adaQAC: Adaptive Query Auto-Completion via Implicit Negative Feedback

Aston Zhang*[1], Amit Goyal[2], Weize Kong[3], Hongbo Deng[2]
Anlei Dong[2], Yi Chang[2], Carl A. Gunter[1], Jiawei Han[1]
[1]University of Illinois at Urbana-Champaign, IL, USA, [2]Yahoo! Labs, Sunnyvale, CA, USA
[3]University of Massachusetts Amherst, MA, USA
{lzhang74, cgunter, hanj}@illinois.edu,
{goyal, hbdeng, anlei, yichang}@yahoo-inc.com, wkong@cs.umass.edu

ABSTRACT

Query auto-completion (QAC) facilitates user query composition by suggesting queries given query prefix inputs. In 2014, global users of Yahoo! Search saved more than 50% keystrokes when submitting English queries by selecting suggestions of QAC.

Users' preference of queries can be inferred during user-QAC interactions, such as dwelling on suggestion lists for a long time without selecting query suggestions ranked at the top. However, the wealth of such implicit negative feedback has not been exploited for designing QAC models. Most existing QAC models rank suggested queries for given prefixes based on certain relevance scores.

We take the initiative towards studying implicit negative feedback during user-QAC interactions. This motivates re-designing QAC in the more general "(static) relevance–(adaptive) implicit negative feedback" framework. We propose a novel adaptive model adaQAC that adapts query auto-completion to users' implicit negative feedback towards unselected query suggestions. We collect user-QAC interaction data and perform large-scale experiments. Empirical results show that implicit negative feedback significantly and consistently boosts the accuracy of the investigated static QAC models that only rely on relevance scores. Our work compellingly makes a key point: QAC should be designed in a more general framework for adapting to implicit negative feedback.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query Formulation*

Keywords

Query Auto-Completion; Implicit Negative Feedback

1. INTRODUCTION

Query auto-completion (QAC) helps user query composition by suggesting queries given prefixes. As illustrated in Fig. 1, upon a user's keystroke, QAC displays a *suggestion list* (or *list*) below the current *prefix*. We refer to queries in a suggestion list as *suggested queries* or *query suggestions*. A user can select to submit a suggested query; a user can also submit a query without selecting query suggestions. In 2014, global users of Yahoo! Search saved more than 50% keystrokes when submitting English queries by selecting suggestions of QAC.

Typically, a user *favors* and submits a query if it reflects the user's query intent in a query composition. However, predicting query intent is challenging. Many of the recently proposed QAC models rank a list of suggested queries for each prefix based on different relevance scores, such as popularity-based QAC (using historical query frequency counts) [2], time-based QAC (using time information) [32, 35], context-based QAC (using previous query information of users) [2], personalized QAC (using user profile information) [31], time-and-context-based QAC (using both time and previous query information of users) [5].

The aforementioned models use different relevance features but do not fully exploit user-QAC interactions, such as users' dwell time on suggestion lists and ranked positions of suggested queries by QAC. When users do not select query suggestions at keystrokes of compositions, users implicitly express negative feedback to these queries. Hence at such keystrokes, the user-QAC interaction information is users' *implicit negative feedback* to unselected queries. We aim at complementing relevance features with implicit negative feedback to improve the existing QAC models.

We start with a motivating example.

Motivating Example: Consider a user who wants to query Apple Inc.'s "facetime" with a popularity-based QAC [2]. When the user types "fac", "facebook" is ranked at the top in the suggestion list because it is most popular in historical query logs. The user dwells for a long time to examine the suggested query "facebook" but does not select it because it is not "facetime". However, in the next keystroke "e", popularity-based QAC still makes "facebook" top in the list because it is still the most popular query that matches

*Part of the work was completed at Yahoo! Labs.

*SIGIR'*15, August 9-13, 2015, Santiago, Chile.
©2015 ACM. ISBN 978-1-4503-3621-5/15/08...$15.00
DOI: http://dx.doi.org/10.1145/2766462.2767697

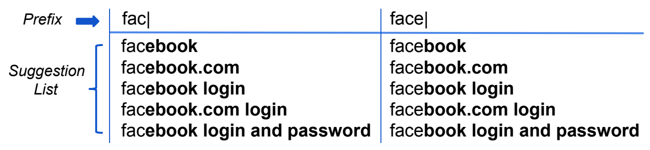


Figure 1: A commercial search engine QAC. Given prefixes "fac" and "face", popular "facebook"-related queries are suggested to users after being ranked by certain relevance scores.

the prefix "face". Fig. 1 depicts our interactions with a commercial search engine QAC known to depend on relevance scores only.

Here the user implicitly expresses negative feedback to "facebook": "facebook" is the top query suggestion, and the user dwells on the suggestion list for a long time without selecting this query. Hence, based on such implicit negative feedback, the user may not favor this unselected query. Can QAC be more accurate and demote "facebook" properly given the prefix "face"?

Our Approach: To the best of our knowledge, no existing QAC adapts its ranking of query suggestions to implicit negative feedback. We refer to a QAC model as **static QAC**, if its ranking of suggested queries does not adapt to implicit negative feedback in a query composition. Examples include popularity-based QAC, time-based QAC, and context-based QAC.

We go beyond static QAC by designing QAC in the new and more general "(static) relevance–(adaptive) implicit negative feedback" framework. In this framework, we propose a novel adaQAC model that adapts QAC to implicit negative feedback. adaQAC reuses the relevance scores of queries from static QAC to pre-index top-N queries. In a single query composition, adaQAC re-ranks these N queries at every keystroke based on users' implicit negative feedback. Personalized learning for every different user with batch inference is employed by adaQAC, and adaQAC can be extended by un-personalized learning and online inference.

Our Contributions: This work has many distinctions from related research in QAC, negative feedback, and dynamic information retrieval; we present detailed discussions on such distinctions in §5. Our contributions are summarized as follows.

• To the best of our knowledge, this is the first study on implicit negative feedback in user-QAC interactions. We find that the strength of implicit negative feedback to unselected query suggestions can be inferred, and a simple model fails (§2).

• We go beyond static QAC under a general "(static) relevance–(adaptive) implicit negative feedback" framework: we propose a novel adaQAC model that adapts QAC to implicit negative feedback using personalized learning with batch inference, including un-personalized learning and online inference extensions (§3).

• We collect user-QAC interaction data from a commercial search engine and perform large-scale experiments. We show implicit negative feedback significantly and consistently boosts the accuracy of the investigated static QAC models (§4).

2. IMPLICIT NEGATIVE FEEDBACK IN USER-QAC INTERACTIONS

We study QAC log data from a commercial search engine and discuss several motivational observations on implicit negative feedback in user-QAC interactions.

Terminology: In general, on search engines, queries are *submitted* upon users' *selection* from suggestion lists. Below are the other used terms.

Query composition (Composition): The duration of composing and submitting a single query. It starts from the keystroke of a new query's first character, or from the keystroke starting to edit a previous query. It ends when a query is submitted.

Dwell time: The time dwelled on a suggestion list. It is the time gap between two immediate keystrokes in a query composition.

Position: The ranked position of a query in a suggestion list by QAC. Position 1 means being ranked highest or at the top, while 10 corresponds to being ranked lowest or at the bottom.

QAC log data: Our collected user-QAC interaction data from Yahoo! Search. There are 2,932,035 query compositions via desktops and they are sampled over five months in 2014. A composition has the prefixes, timestamps and suggested queries of every keystroke, and the submitted query. More details of the data are in §4.1. Due to the proprietary nature of the data, some details are omitted in data descriptions and figures.

2.1 Implicit Negative Feedback

Typically, a user favors and submit a query that reflects the user's query intent in a query composition. We make the following assumption.

ASSUMPTION 1. *In a query composition, a user submits a query if and only if the user favors it.*

When a suggestion list is displayed, a user may examine or ignore a suggested query [21]. If a user ignores and does not select a suggested query, whether the user favors this query is unknown. If a user examines a suggested query but does not select it, there are two usual cases: (1) the user does not favor it; (2) the user still favors the suggestion but the user thinks selecting it from the list is less convenient than typing. In spite of possibly complicated cases, under Assumption 1 we make the following assumption.

ASSUMPTION 2. *In a query composition, suppose a user favors a suggested query. For the user, the likelihood of selecting this query is proportional to the likelihood of examining the query.*

Suppose a user examines a suggested query in a composition with a higher likelihood. From Assumption 2, if the user favors the query, the user selects it with a higher likelihood. Otherwise, if the user does not select a suggested query after examining the query, it hints that the user may not favor this query. Under Assumption 1, this user may not submit this unfavored query in the composition. Hence, the examined but unselected query may be demoted at the subsequent keystrokes in the same composition; it allows the user's favored query to rank higher in the composition.

