the next query; if the issued query is under the same topic, the user will probably type the following query fast as well.

Li et al. [18] studied and designed various QAC and click features in quantitatively capturing user behaviors on QAC and click logs, Among features of QAC behaviors, "Type Speed Standard Deviation" is designed to reflect the stability of a user's typing speed. A user who examines his/her intended queries from the suggestion list from time to time may hardly maintain a stable typing speed, even if the user has good typing skills. On the contrary, a user who plans to type the entire query without clicking a suggestion may illustrate a stable typing speed. "Typing Completion" is designed to show whether a user prefers typing than clicking suggestions. Among the features of click behaviors, "Search Time" is defined to be how fast a user can find his/her intended web documents after submitting a query. Notice that users' behaviors on different types of logs are not independent. On the QAC log, an experienced user usually spends less time to complete a QAC session than an unexperienced user, i.e., has a small "Time Duration." While on the click log, he/she is very likely to make his/her first click after only a short while, i.e., a small feature value for "Search Time." A user who tends to trust the results of search engines may miss the QAC behavior feature "Typing Completion" and own a higher value of the click behavior feature "Click Number." Thus the above designed QAC and click behavior features are somehow related.

To detect user behavior patterns from logs, this work proposed a graphical model based on latent Dirichlet allocation (LDA) [4], which has been proven to be effective in topic discovery by clustering words that co-occur in the same document into topics. It treats each user's query sequence as a document, and clustered user behaviors that co-occur frequently in the same query sequence into topics, since each user maintains certain behavior patterns in query submission, and different groups of users prefer different behavior patterns. The model assumed K behavior patterns lie in the given query sequences, and each user m is associated with a randomly drawn vector π_m, where $\pi_{m,k}$ denotes the probability that the user behavior in a query session of user m belongs to behavior pattern k. For the n-th query in the query sequence of user m, a K-dimensional binary vector $Y_{m,n} = [y_{m,n,1}, \ldots, y_{m,n,K}]^T$ is used to denote the pattern membership of the user behavior in that query session.

To model the influence of the context on user's choice of the behavior pattern in the current query session, the proposed model assumed user's preference of behavior patterns depends on the context, rather than the user alone. That is to say, a "document" in the LDA model does not contain the user behaviors in all query sessions of a user, but only the behaviors in those query sessions that the user conducts under the same status, for instance, in the same mood, or sharing the same topic.

7.5 Historical Notes

Query auto-completion (QAC) has been attracting people's attention for quite a long time. The main objective of QAC is to predict users' intended queries and assist them to formulate a query while typing. The most popular QAC algorithm is to suggest completions according to their past popularity. Generally, a popularity score is assigned to each query based on the frequency of the query in the query log from where the query database was built. This simple QAC algorithm is called Most Popular Completion (MPC), which can be regarded as an approximate maximum likelihood estimator [1].

Several QAC methods [1, 26, 27, 31] were proposed to extend MPC from various aspects. Bar-Yossef and Kraus [1] introduced the context-sensitive QAC method by treating users' recent queries as context and taking into account the similarity of QAC candidates with this context for ranking. But there is no consensus of how to optimally train the relevance model. Shokouhi [26] employed a learning-based strategy to incorporate several global and personal features into the QAC model. However, these methods only exploit the final submitted query or simulate the prefixes of the clicked query, which do not investigate the users' interactions with the QAC engine.

In addition to the above models, there are several studies addressing different aspects of QAC. For example, [27, 31] focused on the time-sensitive aspect of QAC. Other methods studied the space efficiency of index for QAC [2, 12]. Duan and Hsu [9] addressed the problem of suggesting query completions when the prefix is misspelled. Kharitonov et al. [15] proposed two new metrics for offline QAC evaluation and [14] investigated user reformation behavior for QAC.

