Query Auto-Completion using an Abstract Language Model

BACHELOR THESIS BY NATALIE PRANGE, 02.11.2016
Introduction to query auto-completion

• Query auto-completion (QAC): suggesting completions for a query prefix entered by a user

• Objective:
  • Reduce the user’s effort to enter a query
  • Prevent spelling mistakes
  • Assist in formulating a query

→ A QAC-algorithm must suggest the desired query after a minimal amount of keystrokes at a high rank
A common solution

• Suggest the most popular queries from a query log that match the given prefix

• Problems with this approach:
  • Recent and large enough query logs are hard to get
  • Queries which are asked for the first time are not suggested
A language-model-based solution

- Focus in this work is on whole questions
  → possible solution: use a language model
- Language model = probability distribution learned over sequences of words
- Can be used to predict the word most likely to follow a given sequence
- Typical problem: data sparsity
This approach

• Use an abstract language model: specific entities are replaced by abstract types
  • E.g.: "Who played Gandalf in The Lord of the Rings?"
    → "Who played [fictional character] in [film]?"

• When the language model predicts a type, entities are inserted again

• A prominence score and word vector similarity are used to rank suggestions
Basic pipeline of the Auto-Completion algorithm.
Building the abstract language model

• Choosing a type for each entity:
  • Out of a list of types of an entity, choose the most general but still meaningful type
  • E.g.: Albert Einstein: [Person, Astronomer, Diet Follower, Topic, ...] \(\rightarrow\) Person
  • Choose a type according to a hand-picked list of preferred types
Building the abstract language model

• The training set consists of questions in which recognized entities are replaced by their type

• An n-gram language model is learned on these questions

• N-gram model:
  • Estimate the probability of a word given it’s (n-1) predecessors:
  • $P(w_m|w_1, ..., w_{m-1}) \approx P(w_m|w_{m-(n-1)}, ..., w_{m-1}) = \frac{\text{count}(w_{m-(n-1)}, ..., w_{m-1}, w_m)}{\text{count}(w_{m-(n-1)}, ..., w_{m-1})}$
Building the Word2vec model

- Word2vec uses a neural network to learn vector representations of words.
- The more common context two words share, the higher the cosine similarity of their word vectors.
  → can be used to compute semantic similarity between words.
- E.g.: \( \text{vector(Berlin)} - \text{vector(Germany)} + \text{vector(France)} \approx \text{vector(Paris)} \)
Predicting possible next words

- Normally:
  - last (n-1) complete words = n-gram context
  - last incomplete word = prefix of the next word

- Here: a predicted type can correspond to multiple words typed by the user
  - E.g.: „Who played [Fictional Character|Iron Man] in the first A“
  - Which words are part of an entity name and which are normal words?

- Get predictions for all possible prefixes and their corresponding n-gram context
Inserting entities for types

- Insert entities for every type predicted by the n-gram model
- Entities need to match the given type and match the given prefix
- Prefix trees are used for retrieval of entities
Prefix tree, built from the words [to, too, tool, tour, see, fee]
Rating and ranking

- 1st scenario: the question prefix does not contain any entity
  - Use a prominence score to rate entities
  - Normalize score
  - \[ s_{final} = p_{n-gram} \times (s_{norm})^{0.3} \]

- 2nd scenario: the question prefix contains at least one entity
  - Compute word vector similarity between the contained, and the suggested entity
  - Fill in the word vector similarity for \( s_{norm} \)

- Normal words are assigned a fixed score in both approaches
Filling up the completion suggestions

- Use words that were not predicted by the n-gram model
- Use the prominence score and word count for rating the fill-up words
- Always append completely typed entities to the completion suggestions
Evaluation: Metrics

• **User Interaction:**
  
  \[
  \frac{(\text{total keystrokes} + \text{total selections})}{(\text{total number of characters in question})}
  \]

• **Mean Reciprocal Rank (MRR):**
  
  \[
  RR(q_c, S) = \frac{1}{rank(q_c, S)}
  \]
  
  • RR is computed after typing the first letter of a word
  • MRR is the mean of the RR’s of every word / entity name in every question

• **Percentage of unidentified entities**
Evaluation: Tested algorithm versions

- Baseline:
  - Without filling up completion suggestions
  - Without appending complete words
- 2nd Version: Without appending complete words
- 3rd Version: Only prominence score for rating (no word vectors)
- 4th Version: Complete algorithm as described
# Evaluation: Results

<table>
<thead>
<tr>
<th>Algorithm Version</th>
<th>MRR</th>
<th>User Interaction</th>
<th>Unid. Entities</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.376</td>
<td>0.64</td>
<td>38.9%</td>
<td>0.027 secs</td>
</tr>
<tr>
<td>w/o complete entities</td>
<td>0.469</td>
<td>0.49</td>
<td>11.1%</td>
<td>0.047 secs</td>
</tr>
<tr>
<td>w/o Word2vec model</td>
<td>0.449</td>
<td>0.49</td>
<td>6.3%</td>
<td>0.040 secs</td>
</tr>
<tr>
<td>Complete algorithm</td>
<td>0.457</td>
<td>0.49</td>
<td>6.3%</td>
<td>0.047 secs</td>
</tr>
</tbody>
</table>
## Evaluation: Results

### Questions containing one entity

<table>
<thead>
<tr>
<th>Algorithm Version</th>
<th>MRR</th>
<th>User Interaction</th>
<th>Unid. Entities</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.373</td>
<td>0.64</td>
<td>33.2%</td>
<td>0.028 secs</td>
</tr>
<tr>
<td>No complete entities</td>
<td>0.469</td>
<td>0.49</td>
<td>10.2%</td>
<td>0.048 secs</td>
</tr>
<tr>
<td>w/o Word2vec model</td>
<td>0.449</td>
<td>0.50</td>
<td>6.2%</td>
<td>0.041 secs</td>
</tr>
<tr>
<td>Complete algorithm</td>
<td>0.457</td>
<td>0.50</td>
<td>6.2%</td>
<td>0.047 secs</td>
</tr>
</tbody>
</table>

### Questions containing two or more entities

<table>
<thead>
<tr>
<th>Algorithm Version</th>
<th>MRR</th>
<th>User Interaction</th>
<th>Unid. Entities</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.385</td>
<td>0.66</td>
<td>50.4%</td>
<td>0.025 secs</td>
</tr>
<tr>
<td>No complete entities</td>
<td>0.465</td>
<td>0.49</td>
<td>15.7%</td>
<td>0.046 secs</td>
</tr>
<tr>
<td>w/o Word2vec model</td>
<td>0.444</td>
<td>0.47</td>
<td>6.7%</td>
<td>0.037 secs</td>
</tr>
<tr>
<td>Complete algorithm</td>
<td>0.452</td>
<td>0.48</td>
<td>6.8%</td>
<td>0.046 secs</td>
</tr>
</tbody>
</table>
Completion suggestions using the Word2vec model:

<table>
<thead>
<tr>
<th>Completion suggestion</th>
</tr>
</thead>
<tbody>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
</tbody>
</table>

Completion suggestions using only an entity prominence score:

<table>
<thead>
<tr>
<th>Completion suggestion</th>
</tr>
</thead>
<tbody>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
<tr>
<td>who played [fictional_character</td>
</tr>
</tbody>
</table>
Future work

- Integrate proper entity recognition
  - E.g.: USA → United States of America
- Robustness against spelling mistakes
- Multiple-word suggestions