Therefore, in a composition when a user does not select a suggested query, it may be helpful to know whether the user examines the unselected query. In other words, if the user examines an unselected query with a higher likelihood, this query may be demoted more heavily at the subsequent keystrokes of the composition.

For an unselected query suggestion, although whether a user examines it, is not observed, user-QAC interactions can be observed. Such interaction information includes user behavior (dwell time) and settings (position) that are observed during the interactions.

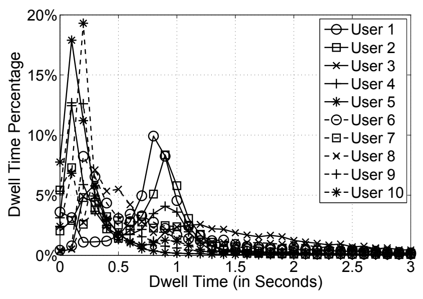
Implicit negative feedback from a user to an unselected query suggestion is observed user-QAC interaction information, when the query is suggested to the user upon keystrokes of a composition. In other words, a user can implicitly express negative feedback to an unselected query "facebook": "facebook" is the top query suggestion, and the user dwells on the list for long without selecting it.

We claim that implicit negative feedback can be strong or weak, and its *strength* cannot be directly observed thus has to be inferred. The properly inferred implicit negative feedback strength may be used to properly demote unselected query suggestions. Recall the discussion that "if the user examines an unselected query with a higher likelihood, this query may be demoted more heavily". Some implicit negative feedback may indicate the likelihood of a user's examination of an unselected query suggestion. Hence, such feedback is of interest. Important examples are dwell time and position.

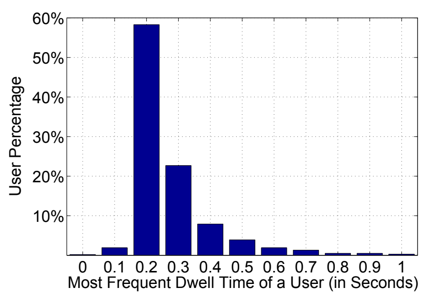
2.2 Dwell Time

If a user dwells on a suggestion list for a longer time, the user may have more time to carefully examine the suggested queries.

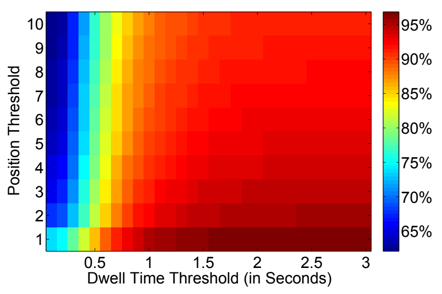
On the other hand, if a user dwells for a shorter time, more likely the suggested queries are ignored; thus, even if these queries are unselected, whether the user favors them is unknown.



(a) Distributions of dwell time from 10 randomly sampled users from QAC log data.



(b) Histogram of the most frequent dwell time of a user. Bins with a user percentage below 0.25% are omitted.



(c) Percentage of the unselected query suggestions that are not submitted by users.

Figure 2: Dwell time and position study. In (a) and (b), Value t at the horizontal axis corresponds to the dwell time bin $[t, t + 0.1)$.
(a) The two peak clusters imply two broad groups of users in the figure: User 1 and 2 generally type slower than the rest;
(b) The distribution shows that different users may have different typing speed;
(c) The percentage varies with different combinations of dwell time and position thresholds. Red color (wide on the right) corresponds to a higher percentage while blue color (narrow on the left) corresponds to a lower percentage. With a longer dwell time and a higher position, the likelihood that an unselected query suggestion will not be submitted by users at the end of query compositions is higher.

Fig. 2(a) elucidates the distributions of the 0.1-second dwell time bin between 0 and 3.1 seconds of 10 randomly sampled users from QAC log data[1]. Dwell time t (in seconds) falls in the bin $[t, t + 0.1)$. As the peak shows the most frequent dwell time bin of a user, it may suggest the user's comfortable typing speed: if the peak falls in the bin of a longer dwell time, the user's general typing speed is slower. The observed heavy-tails of the distributions manifest that longer dwell time is generally rarer, and the peak can characterize the user's typing speed. Thus, in Fig. 2(a), the two peak clusters may imply two broad groups of users: User 1 and 2 generally type slower than the rest.

Fig. 2(b) zooms out from 10 users' dwell time distributions to all the users' implied comfortable typing speed with a distribution for dwell time of the peaks in Fig. 2(a). It demonstrates that different users may have different typing speed. Hence, inference of implicit negative feedback strength by dwell time should be personalized.

2.3 Dwell Time and Position as Implicit Negative Feedback

We study dwell time and position of unselected query suggestions that are not submitted by users.

The suggested queries at all the keystrokes in query compositions are collected. Then, suggested queries at the final keystrokes in query compositions are excluded because users may select a suggested query at the final keystroke: only the percentage of unselected queries that are not submitted by users is of interest.

Suppose a dwell time threshold T_{DT} and a position threshold T_P are set up. Consider all the suggested queries $\mathbf{Q}(T_{DT}, T_P)$ that are, both in the list that is dwelled for no shorter than T_{DT}, and, ranked at positions no lower than T_P (dwell time $\geq T_{DT}$ and position $\leq T_P$). Given T_{DT} and T_P, $\forall q \in \mathbf{Q}(T_{DT}, T_P)$, the percentage of occurrences of q that are not submitted by users at the end of query compositions is recorded. The recorded results are illustrated in Fig. 2(c), with 300 different combinations of T_{DT} and T_P values, where $T_{DT} \in \{0.1, 0.2, \ldots, 3.0\}$ and $T_P \in \{1, 2, \ldots, 10\}$.

Recall Assumption 1 that a user submits the favored query in a composition. The percentage of users' unselected query suggestions that are not favored by them, can be interpreted by the corresponding color in Fig. 2(c). As discussed in §2.1, implicit negative feedback strength may indicate how to demote unselected queries. For a more accurate QAC, the demotion should properly reflect the likelihood of not favoring or submitting an unselected query: such likelihood is higher with a longer dwell time and a higher position, as shown in Fig. 2(c). Thus, the results in Fig. 2(c) support the hypothesis that dwell time and position are important to infer the strength of implicit negative feedback.

From Fig. 2(c), when a position threshold T_P is fixed, a dwell time threshold T_{DT} better differentiates the likelihood of not favoring or submitting an unselected query, when $0 < T_{DT} < 1$. This agrees with the results in Fig. 2(a)—2(b) that, longer dwell time is generally rarer.

2.4 Filtering Queries by Thresholds Fails

Following the findings in Fig. 2(c), it is tempting to extend an existing QAC model by filtering out all the suggested queries based on dwell time and position thresholds. Thus, we set up a baseline model **Filtering QAC** to filter out all the suggested queries by using fixed dwell time and position thresholds in the subsequent keystrokes of a query composition. For instance, for $T_{DT} = 2$ and $T_P = 3$, any previously suggested queries with positions higher than or equal to 2 and dwell time longer than or equal to 3 seconds are not suggested anymore in the subsequent keystrokes of the same query composition. To ensure a higher ranking accuracy, the results of Filtering QAC are tuned among $T_{DT} \in \{0.1, 0.2, \ldots, 3.0\}$ and $T_P \in \{1, 2, \ldots, 10\}$.

However, experimental results (§4.2) show this simple model fails to significantly boost the static QAC models.

3. ADAPTIVE QUERY AUTO-COMPLETION

Motivated by the findings from large commercial search engine QAC log data in §2, we propose a novel **adaQAC** model that adapts query auto-completion to implicit negative feedback.

3.1 Method Overview

We describe the system design of adaQAC to rank the suggested queries for a given prefix. A toy example with two queries "face-

[1]Binning masks details of the data for its proprietary nature.

