adaQAC: Adaptive Query Auto-Completion via Implicit Negative Feedback

Aston Zhang¹, Amit Goyal², Weize Kong³, Hongbo Deng²
Anlei Dong², Yi Chang², Carl A. Gunter¹, Jiawei Han¹

¹University of Illinois at Urbana-Champaign, USA
²Yahoo! Labs, USA
³University of Massachusetts Amherst, USA

PRESENTED BY Aston Zhang | Aug 10, 2015 | Santiago, Chile, SIGIR
Query auto-completion (QAC) is important

- People rely on query auto-completion to reduce efforts in formulating queries.
- In 2014, global users of Yahoo! Search saved more than 50% keystrokes when submitting English queries by selecting suggestions of QAC.
Existing QAC models make use of various information:

- Context-sensitive QAC (WWW’11)
- Time-sensitive QAC (SIGIR’12)
- Personalize QAC (SIGIR’13)
- Recent & robust QAC (WWW’14)
- Two-dimensional QAC (SIGIR’14)
- Query-reformulation QAC (SIGIR’14)
- …

They all index queries for prefixes **statically**
We propose Adaptive QAC (adaQAC)

- Maintain rank of queries in previous suggestion list where user skipped

- Demote rank of queries in previous suggestion list where user examined but did not click (Implicit Negative Feedback)
Maintain rank of queries in previous suggestion list where user skipped

- **Example:**
  - User wants to query “facebook”
  - User types “fac” and quickly skips it, although “facebook” shows in the suggestion list
  - In the next keystroke, when user formulates his prefix as “face”
    - For prefix “face”, rank “facebook” as top 1 in the suggestion list since it is among the most popular queries
    - Static QAC fits this case
Demote rank of queries in previous suggestion list where user examined but did not click

- Example:
  - User wants to query “facetime”
  - User types “fac” and examines the suggestion list, where “facebook” ranks top 1
  - In the next keystroke, when user formulates his prefix as “face”
    - Static QAC for prefix “face” may still rank “facebook” as top 1 since it is among the most popular queries
    - Adaptive QAC: Can we demote the ranking of “facebook” if we know user does not want to query “facebook” given the implicit negative feedback from previous interactions?
Negative feedback can be expressed via longer dwell time for query suggestions at higher positions.

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.
System overview of adaQAC

User

Front-End Interface

Prefix: *face*

*facetime*

Facebook

Back-End System

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.
$\text{adaQAC}(q,c) = f \left[ \text{relevance}(q), \text{implicit negative feedback} \ (q, \ c) \right]$
Probabilistic model: softmax[adaQAC(q)]

- **Symbols**
  - O(n) = Top n queries ranked by r(q): relevance score
    - Re-rank these n queries given nf(q,s)
  - q*: the final submitted query
- **Training (max likelihood with regularization):**
  \[
p(q = q^*) = \frac{\exp[r(q^*) + \phi^T x(q,s)]}{\sum_{q_i \in O(n) \cup \{q^*\}} \exp[r(q_i) + \phi^T x(q,s)]}
\]
- **Testing:**
  \[
p(q = q^*) = \frac{\exp[r(q^*) + \phi^T x(q,s)]}{\sum_{q_i \in O(n)} \exp[r(q_i) + \phi^T x(q,s)]}
\]
## Detailed formalization & inference: symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U )</td>
<td>User set.</td>
</tr>
<tr>
<td>( u )</td>
<td>User.</td>
</tr>
<tr>
<td>( C(u) )</td>
<td>Query composition set of a user ( u ).</td>
</tr>
<tr>
<td>( c )</td>
<td>Query composition.</td>
</tr>
<tr>
<td>( K(c) )</td>
<td>Number of keystrokes in a query composition ( c ).</td>
</tr>
<tr>
<td>( k )</td>
<td>Keystroke index: ( k \in { 1, 2, \ldots, K(c) } ).</td>
</tr>
<tr>
<td>( Q )</td>
<td>Query set.</td>
</tr>
<tr>
<td>( q, q' )</td>
<td>Query.</td>
</tr>
<tr>
<td>( q^*(c) )</td>
<td>Submitted query in a query composition ( c ).</td>
</tr>
<tr>
<td>( r^{(k)}(u, q, c) )</td>
<td>Relevance score for a user ( u ) of a query ( q ) that matches the prefix at a keystroke ( k ) in a query composition ( c ).</td>
</tr>
<tr>
<td>( Q^{(k)}(r, u, c, N) )</td>
<td>Set of top ( N ) queries ranked by ( r^{(k)}(u, q, c) ).</td>
</tr>
<tr>
<td>( x^{(k)}_{1 \times 1}(u, q, c) )</td>
<td>Implicit negative feedback feature vector from a user ( u ) to a query ( q ) at a keystroke ( k ) in a query composition ( c ).</td>
</tr>
<tr>
<td>( \Phi_{1 \times m}(U) )</td>
<td>Implicit negative feedback feature weight matrix for a user set ( U ).</td>
</tr>
<tr>
<td>( \phi_{1 \times 1}(u) )</td>
<td>Implicit negative feedback feature weight vector for a user ( u ).</td>
</tr>
<tr>
<td>( p^{(k)}(u, q, c) )</td>
<td>Preference for a query ( q ) of a user ( u ) at a keystroke ( k ) in a query composition ( c ).</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>Regularizer weight parameter.</td>
</tr>
</tbody>
</table>
We use gradient descent for batch inference (adaQAC-batch)