The QAC is a complex process where a user goes through a series of interactions with the QAC engine before clicking on a suggestion. Smith et al. [28] presented an exploratory study of QAC usage during complete search sessions based on the lab study of tens of search engine users, the result implicated the effectiveness of the knowledge from prior queries within the same search session in improving the suggestions over successive queries in query auto-completion. As can be seen from the related work, little attention has been paid to understand the interactions with the QAC engine. Until recently, Li et al. [19] created a two-dimensional click model to combine users' behaviors with the existing learning-based QAC models. This study assumed users' behaviors at different keystrokes, even for the consecutive two keystrokes, are independent in order to simplify the model estimation, which results in information loss. In advance, Li et al. [16] attempted to directly model and leverage the relationship between users' behaviors, so as to improve the performance of QAC. Furthermore, users' behaviors besides the QAC process, such as the behaviors in click logs, have also be explored in benefiting the QAC task in [18].

In recent years, more and more special scenarios under the QAC problem have been explored. Wang et al. [30] formulated the QAC task as a ranked Multi-Armed Bandits (MAB) problem to timely and adaptively suggest queries and expected to reflect time-sensitive changes in an online fashion. Vargas et al. [29] claimed that the traditional whole-query completion mechanism is not the optimal solution for mobile search scenarios. Inspired by predictive keyboards that suggests to the user one term at a time, they proposed the idea of term-by-term QAC. Liu et al. [22] investigated into the promotion campaign issue in QAC engines, where some malicious users provided a new malicious advertising service by attacking the search engines through using manipulated contents to replace legitimate auto-completion candidate suggestions, so as to promote their customers' products in QAC. Modern techniques have also been utilized in solving the QAC problem. Recurrent neural network (RNN) models have been employed to address the QAC task in [23], in order to improve the quality of suggested queries when facing previously unseen text.

7.6 Summary

Last, we briefly summarize the main content introduced in this chapter and discuss potential future research directions.

In this chapter, we have presented the main contributions in the field of query auto-completion in information retrieval. In specific, in Sect. 7.1, we gave a general introduction of query auto-completion and provided a formal definition of the QAC problem. In Sect. 7.2, we introduced existing metrics utilized in measuring the QAC performance, including both ranking quality and assist efficiency. In Sect. 7.3, different types of QAC logs utilized in existing QAC works are studied. In Sect. 7.4, we have introduced the most prominent QAC approaches in the literature, and how the usage of different types of information can benefit the prediction of user intent. Those information includes temporal, contextual, personal information, and user's interaction inside and outside the QAC process.

References

1. Ziv Bar-Yossef and Naama Kraus. Context-sensitive query auto-completion. In *Proceedings of the 20th International Conference on World Wide Web*, pages 107–116, 2011.
2. H. Bast and Ingmar Weber. Type less, find more: fast autocompletion search with a succinct index. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 364–371, 2006.
3. Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 795–804, 2011.
4. David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *J. Mach. Learn. Res.*, 3: 993–1022, 2003.
5. Christopher J. C. Burges, Krysta M. Svore, Paul N. Bennett, Andrzej Pastusiak, and Qiang Wu. Learning to rank using an ensemble of lambda-gradient models. In *Proceedings of the Yahoo! Learning to Rank Challenge, held at ICML 2010*, volume 14 of *JMLR Proceedings*, pages 25–35, 2011.