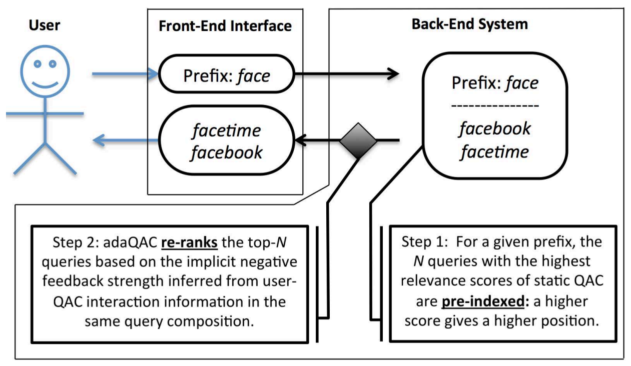

User
Front-End Interface
Back-End System
Prefix: face
facetime
facebook
Prefix: face
facebook
facetime
Step 2: adaQAC re-ranks the top-N queries based on the implicit negative feedback strength inferred from user-QAC interaction information in the same query composition.
Step 1: For a given prefix, the N queries with the highest relevance scores of static QAC are pre-indexed: a higher score gives a higher position.

Figure 3: The system design and data flow of adaQAC

book" and "facetime" that match prefixes "fac" and "face" at top positions is used to illustrate the idea. Fig. 3 explains the system design and data flow of adaQAC: it has two stages.

Stage 1 (Pre-indexing): For a given prefix, top-N query suggestions with the highest relevance scores of static QAC are pre-indexed: the higher score, the higher position. In Fig. 3, for the prefix "face", the top-2 ($N = 2$) queries "facebook" and "facetime" are pre-indexed by static QAC based on the historical query frequency counts.

Stage 2 (Re-ranking): adaQAC re-ranks these top-N queries based on the implicit negative feedback strength inferred from user-QAC interaction information in the same composition. To illustrate Stage 2, upon a keystroke "e" following the prefix "fac" from a user, the front-end interface takes the current prefix "face" as an input and immediately fetches the pre-indexed queries "facebook" and "facetime". Suppose when "facebook" was ranked highest in the suggestion list at the prefix "fac", the user dwells for a long time but does not select it. With this observation, suppose adaQAC is able to infer the user's implicit negative feedback strength. Thus, adaQAC updates the ranking score of "facebook" and re-ranks the top 2 ($N = 2$) queries "facebook" and "facetime". With re-ranking, "facetime" is now at Position 1, after "facebook" is demoted to Position 2.

The number of the pre-indexed top queries N can be set to a small positive integer in a production, such as 10 in our experiments. With a small constant value N, sorting N queries based on the updated ranking scores can be achieved in constant time [7].

3.2 "(Static) relevance–(Adaptive) Implicit Negative Feedback" Framework

We highlight that, adaQAC is designed in a more general **"(static) relevance–(adaptive) implicit negative feedback" framework**. The "relevance" component is the relevance score of a query for a user that can be obtained from an existing static QAC model, such as popularity-based QAC; the other "implicit negative feedback" component adapts QAC to implicit negative feedback.

The "(static) relevance–(adaptive) implicit negative feedback" framework is more general for both reusing existing static QAC research and adapting QAC to the newly discovered implicit negative feedback. In this framework, adaQAC is not constrained to employ a certain relevance score: in §4 we investigate several different relevance scores with different parameter values in these scores.

3.3 Problem Formulation

Consider a user $u \in \mathbf{U}$, where $\mathbf{U}$ is the set of all adaQAC users, at the k-th keystroke in a query composition $c \in \mathbf{C}(u)$, where $\mathbf{C}(u)$

Table 1: Main Notations

Symbol	Description
$\mathbf{U}$	User set.
u	User.
$\mathbf{C}(u)$	Query composition set of a user u.
c	Query composition.
$K(c)$	Number of keystrokes in a query composition c.
k	Keystroke index: $k \in \{1, 2, \ldots, K(c)\}$.
$\mathbf{Q}$	Query set.
q, q'	Query.
$q^*(c)$	Submitted query in a query composition c.
$r^{(k)}(u, q, c)$	Relevance score for a user u of a query q that matches the prefix at a keystroke k in a query composition c.
$\mathbf{Q}^{(k)}(r, u, c, N)$	Set of top N queries ranked by $r^{(k)}(u, q, c)$.
$\mathbf{x}^{(k)}_{l\times 1}(u, q, c)$	Implicit negative feedback feature vector from a user u to a query q at a keystroke k in a query composition c.
$\mathbf{\Phi}_{l\times m}(\mathbf{U})$	Implicit negative feedback feature weight matrix for a user set $\mathbf{U}$.
$\boldsymbol{\phi}_{l\times 1}(u)$	Implicit negative feedback feature weight vector for a user u.
$p^{(k)}(u, q, c)$	Preference for a query q of a user u at a keystroke k in a query composition c.
λ	Regularizer weight parameter.

Table 2: Feature descriptions of the adaQAC model. The implicit negative feedback feature vector $\mathbf{x}^{(k)}(u, q, c)$, from a user u to a query q at a keystroke k in a query composition c, contains the following information collected from the beginning of c to the $(k-1)$-th keystroke in c.

Feature	Description
DwellT-M	The maximum dwell time when q is suggested.
DwellT	Total dwell time where q is suggested.
WordBound	No. of the keystrokes at word boundaries when q is suggested.
SpaceChar	No. of the keystrokes at space characters when q is suggested.
OtherChar	No. of the keystrokes at non-alphanum. char. when q is suggested.
IsPrevQuery	1 if q is the immediately previous query; 0 otherwise.
Pos@i	No. of the keystrokes when q is at Position i of a suggestion list ($i = 1, 2, \ldots, 10$).

*Dwell time greater than 3 seconds at one suggestion list is set to 3 seconds.

is the query composition set of u. adaQAC suggests a ranked list of queries in $\mathbf{Q}$ according to the ranking scores determined by a probabilistic model. The probabilistic model is based on a combination of the relevance score and the inferred strength of implicit negative feedback. For a query q that matches the prefix at the keystroke k in the query composition c, the relevance score of q for the user u is denoted as $r^{(k)}(u, q, c)$.

Implicit negative feedback from the user u to the query q at the k-th keystroke in the query composition c is represented by a feature vector $\mathbf{x}^{(k)}_{l\times 1}(u, q, c)$, where l is the number of features. The strength of implicit negative feedback is based on $\mathbf{x}^{(k)}_{l\times 1}(u, q, c)$ and its associated implicit negative feedback feature weight vector $\boldsymbol{\phi}_{l\times 1}(u)$ for u. $\boldsymbol{\phi}_{l\times 1}(u)$ is a column vector indexed by u from the implicit negative feedback feature weight matrix $\mathbf{\Phi}_{l\times m}(\mathbf{U})$ for all the users in $\mathbf{U}$. Here m is the number of users in $\mathbf{U}$.

In a query composition c, prefixes with the corresponding suggestion lists are referred to by sequential keystroke indices $k \in \{1, 2, \ldots, K(c)\}$, where $K(c)$ is the number of keystrokes in a query composition c. For instance, for a query composition c starting from an empty string with three keystrokes "fac" ($K(c) = 3$), the prefix "fac" with the suggestion list in the left of Fig. 1 can be referred to by $k = 3$ in c or simply $K(c)$ in c. Table 1 briefly summarizes the main notations.

3.4 Personalized Learning

Table 2 lists the features used by adaQAC to fit in the "implicit negative feedback" component. Dwell time and positions are studied in §2.3. Likewise, the other features also indicate how likely users examine query suggestions.

Based on §2.2, such as the observation that different users may have different typing speed, personalized learning is used: $\boldsymbol{\phi}(u)$ is to be learned separately for each $u \in \mathbf{U}$ to form $\boldsymbol{\Phi}(\mathbf{U})$.

3.4.1 Probabilistic Model

We model preference $p^{(k)}(u, q, c)$ for a query q of a user u at a keystroke k in a query composition c, by a generalized additive model [10]:

$$p^{(k)}(u, q, c) = r^{(k)}(u, q, c) + \boldsymbol{\phi}^\top(u)\mathbf{x}^{(k)}(u, q, c). \tag{1}$$

In Equation 1, 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 a query composition c. With the associated feature weights $\boldsymbol{\phi}(u)$ personalized for u, $\boldsymbol{\phi}^\top(u)\mathbf{x}^{(k)}(u, q, c)$ encodes the strength of implicit negative feedback to q from u with personalization.