Objectives function:

\[
\begin{align*}
\text{minimize} & \quad \sum_{c \in C(u)} \log \sum_{q \in Q^{(k)}(r, u, c, N) \cup \{q^*(c)\}} \exp \left[ p(K(c))(u, q, c) \right] \\
& - p(K(c))(u, q^*(c), c) \\
\text{subject to} & \quad \|\phi(u)\|_2^2 \leq \nu, \; \nu \in \mathbb{R}^+.
\end{align*}
\]

Inference:

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

where

\[
\nabla f \left[ \phi(u) \right] = \left[ \frac{\partial f \left[ \phi(u) \right]}{\partial \phi_1(u)}, \frac{\partial f \left[ \phi(u) \right]}{\partial \phi_2(u)}, \ldots, \frac{\partial f \left[ \phi(u) \right]}{\partial \phi_l(u)} \right]^T,
\]

and \( \forall i = 1, 2, \ldots, l, \)

\[
\frac{\partial f \left[ \phi(u) \right]}{\partial \phi_i(u)} = \sum_{c \in C(u)} \frac{S_1}{S_2} - x_i^{(K(c))}(u, q^*(c), c) + \lambda \phi_i(u),
\]

where by denoting \( \exp \left[ r^{(k)}(u, q, c) + \phi^T(u) x^{(K(c))}(u, q, c) \right] \)

as \( E(q), \)

\[
S_1 = \sum_{q \in Q^{(k)}(r, u, c, N) \cup \{q^*(c)\}} E(q) x_i^{(K(c))}(u, q, c),
\]

\[
S_2 = \sum_{q \in Q^{(k)}(r, u, c, N) \cup \{q^*(c)\}} E(q).
\]
We use stochastic gradient descent for online inference (adaQAC-online)

**Objective function:**

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

**Inference:**

\[
\phi^{(t+1)}(u) = \phi^{(t)}(u) - \eta \nabla f[\phi^{(t)}(u)],
\]

where

\[
\nabla f[\phi(u)] = \left[ \frac{\partial f[\phi(u)]}{\partial \phi_1(u)}, \frac{\partial f[\phi(u)]}{\partial \phi_2(u)}, \ldots, \frac{\partial f[\phi(u)]}{\partial \phi_l(u)} \right]^T,
\]

and \( \forall i = 1, 2, \ldots, l, \)

\[
\frac{\partial f[\phi(u)]}{\partial \phi_i(u)} = \frac{S_1}{S_2} - x_i^{(K(c))}(u,q^*(c),c) + \lambda \phi_i(u),
\]

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

\[
\begin{align*}
S_1 &= \sum_{q \in Q^{(k)}(r,u,c,N) \cup \{q^*(c)\}} E(q)x_i^{(K(c))}(u,q,c), \\
S_2 &= \sum_{q \in Q^{(k)}(r,u,c,N) \cup \{q^*(c)\}} E(q).
\end{align*}
\]
We perform personalized learning because different users have different typing speed and it is scalable on MapReduce.

(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.
We perform evaluation on a larger data set

- Total query compositions: 2,932,035 (SIGIR'14 paper: 125,392)
- Unique submitted queries: 481,417
- Average length of query prefixes: 8.53 characters
- Time range: 2014.2.28-2014.7.28
- All queries are submitted via PC
Both adaQAC-Batch and adaQAC-Online significantly and consistently boost the accuracy of static QAC under all prefix lengths for each relevance score.

- All in percentage
- Testing on all the keystrokes of a prefix
- MRR: Mean Reciprocal Rank
- SR@i: Success Rate at top i suggestions
- 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
Parameter Study: adaQAC (TimeSense-S) & testing the last keystroke of a prefix
Further research on more accurate relevance score is still required.
Case 1 (Disambiguation): When users have clear query intent and prefer disambiguated queries, adaQAC generally outperforms static QAC.

- User wants to query: lefont sandy springs showtime
- l – le – lef – lefo – lefon (lefont sandy springs is ranked top)
- At prefix “lefont”
  - lefont sandy springs showtime > lefont sandy springs
Case 2 (Query Reformulation): When users prefer new queries when reformulating older queries, adaQAC generally outperforms static QAC.

- User wants to query: **detroit lions** after querying **detroit red wings**
- **detroit red wings** – … – **detroit r** (**detroit red wings** is ranked top)
- At prefix “detroit”
  - detroit lions > detroit red wings
Case 3: Smoothing “over-sense”

- Certain relevance scores may be sensitive to specific signals: e.g., TimeSense is sensitive to time.
- User wants to query an earlier event: **russia attack georgia**
- r – … – russia att (a more recent event: **russia attack ukraine** is ranked top)
- With adaQAC,
  - russia attack georgia > russia attack ukraine
Thank you!