6. Fei Cai and Maarten de Rijke. A survey of query auto completion in information retrieval. *Found. Trends Inf. Retr.*, 10 (4): 273–363, 2016.
7. Fei Cai, Shangsong Liang, and Maarten de Rijke. Time-sensitive personalized query auto-completion. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1599–1608, 2014.
8. Nick Craswell, Rosie Jones, Georges Dupret, and Evelyne Viegas. WSCD '09: Proceedings of the 2009 workshop on web search click data. 2009.
9. Huizhong Duan and Bo-June Paul Hsu. Online spelling correction for query completion. In *Proceedings of the 20th International Conference on World Wide Web*, pages 117–126, 2011.
10. Katja Hofmann, Bhaskar Mitra, Filip Radlinski, and Milad Shokouhi. An eye-tracking study of user interactions with query auto completion. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 549–558, 2014.
11. Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. In *International Journal of Forecasting*, volume 20, pages 5–10, 2004.
12. Bo-June Paul Hsu and Giuseppe Ottaviano. Space-efficient data structures for top-k completion. In *Proceedings of the 22nd International World Wide Web Conference*, pages 583–594, 2013.
13. Jyun-Yu Jiang, Yen-Yu Ke, Pao-Yu Chien, and Pu-Jen Cheng. Learning user reformulation behavior for query auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 445–454, 2014a.
14. Jyun-Yu Jiang, Yen-Yu Ke, Pao-Yu Chien, and Pu-Jen Cheng. Learning user reformulation behavior for query auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 445–454, 2014b.
15. Eugene Kharitonov, Craig Macdonald, Pavel Serdyukov, and Iadh Ounis. User model-based metrics for offline query suggestion evaluation. In *Proceedings of the 36th International ACM SIGIR conference on research and development in Information Retrieval*, pages 633–642, 2013.
16. Liangda Li, Hongbo Deng, Anlei Dong, Yi Chang, Hongyuan Zha, and Ricardo Baeza-Yates. 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*, pages 123–132, 2015.
17. Liangda Li, Hongbo Deng, Jianhui Chen, and Yi Chang. 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*, pages 131–139, 2017a.
18. Liangda Li, Hongbo Deng, Anlei Dong, Yi Chang, Ricardo Baeza-Yates, and Hongyuan Zha. 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*, pages 539–548, 2017b.
19. Yanen Li, Anlei Dong, Hongning Wang, Hongbo Deng, Yi Chang, and ChengXiang Zhai. A two-dimensional click model for query auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 455–464, 2014.
20. Chunbin Lin, Jianguo Wang, and Jiaheng Lu. Location-sensitive query auto-completion. In *Proceedings of the 26th International Conference on World Wide Web*, pages 819–820, 2017.
21. Yang Liu, Ruihua Song, Yu Chen, Jian-Yun Nie, and Ji-Rong Wen. Adaptive query suggestion for difficult queries. In *Proceedings of the 35th International ACM SIGIR conference on research and development in Information Retrieval*, pages 15–24, 2012.
22. Yuli Liu, Yiqun Liu, Ke Zhou, Min Zhang, Shaoping Ma, Yue Yin, and Hengliang Luo. Detecting promotion campaigns in query auto completion. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pages 125–134, 2016.
23. Dae Hoon Park and Rikio Chiba. A neural language model for query auto-completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1189–1192, 2017.

24. Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, volume 152, page 1, 2006.
25. Patrick O. Perry and Patrick J. Wolfe. Point process modeling for directed interaction networks. *CoRR*, abs/1011.1703, 2010.
26. Milad Shokouhi. Learning to personalize query auto-completion. In *Proceedings of the 36th International ACM SIGIR conference on research and development in Information Retrieval*, pages 103–112, 2013.
27. Milad Shokouhi and Kira Radinsky. Time-sensitive query auto-completion. In *Proceedings of the 35th International ACM SIGIR conference on research and development in Information Retrieval*, pages 601–610, 2012.
28. Catherine L. Smith, Jacek Gwizdka, and Henry Feild. The use of query auto-completion over the course of search sessions with multifaceted information needs. *Inf. Process. Manag.*, 53 (5): 1139–1155, 2017.
29. Saúl Vargas, Roi Blanco, and Peter Mika. Term-by-term query auto-completion for mobile search. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 143–152, 2016.
30. Suhang Wang, Yilin Wang, Jiliang Tang, Charu C. Aggarwal, Suhas Ranganath, and Huan Liu. Exploiting hierarchical structures for unsupervised feature selection. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 507–515, 2017.
31. Stewart Whiting and Joemon M. Jose. Recent and robust query auto-completion. In *Proceedings of the 23rd International World Wide Web Conference*, pages 971–982, 2014.
32. Aston Zhang, Amit Goyal, Weize Kong, Hongbo Deng, Anlei Dong, Yi Chang, Carl A. Gunter, and Jiawei Han. 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*, pages 143–152, 2015.