When a user u submits a query $q^*(c)$ at the final keystroke $K(c)$ in a query composition c, c ends. The likelihood of the observations on the submitted query in a query composition together with implicit negative feedback in Table 2 is to be maximized. Hence, we define a probabilistic model for a submitted query $q^*(c)$ by u at $K(c)$ in c with a softmax function that represents a smoothed version of the "max" function [3, 38]:

$$\mathbb{P}\Big(Q = q^*(c) \mid u, c, K(c)\Big) = \frac{\exp\Big[p^{\big(K(c)\big)}\Big(u, q^*(c), c\Big)\Big]}{\sum_{q \in \mathbf{Q}^{(k)}(r,u,c,N) \bigcup \{q^*(c)\}} \exp\Big[p^{\big(K(c)\big)}(u, q, c)\Big]}, \tag{2}$$

where $\mathbf{Q}^{(k)}(r, u, c, N)$ represents the set of top N queries ranked by $r^{(k)}(u, q, c)$. Its union with $\{q^*(c)\}$ ensures proper normalization. Likewise, adaQAC predicts the likelihood that a query $q' \in \mathbf{Q}^{(k)}(r, u, c, N)$ to be submitted by a user u at any k in c by

$$\mathbb{P}\Big(Q = q' \mid u, c, k\Big) = \frac{\exp\Big[p^{(k)}(u, q', c)\Big]}{\sum_{q \in \mathbf{Q}^{(k)}(r,u,c,N)} \exp\big[p^{(k)}(u, q, c)\big]} \propto p^{(k)}(u, q', c). \tag{3}$$

In practice, the simpler form $p^{(k)}(u, q', c)$ in Equation 3 is used for re-ranking in Stage 2 of adaQAC (§3.1) after $\boldsymbol{\phi}(u)$ in Equation 1 is inferred. If a query q never appears in any suggestion list before a keystroke k in a query composition c, $\mathbf{x}^{(k)}(u, q, c)$ is a zero vector and the user u's preference for q is the same as the relevance score $r^{(k)}(u, q, c)$. Here k, c are used to refer to the prefix at k in c and suggested queries must match the prefix. However, if u expresses possibly stronger implicit negative feedback to q before k in c, say q is dwelled longer and at a higher position for several times, then the corresponding weights in $\boldsymbol{\phi}(u)$ updates preference for q of u at k in c with a lower $p^{(k)}(u, q, c)$ value; while possibly weaker implicit negative feedback may correspond to shorter dwell time and a lower position. The strength of the expressed implicit negative feedback determines the level of penalizing u's preference for q in $p^{(k)}(u, q, c)$, which affects how to re-rank in Stage 2 of adaQAC. This agrees with the earlier discussions on using proper implicit negative feedback strength to properly demote an unselected query suggestion (§2).

We highlight that, the preference model $p^{(k)}(u, q, c)$ in Equation 1 is designed in the more general framework as discussed in §3.2. The "(static) relevance" component is $r^{(k)}(u, q, c)$, and $\boldsymbol{\phi}^\top(u)\mathbf{x}^{(k)}(u, q, c)$ acts as "(adaptive) implicit negative feedback".

3.4.2 Batch Inference

In Equation 1 $\boldsymbol{\phi}(u)$ is inferred with batch inference. The likelihood for all compositions $C(u)$ of a user u should be maximized.

$$\underset{\boldsymbol{\phi}(u)}{\text{maximize}} \quad \prod_{c \in \mathbf{C}(u)} \mathbb{P}\Big(Q = q^*(c) \mid u, c, K(c)\Big). \tag{4}$$

By Equation 2 and 4, a constrained optimization problem out of minimizing negative log-likelihood with $L2$ regularization (to avoid overfitting) is obtained as

$$\begin{aligned} \underset{\boldsymbol{\phi}(u)}{\text{minimize}} \quad & \sum_{c \in \mathbf{C}(u)} \log \sum_{q \in \mathbf{Q}^{(k)}(r,u,c,N) \bigcup \{q^*(c)\}} \exp\Big[p^{\big(K(c)\big)}(u, q, c)\Big] \\ & - p^{\big(K(c)\big)}\big(u, q^*(c), c\big) \\ \text{subject to} \quad & \|\boldsymbol{\phi}(u)\|_2^2 \le v, \ v \in \mathbb{R}^+. \end{aligned} \tag{5}$$

There is a one-to-one correspondence between the parameters v in Equation 5 and $\lambda \in \mathbb{R}^+$, and the corresponding un-constrained optimization problem is:

$$\begin{aligned} \underset{\boldsymbol{\phi}(u)}{\text{minimize}} \quad & \sum_{c \in \mathbf{C}(u)} \log \sum_{q \in \mathbf{Q}^{(k)}(r,u,c,N) \bigcup \{q^*(c)\}} \exp\Big[p^{\big(K(c)\big)}(u, q, c)\Big] \\ & - p^{\big(K(c)\big)}\big(u, q^*(c), c\big) + \frac{\lambda}{2}\|\boldsymbol{\phi}(u)\|_2^2, \end{aligned} \tag{6}$$

where λ is the regularizer weight parameter. As there is no closed-form solution for the optimization problem in Equation 6 due to non-linearity of the softmax function [3], iterative batch inference by gradient descent is used. We refer to an adaQAC model using personalized learning with batch inference as **adaQAC-Batch**. Details for inferring $\boldsymbol{\phi}(u)$ are in Appendix A.

3.4.3 Optimum and Convergence

The objective function of negative log-likelihood for softmax functions with $L2$ regularization in Equation 6 is strongly convex [28]. Hence, the inference is guaranteed to converge to the global optimum [29]: adaQAC-Batch can be inferred precisely. As we know, for a strongly convex objective function $f(x)$ whose optimal value is achieved with $x = x^*$, the number of iterations to get to accuracy $|f(x^*) - f(x)| \le \epsilon$ takes a $\mathcal{O}(\ln(\frac{1}{\epsilon}))$ time [4]. Our experiments in §4.3 reinforce that, adaQAC-Batch converges quickly and reaches the global optimum within a constant number of iterations.

3.4.4 Computational Complexity

Suppose the relevance scores of queries for users, which depend on static QAC, are available. During the training phase for a user u, $\boldsymbol{\phi}(u)$ is inferred with the constructed feature vectors. Assuming the number of queries in a suggestion list and the number of top queries for re-ranking (N in §3.1) are fixed small constants, the feature construction has a time complexity of $\mathcal{O}(lK(c))$, where l is the feature vector size and $K(c)$ is the number of keystrokes in a query composition c. Since the inference algorithm in Appendix A converges within a constant number of steps (§3.4.3), it takes a $\mathcal{O}(l^2\,|\mathbf{C}(u)|)$ time with a constant factor corresponding to the number of convergence steps or a predefined value. Here $|\mathbf{C}(u)|$ is the number of query compositions for a user u. Note that the features in Table 2 are all distributive functions: the result derived by applying the function to aggregate values is the same as that derived by applying the function on all the data without partitioning. To explain, let $\mathbf{x}_i^{(k)}(u, q, c)$ be *DwellT-M*$^{(k)}(u, q, c)$;

DwellT-M$^{(k+1)}(u, q, c)$ can be updated by simply taking the larger value of *DwellT-M*$^{(k)}(u, q, c)$ and the dwell time at $k+1$ in c, if q appears in the suggestion list. With a fixed small constant value N (§3.1), the suggestion at each keystroke takes a $\mathcal{O}(l)$ time.

3.4.5 Scalability on Hadoop MapReduce

A nice property of personalized learning is scalability. As adaQAC-Batch infers $\boldsymbol{\phi}(u)$ for each individual user u, the inference is parallel for different users on big query log data.

In particular, in the Hadoop MapReduce framework, the $\boldsymbol{\Phi}(\mathbf{U})$ inference phase of our experiments is conducted in parallel for different users by different Reducer nodes.

3.5 Extensions

For a user u, adaQAC-Batch requires training data related to u to infer the feature weight $\boldsymbol{\phi}(u)$. Now we consider a more challenging cold-start scenario where u is a new user without related data for training. Two ways of extensions can address the challenge.

3.5.1 Un-Personalized Learning

The first way is to infer the feature weights from all the existing users excluding the new user. To maintain scalability on Hadoop MapReduce, a gradient descent variant with averaging is used [38]. This un-personalized approach does not differentiate one user from another, and is referred to as **adaQAC-UnP**.

Because only one feature weight vector is stored and shared by all the users, adaQAC-UnP is cheap in storage.

3.5.2 Online Inference

adaQAC-Batch can be extended to an online inference style. For a new user, first, assign the un-personalized learning output to initialize the feature weights; then, keep update the feature weights with more observations of the user's interactions with QAC.

We call this personalized online learning style extension **adaQAC-Online**. Stochastic gradient descent is used for the online inference. It is similar to batch inference with the constrained optimization problem out of minimizing negative log-likelihood with $L2$ regularization in Equation 5 replaced by

$$\begin{aligned}
\underset{\boldsymbol{\phi}(u)}{\text{minimize}} \quad & \log \sum_{q \in \mathbf{Q}^{(k)}(r,u,c,N) \bigcup \{q^*(c)\}} \exp\left[p^{\left(K(c)\right)}(u, q, c)\right] \\
& - p^{\left(K(c)\right)}(u, q^*(c), c) \\
\text{subject to} \quad & \|\boldsymbol{\phi}(u)\|_2^2 \leq v,\ v \in \mathbb{R}^+.
\end{aligned}$$

Details for inferring $\boldsymbol{\phi}(u)$ are in Appendix B.

Cost Analysis: adaQAC-Online costs more storage than adaQAC-UnP due to maintaining different weights for all the users. As shown in §4.4, adaQAC-Online trades its storage cost for slightly higher accuracy than adaQAC-UnP. Compared with adaQAC-Batch, the inference of adaQAC-Online takes a $\mathcal{O}(tl^2)$ time, where t is the number of observations and l is the feature vector size. Generally adaQAC-Online takes less time than adaQAC-Batch in inference and has the same storage requirement for maintaining different feature weights for all the users. Comparing with adaQAC-Batch, adaQAC-UnP takes the same order of time with less storage requirement as it maintains only one feature weight vector that is shared by all the users.

4. EVALUATION

We evaluate the proposed adaQAC-Batch and its two extensions adaQAC-UnP and adaQAC-Online on QAC log data.

4.1 Data and Evaluation Measures

Data: We describe important details of our collected QAC log data. Due to the proprietary nature of the data, some details are omitted. The QAC log data are collected from Feb 28 to Jul 28, 2014 and all the queries are submitted via desktops. If a query is submitted by more than two different users, its corresponding query composition is used for evaluation. As adaQAC-Batch requires training data for the feature weight inference, all the users with fewer than 100 query compositions during the given five-month range are filtered out. After the filtering, users are randomly sampled and their 2,932,035 query compositions constitute the evaluation data. There are in total 481,417 unique submitted queries. All the query compositions have their anonymized user IDs and the submitted queries. In one composition, the prefixes, timestamps and suggested queries of every keystroke are collected.

The training, validation and testing data are split with a ratio of 50%/25%/25% in an ascending time order: the first half of a user's query compositions are used for training; the second and third quarters are for validation and testing respectively. The validation data are only used for parameter tuning. As adaQAC infers implicit negative feedback from user-QAC interactions in query compositions, in §4.2—§4.5 we experiment on the prefixes at the last keystroke of query compositions to use more interaction information. The average length of query prefixes is 8.53 characters.

The data standardization procedure is transforming data to zero mean and unit variance. All the feature values in Table 2 and the relevance scores are standardized.

Measures for Accuracy: Mean reciprocal rank (MRR) is the average reciprocal of the submitted query's ranking in a suggestion list. It is a widely-adopted measure to evaluate the ranking accuracy of QAC [2, 21, 15, 31]. Success Rate@top-k (SR@k) denotes the average percentage of the submitted queries that can be found in the top-k suggested queries on the testing data, and was also used to evaluate the QAC ranking accuracy [15]. In general, a higher MRR or SR@k indicates a higher ranking accuracy of QAC [2, 21, 15, 31, 5]. Paired-t test is used to validate the statistical significance of the accuracy improvement ($p < 0.05$).

4.2 Boosting the Accuracy of Static QAC with Implicit Negative Feedback

Following the "(static) relevance–(adaptive) implicit negative feedback" framework (§3.2), we investigate relevance scores from popular static QAC with different parameter settings to compare the accuracy of adaQAC-Batch, Filtering QAC, and static QAC.

The relevance scores reuse the existing research: MPC [2, 15, 21, 31], Personal(-S) [2, 5, 31], and TimeSense(-S) [5, 32, 35, 27].

• **MPC:** Most Popular Completion (MPC) ranks suggested queries for a prefix based on the historical popularity of a query. A more popular query gets a higher rank. Despite its simplicity, it was found competitive by various studies [2, 15, 21, 31].

• **Personal:** Personal QAC for distinguishing different users can achieve better accuracy [2, 5, 31]. Although personal information may take many different forms, the Personal relevance score in this work is an equal-weighted linear combination of the MPC score and the standardized personal historical query frequency counts.

• **Personal-S:** It is the Personal relevance score with an optimal combination with different weights of the MPC score and the standardized personal query frequency counts. The optimal weights achieving the highest accuracy are tuned on validation data. Tuning to the optimal weights makes Personal-S more competitive.

• **TimeSense:** Time is useful in QAC [5, 32, 35]. Hence, TimeSense is the same as Personal except that the personal historical

Table 3: Accuracy comparison of static QAC, Filtering QAC, and adaQAC-Batch (in percentage). Boldfaced results denote that the accuracy improvement over static QAC is statistically significant ($p < 0.05$) for the same relevance score. adaQAC-Batch significantly and consistently boosts the accuracy of static QAC for each relevance score. For instance, adaQAC-Batch (MPC) significantly boosts static QAC (MPC) by 21.2% in MRR.

Relevance	MRR			SR@1			SR@2			SR@3		
	Static	Filter	adaQAC-Batch	Static	Filter	adaQAC-Batch	Static	Filter	adaQAC-Batch	Static	Filter	adaQAC-Batch
MPC	50.62	51.83	**61.33 (+21.2%)**	40.74	42.27	**55.86 (+37.1%)**	52.03	53.19	**63.17 (+21.4%)**	58.09	59.21	**66.09 (+13.8%)**
Personal	61.85	62.68	**70.97 (+14.8%)**	51.31	52.45	**64.27 (+25.3%)**	64.02	64.78	**73.71 (+15.1%)**	70.34	71.09	**76.94 (+9.4%)**
Personal-S	66.02	66.52	**74.43 (+12.7%)**	55.30	56.24	**67.09 (+21.3%)**	68.51	68.92	**77.73 (+13.5%)**	74.58	74.97	**80.97 (+8.6%)**
TimeSense	64.32	65.14	**73.70 (+14.6%)**	53.77	54.92	**66.82 (+24.3%)**	66.54	67.45	**76.41 (+14.8%)**	72.39	73.28	**79.81 (+10.3%)**
TimeSense-S	65.56	66.19	**74.69 (+13.9%)**	55.02	56.11	**67.57 (+22.8%)**	67.83	68.27	**77.76 (+14.6%)**	73.68	74.11	**80.97 (+9.9%)**

*Static: Static QAC; Filter: Filtering QAC

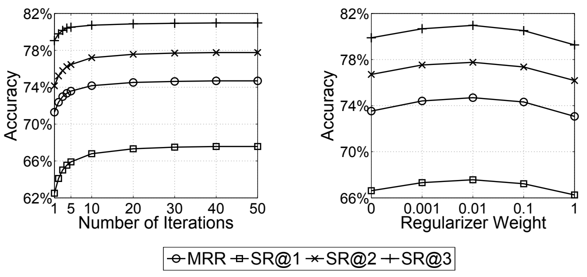


Figure 4: Convergence (left) and regularizer weight (right) study for adaQAC-Batch (TimeSense-S). Plots are similar for the other relevance scores. adaQAC-Batch converges quickly and is not sensitive to the chosen regularizer weight near its optimum.

query frequency counts is replaced by the all-user popularity counts of a query in the 28-day time window before a query composition.

• **TimeSense-S:** It is the same as Personal-S except that Personal is replaced by TimeSense.

For brevity, we denote "static QAC employing the MPC relevance score" as "Static (MPC)". Similar notations are used for QAC models employing any relevance score.

Parameters values are tuned to achieve the highest accuracy on validation data. Unless otherwise stated we set the number of iterations to 40 (adaQAC-Batch and adaQAC-UnP) and the regularizer weight to 0.01. Personal-S and TimeSense-S both combine a MPC score with the optimal weight α and the other score with the weight $1 - \alpha$. The optimal weights in Personal-S ($\alpha = 0.34$) and TimeSense-S ($\alpha = 0.42$) achieve the highest MRR for static QAC.

In §2.4 we set up Filtering QAC with relevance scores, by additionally filtering out all the suggested queries with certain dwell time thresholds (T_{DT}) and position thresholds (T_P) in the subsequent keystrokes in a composition. To ensure higher competitiveness, the model is tuned among the 300 threshold value combinations in §2.4. We set $T_{DT} = 0.9$ and $T_P = 1$.

Table 3 presents the accuracy comparison of static QAC, Filtering QAC, and adaQAC-Batch. The simple Filtering QAC model fails to outperform the corresponding static QAC with the same relevance scores significantly. For each same relevance score, adaQAC-Batch exploiting the added implicit negative feedback information significantly and consistently boosts the accuracy of these static QAC models that only use relevance scores. With more accurate relevance scores such as Personal and TimeSense, adaQAC-Batch is more accurate. Given the relevance scores with different parameter settings (Personal *vs.* Personal-S and TimeSense *vs.* TimeSense-S), the accuracy of adaQAC-Batch slightly varies depending on the accuracy of the relevance scores for the chosen parameter values.

The newly-discovered implicit negative feedback is promising in boosting the accuracy of the existing static QAC models.

4.3 Parameter Study

Here we set the number of iterations and regularizer weight to different values for the parameter study on the validation data. adaQAC-Batch (TimeSense-S) is tested. The results for the other relevance scores are similar.

Convergence: Fig. 4 (left) shows the evaluation measures against the number of iterations. The results reinforce the fact that, adaQAC-Batch converges quickly and the precise global optimum can be reached within a constant number of iterations (§3.4.3).

Regularizer Weight: Fig. 4 (right) plots the evaluation measures of adaQAC-Batch (TimeSense-S) with regularizer weights that are varied around the optimum 0.01. adaQAC-Batch is not sensitive to different regularizer weights near the optimum. This property shows that the accuracy of adaQAC-Batch has less dependence on the chosen regularizer weight value.

4.4 Un-Personalized Learning and Online Inference

Motivated by the more challenging cold-start scenario where there is a lack of training data for new users, we evaluate the two adaQAC extensions adaQAC-UnP (§3.5.1) and adaQAC-Online (§3.5.2).

For a user u, the un-personalized learning is performed by learning from training data related to all the users excluding u, and the learned feature weights are fed into adaQAC-Online for u as the initial feature weights. Neither adaQAC-UnP nor adaQAC-Online uses the training data related to u.

Table 4 shows that, both adaQAC-UnP and adaQAC-Online significantly and consistently boost the accuracy of static QAC for each relevance score. The mean measure values of adaQAC-UnP and adaQAC-Online are slightly lower than those of adaQAC-Batch for the same relevance score. This slight difference can be justified by the added benefits of the more expensive personalized learning with batch inference of adaQAC-Batch.

It was pointed out that (§3.5.2), adaQAC-Online costs more storage than adaQAC-UnP due to maintaining different weights for all the users. The slight difference between the mean of the measure values of adaQAC-Online and adaQAC-UnP in Table 4 shows that, adaQAC-Online trades its storage cost for slightly higher accuracy than adaQAC-UnP. In addition to the benefits for addressing the cold-start challenge, according to the cost analysis in §3.5.2, an important practical implication from the results of Table 4 is, adaQAC-UnP and adaQAC-Online can be good substitutes for the more expensive adaQAC-Batch if time and storage budgets are limited in the real-world productions.

Table 4: Accuracy of adaQAC-UnP and adaQAC-Online in comparison with static QAC (in percentage). Boldfaced results denote that the accuracy improvement over static QAC is statistically significant ($p < 0.05$) for the same relevance score. Both of the adaQAC extension models significantly and consistently boost the accuracy of static QAC for each relevance score. For instance, adaQAC-Online (MPC) significantly boosts static QAC (MPC) by 20.3% in MRR.

Relevance	MRR		SR@1		SR@2		SR@3	
	adaQAC-UnP	adaQAC-Online	adaQAC-UnP	adaQAC-Online	adaQAC-UnP	adaQAC-Online	adaQAC-UnP	adaQAC-Online
MPC	**60.60 (+19.7%)**	**60.92 (+20.3%)**	**54.54 (+33.9%)**	**55.06 (+35.1%)**	**62.75 (+20.6%)**	**62.99 (+21.1%)**	**66.01 (+13.6%)**	**66.11 (+13.8%)**
Personal	**69.80 (+12.9%)**	**70.22 (+13.5%)**	**62.27 (+21.4%)**	**62.98 (+22.7%)**	**72.76 (+13.7%)**	**73.06 (+14.1%)**	**76.64 (+9.0%)**	**76.77(+9.1%)**
Personal-S	**73.16 (+10.8%)**	**73.59 (+11.5%)**	**64.87 (+17.3%)**	**65.59 (+18.6%)**	**76.83 (+12.1%)**	**77.11 (+12.6%)**	**80.65 (+8.1%)**	**80.81 (+8.4%)**
TimeSense	**72.69 (+13.0%)**	**73.05 (+13.6%)**	**65.00 (+20.9%)**	**65.61 (+22.0%)**	**75.69 (+13.8%)**	**75.97 (+14.2%)**	**79.64 (+10.0%)**	**79.74 (+10.2%)**
TimeSense-S	**73.57 (+12.2%)**	**73.96 (+12.8%)**	**65.65 (+19.3%)**	**66.26 (+20.4%)**	**76.78 (+13.2%)**	**77.13 (+13.7%)**	**80.74 (+9.6%)**	**80.91 (+9.8%)**

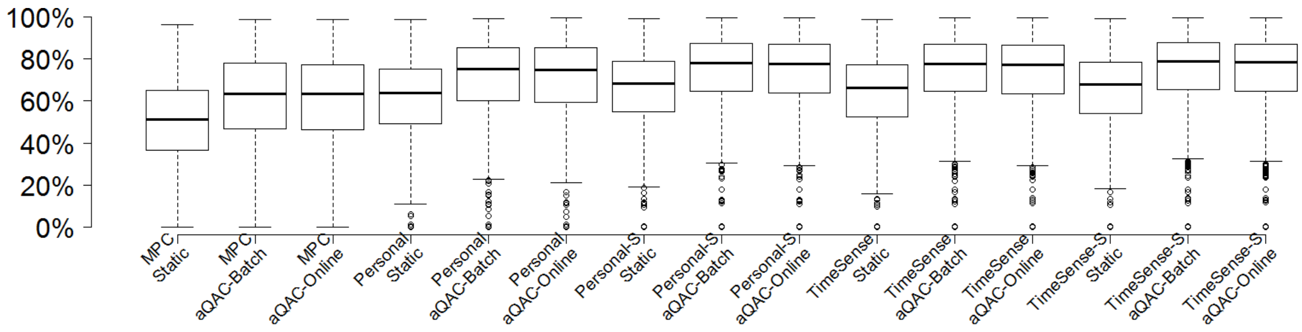


Figure 5: Box-and-Whisker plots of individual users' MRR for static QAC, adaQAC-Batch, and adaQAC-Online with five relevance scores. Each data instance is the corresponding MRR on one user. The minimum (bottom bar), quartiles (box edges), median (middle of the box), and maximum (top bar) after removal of the detected outliers (empty circles) are depicted. adaQAC with more accurate relevance scores are able to detect more outliers with the raised minimum bars.

4.5 Model Accuracy on Different Users

We study the model accuracy on different users using the Box-and-Whisker plots. With each data instance being the MRR on one user, Fig. 5 shows the minimum (bottom bar), quartiles (box edges), median (middle of the box), maximum (top bar) after removal of the detected outlier users (empty circles).

In general, model comparison using medians and quartiles of MRR agrees with the results in Table 3—4 and reaffirms the boosted accuracy by the added implicit negative feedback.

Note that, all the models perform poorly on a few users. Models with the MPC relevance score fail to detect any outlier, and have minimum bars close to 0. The other models still perform poorly on certain users with MRR close to 0. These users are detected as the outliers. The outlier users may behave inconsistently, submit rare queries, or the collected data related to them are noisy or incomplete due to unknown reasons.

To explain, models with the MPC relevance have a larger MRR variance (implied by a longer box in Fig. 5) so outlier users cannot be easily detected. It is easier to see when comparing adaQAC-Online (MPC) with Static (Personal): they have close medians but the lower-variance Static (Personal) is able to detect a few outliers and raise its minimum bar after their removal. When the relevance score is more accurate with a lower variance, adaQAC is able to detect more outliers thus raises the minimum bar by further improving the MRR on the majority of the users.

Hence, even though the implicit negative feedback research is promising, further research on more accurate relevance scores is still required.

4.6 Varying-Length Prefix Study

Now we consider another challenging scenario where testing is based on all possible prefixes in query compositions. Table 5 reports MRR of static QAC, adaQAC-Static and adaQAC-Online for prefixes with varying lengths at every keystroke in query compositions. Both adaQAC-Batch and adaQAC-Online still significantly and consistently boost the accuracy of static QAC under all prefix lengths for each relevance score.

The MRR gap between adaQAC-Batch and adaQAC-Online is subtle and both are more accurate when prefixes are of "middle" lengths. That is, when the prefixes are short, the collected implicit negative feedback features probably contain little useful information to improve the re-ranking in Stage 2 of adaQAC (§3.1). When prefixes get longer, more user-QAC interaction information is obtained to make adaQAC more accurate in the adaptive re-ranking stage. However, when prefixes are longer, the QAC problem becomes less challenging due to a reduction of the matched queries: static QAC employing relevance scores are more accurate and it is harder to further improve the accuracy, even though the implicit negative feedback information may be richer.

4.7 Case Study

adaQAC has advantages over static QAC. We describe the following cases of Yahoo! Search, and hope that this work can inspire ongoing studies in a broader research community.

Disambiguation: When users have clear query intent and prefer disambiguated queries, adaQAC generally outperforms static QAC. Typically, users may prefer queries of the form "entity name + attribute" to "entity name only". Suppose a user wants to know the showtime of lefont sandy springs. When the user composes the query during the keystrokes "lefon", the entity name "lefont sandy springs" is the top suggestion. The user does not select it because an entity name query may result in diverse search results. So, the query "lefont sandy springs" receives implicit negative feedback. When the prefix becomes "lefont", "lefont sandy springs" is de-

Table 5: MRR of static QAC, adaQAC-Batch, and adaQAC-Online under prefixes with varying lengths at every keystroke in query compositions (in percentage). Boldfaced results denote that the accuracy improvement over static QAC is statistically significant ($p < 0.05$) for the same relevance score and prefix length range. Both adaQAC-Batch and adaQAC-Online significantly and consistently boost the accuracy of static QAC under all prefix lengths for each relevance score. For instance, adaQAC-Batch (MPC) significantly boosts static QAC (MPC) by 17.1% in MRR under all prefix lengths.

Relevance	Static	adaQAC-Batch	adaQAC-Online	Static	adaQAC-Batch	adaQAC-Online	Static	adaQAC-Batch	adaQAC-Online
	$1 \leq$ Prefix Length ≤ 3			$4 \leq$ Prefix Length ≤ 6			$7 \leq$ Prefix Length ≤ 9		
MPC	21.76	22.66	22.52	33.64	**38.67 (+15.0%)**	**38.40 (+14.1%)**	41.60	**52.21 (+25.5%)**	**52.13 (+25.3%)**
Personal	29.34	30.31	30.10	45.41	**49.52 (+9.1%)**	**49.38 (+8.7%)**	49.81	**56.34 (+13.1%)**	**56.19 (+12.8%)**
Personal-S	31.60	32.59	32.36	50.14	**53.38 (+6.5%)**	**53.30 (+6.3%)**	53.94	**58.69 (+8.8%)**	**58.57 (+8.6%)**
TimeSense	29.98	**31.94 (+6.5%)**	**31.91 (+6.4%)**	47.75	**52.61 (+10.2%)**	**52.65 (+10.3%)**	52.48	**58.63 (+11.7%)**	**58.46 (+11.4%)**
TimeSense-S	30.93	32.69	32.59	49.27	**53.69 (+9.0%)**	**53.67 (+8.9%)**	53.65	**59.19 (+10.3%)**	**59.14 (+10.2%)**
	$10 \leq$ Prefix Length ≤ 12			Prefix Length ≥ 13			All Prefix Lengths		
MPC	47.28	**55.13 (+16.6%)**	**54.82 (+15.9%)**	55.12	**59.28 (+7.5%)**	**58.94 (+6.9%)**	38.19	**44.72 (+17.1%)**	**44.43 (+16.3%)**
Personal	52.16	**57.79 (+10.8%)**	**57.33 (+9.9%)**	56.59	**59.93 (+5.9%)**	59.32	46.67	**51.75 (+10.9%)**	**51.20 (+9.7%)**
Personal-S	55.21	**59.33 (+7.5%)**	**58.77 (+6.4%)**	58.40	60.85	60.16	49.83	**54.30 (+9.0%)**	**53.66 (+7.7%)**
TimeSense	54.91	**59.43 (+8.2%)**	**59.06 (+7.6%)**	58.49	61.08	60.54	48.59	**53.77 (+10.7%)**	**54.01 (+11.2%)**
TimeSense-S	55.73	**59.83 (+7.4%)**	**59.37 (+6.5%)**	58.97	61.29	60.76	49.48	**54.47 (+10.1%)**	**53.95 (+9.0%)**

*Static: Static QAC

moted by adaQAC and "lefont sandy springs showtime" gets promoted.

Query Reformulation: When users prefer new queries when reformulating older queries, adaQAC generally outperforms static QAC. Suppose a user wants to query "detroit lions" after querying "detroit red wings". When the user reformulates the query from "detroit red wings" to "detroit r" by consecutively hitting Backspace, "detroit red wings" is ranked highest but the user does not select it. So, the query "detroit red wings" receives implicit negative feedback. Hence, when the prefix becomes "detroit" after the user hits two more Backspace, "detroit red wings" is demoted by adaQAC; some other queries, such as "detroit lions", are promoted accordingly.

Smoothing "Over-Sense": Certain relevance scores may be sensitive to specific signals: TimeSense is sensitive to time. Studies showed users may have query intent for new or ongoing events [1, 16, 20]. In Yahoo! Search, we investigate the QAC results responded by the time-sensitive component. When a user wants to query an earlier event "russia attack georgia", the time-sensitive QAC keeps ranking a more recent event "russia attack ukraine" highest during keystrokes "russia att". Instead, adaQAC receives users' implicit negative feedback to 'russia attack ukraine" hence demotes it, and raises "russia attack georgia" up to the top.

5. RELATED WORK

Query Auto-Completion (QAC): Numerous QAC models have been developed in recent years, such as popularity-based QAC using historical frequency counts [2], time-based QAC using time information [32, 35], context-based QAC using previous query information of users [2], personalized QAC learning from user profile information [31]. The relevance scores investigated in our work make use of the existing research, such as MPC [2, 15, 21, 31], Personal(-S) [2, 5, 31], and TimeSense(-S) [5, 32, 35, 27]. More recent QAC methods also predicted the probability that a suggested query would be clicked by users based on user models [18, 21], determined suggestion rankings based on query reformulation patterns [15], or combined information such as time and previous queries from users [5]. Furthermore, user interactions with QAC just began to be explored. Mitra *et al.* discussed user-QAC interactions from perspectives such as word boundaries, fraction of query typed, and keyboard distance [26]. Hofmann *et al.* identified common behavior patterns of user-QAC interactions [11].

Other aspects of QAC have also been studied, such as space efficient indexing [13] and spelling error toleration [6, 14, 8, 36].

However, none of the aforementioned work aimed at inferring implicit negative feedback from user-QAC interactions, or adapting QAC to such feedback. We take these initiatives and show that QAC can adapt to implicit negative feedback and be more accurate.

Negative Feedback: Relevance feedback is useful for improving information retrieval models, but further improving it using negative feedback was considered challenging [30, 25]. Recently, more efforts on negative feedback research was made in document retrieval tasks. Wang *et al.* found negative relevance feedback useful to improve vector-space models and language models [34]. Hong *et al.* proposed a hierarchical distance-based measure to differentiate the opposite intent from the true query intent [12]. Zhang and Wang studied language models with negative feedback through positive and negative document proportion on query classification [39]. New models using negative relevance feedback were also developed in TREC [22]. In particular, negative feedback was also found useful to retrieve documents for difficult queries [33, 17, 24].

However, these negative feedback studies focus only on document retrieval tasks. The richer interaction information, presented in the QAC settings, such as dwell time and positions, is not available in general document retrieval settings.

Dynamic IR: Recent work have gone beyond existing IR techniques to incorporate dynamics in session search [9, 23]. In this task, added or removed terms compared with the other queries of the same search session will update term weights to retrieve documents for the completed query [9, 23]. There are important differences between such research and ours. First, search and QAC are different problems. Second, adaQAC emphasizes adapting dynamics over a single query composition rather than multiple queries over a search session. Third, adaQAC does not assign weights to characters, prefixes or terms of a query. Other dynamic IR work was surveyed in a tutorial by Yang *et al.* [37].

6. CONCLUSIONS

We studied interactions between users and QAC where users implicitly express negative feedback to suggested queries. Under the more general "(static) relevance–(adaptive) implicit negative feedback" framework, our proposed adaQAC model can reuse the existing static QAC research and adapt QAC to implicit negative feedback using personalized learning with batch inference. Extensions with un-personalized learning and online inference were also presented. We collected user-QAC interaction data from a commercial search engine. Large-scale empirical results showed that implicit negative feedback significantly and consistently boosts the accuracy of the investigated static QAC models.

Acknowledgment: Research was sponsored in part by National Science Foundation CNS 09-64392, IIS-1017362, IIS-1320617, and IIS-1354329, and by the U.S. Army Research Lab under Cooperative Agreement No. W911NF-09-2-0053 (NSCTA).

7. REFERENCES

[1] E. Adar, D. S. Weld, B. N. Bershad, and S. S. Gribble. Why we search: visualizing and predicting user behavior. In *WWW*, 2007.

[2] Z. Bar-Yossef and N. Kraus. Context-sensitive query auto-completion. In *WWW*, 2011.

[3] C. M. Bishop. *Pattern recognition and machine learning*, volume 1. 2006.

[4] S. Boyd and L. Vandenberghe. *Convex optimization*. 2009.

[5] F. Cai, S. Liang, and M. de Rijke. Time-sensitive personalized query auto-completion. In *CIKM*, 2014.

[6] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *SIGMOD*, 2009.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*, volume 2. 2001.

[8] H. Duan and B.-J. P. Hsu. Online spelling correction for query completion. In *WWW*, 2011.

[9] D. Guan, S. Zhang, and H. Yang. Utilizing query change for session search. In *SIGIR*, 2013.

[10] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*, volume 2. 2009.

[11] K. Hofmann, B. Mitra, F. Radlinski, and M. Shokouhi. An eye-tracking study of user interactions with query auto completion. In *CIKM*, 2014.

[12] Y. Hong, Q. Cai, S. Hua, J. Yao, and Q. Zhu. Negative feedback: the forsaken nature available for re-ranking. In *COLING*, 2010.

[13] B.-J. P. Hsu and G. Ottaviano. Space-efficient data structures for top-k completion. In *WWW*, 2013.

[14] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, 2009.

[15] J.-Y. Jiang, Y.-Y. Ke, P.-Y. Chien, and P.-J. Cheng. Learning user reformulation behavior for query auto-completion. In *SIGIR*, 2014.

[16] S. R. Kairam, M. R. Morris, J. Teevan, D. J. Liebling, and S. T. Dumais. Towards supporting search over trending events with social media. In *ICWSM*, 2013.

[17] M. Karimzadehgan and C. Zhai. Improving retrieval accuracy of difficult queries through generalizing negative document language models. In *CIKM*, 2011.

[18] E. Kharitonov, C. Macdonald, P. Serdyukov, and I. Ounis. User model-based metrics for offline query suggestion evaluation. In *SIGIR*, 2013.

[19] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[20] A. Kulkarni, J. Teevan, K. M. Svore, and S. T. Dumais. Understanding temporal query dynamics. In *WSDM*, 2011.

[21] Y. Li, A. Dong, H. Wang, H. Deng, Y. Chang, and C. Zhai. A two-dimensional click model for query auto-completion. In *SIGIR*, 2014.

[22] Y. Li, X. Tao, A. Algarni, and S.-T. Wu. Mining specific and general features in both positive and negative relevance feedback. In *TREC*, 2009.

[23] J. Luo, S. Zhang, and H. Yang. Win-win search: Dual-agent stochastic game in session search. In *SIGIR*, 2014.

[24] Y. Ma and H. Lin. A multiple relevance feedback strategy with positive and negative models. *PloS ONE*, 9(8), 2014.

[25] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. 2008.

[26] B. Mitra, M. Shokouhi, F. Radlinski, and K. Hofmann. On user interactions with query auto-completion. In *SIGIR*, 2014.

[27] T. Miyanishi and T. Sakai. Time-aware structured query suggestion. In *SIGIR*, 2013.

[28] K. P. Murphy. *Machine learning: a probabilistic perspective*. 2012.

[29] A. Nedić. *Optimization*. Technical Report, UIUC, 2011.

[30] J. J. Rocchio. Relevance feedback in information retrieval. *The SMART Retrieval System Experiments in Automatic Document Processing*, 1971.

[31] M. Shokouhi. Learning to personalize query auto-completion. In *SIGIR*, 2013.

[32] M. Shokouhi and K. Radinsky. Time-sensitive query auto-completion. In *SIGIR*, 2012.

[33] X. Wang, H. Fang, and C. Zhai. Improve retrieval accuracy for difficult queries using negative feedback. In *CIKM*, 2007.

[34] X. Wang, H. Fang, and C. Zhai. A study of methods for negative relevance feedback. In *SIGIR*, 2008.

[35] S. Whiting and J. M. Jose. Recent and robust query auto-completion. In *WWW*, 2014.

[36] C. Xiao, J. Qin, W. Wang, Y. Ishikawa, K. Tsuda, and K. Sadakane. Efficient error-tolerant query autocompletion. *VLDB*, 6(6), 2013.

[37] H. Yang, M. Sloan, and J. Wang. Dynamic information retrieval modeling. In *SIGIR*, 2014.

[38] S.-H. Yang, B. Long, A. J. Smola, H. Zha, and Z. Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *SIGIR*, 2011.

[39] W. Zhang and J. Wang. The study of methods for language model based positive and negative relevance feedback in information retrieval. In *ISISE*, 2012.

APPENDIX

A. INFERENCE FOR ADAQAC-BATCH

Let $f\left[\boldsymbol{\phi}^{(t)}(u)\right]$ be the objective function in Equation 6, where $\boldsymbol{\phi}^{(t)}(u)$ is the value of $\boldsymbol{\phi}(u)$ at the t-th iteration,

$$\boldsymbol{\phi}^{(t+1)}(u) = \boldsymbol{\phi}^{(t)}(u) - \eta \nabla f\left[\boldsymbol{\phi}^{(t)}(u)\right], \tag{7}$$

where

$$\nabla f\left[\boldsymbol{\phi}(u)\right] = \left[\frac{\partial f\left[\boldsymbol{\phi}(u)\right]}{\partial \boldsymbol{\phi}_1(u)}, \frac{\partial f\left[\boldsymbol{\phi}(u)\right]}{\partial \boldsymbol{\phi}_2(u)}, \ldots, \frac{\partial f\left[\boldsymbol{\phi}(u)\right]}{\partial \boldsymbol{\phi}_l(u)}\right]^\top, \tag{8}$$

and $\forall i = 1, 2, \ldots, l$,

$$\frac{\partial f\left[\boldsymbol{\phi}(u)\right]}{\partial \boldsymbol{\phi}_i(u)} = \sum_{c \in \mathbf{C}(u)} \frac{S_1}{S_2} - \mathbf{x}_i^{\left(K(c)\right)}\left(u, q^*(c), c\right) + \lambda \boldsymbol{\phi}_i(u), \tag{9}$$

where by denoting $\exp\left[r^{(k)}(u,q,c) + \boldsymbol{\phi}^\top(u)\mathbf{x}^{\left(K(c)\right)}(u,q,c)\right]$ as $E(q)$,

$$\begin{aligned} S_1 &= \sum_{q \in \mathbf{Q}^{(k)}(r,u,c,N) \bigcup \{q^*(c)\}} E(q)\mathbf{x}_i^{\left(K(c)\right)}(u,q,c), \\ S_2 &= \sum_{q \in \mathbf{Q}^{(k)}(r,u,c,N) \bigcup \{q^*(c)\}} E(q). \end{aligned} \tag{10}$$

In the experiments, $\boldsymbol{\phi}^{(0)}(u)$ in Equation 7 is randomly sampled from:

$$\boldsymbol{\phi}^{(0)}(u) \sim \text{Uniform}(0, 0.01).$$

B. INFERENCE FOR ADAQAC-ONLINE

The feature weight $\boldsymbol{\phi}^{(0)}(u)$ is initialized as the un-personalized learning weight (§3.5.1). After each query composition c, the feature weight is updated as in Equation 7—10 with Equation 9 replaced by

$$\frac{\partial f\left[\boldsymbol{\phi}(u)\right]}{\partial \boldsymbol{\phi}_i(u)} = \frac{S_1}{S_2} - \mathbf{x}_i^{\left(K(c)\right)}\left(u, q^*(c), c\right) + \lambda \boldsymbol{\phi}_i(u),$$

and η is discounted by a factor of 0.9 after each update as an annealing procedure [19].