Instant Search - A Hands-on Tutorial

Ganesh Venkataraman, Abhimanyu Lad, Viet Ha-Thuc, Dhruv Arya
LinkedIn Corporation
Mountain View, CA, USA
{ghvenkat,alad,vhathuc,darya}@linkedin.com

MOTIVATION

Instant search has become a common part of the search experience in most popular search engines and social networking websites. The goal is to provide instant feedback to the user in terms of query completions ("instant suggestions") or directly provide search results ("instant results") as the user is typing their query. The need for instant search has been further amplified by the proliferation of mobile devices and services like *Siri* and *Google Now* that aim to address the user's information need as quickly as possible. Examples of instant results include web queries like *"weather san jose"* (which directly provides the current temperature), social network queries like searching for someone's name on Facebook or LinkedIn (which directly provide the people matching the query). In each of these cases, instant search constitutes a superior user experience, as opposed to making the user complete their query before the system returns a list of results on the traditional search engine results page (SERP).

We consider instant search experience to be a combination of instant results and instant suggestions, with the goal of satisfying the user's information need as quickly as possible with minimal effort on the part of the user. We first present the challenges involved in putting together an instant search solution at scale, followed by a survey of IR and NLP techniques that can be used to address them. We will also conduct a hands-on session aimed at putting together an end-to-end instant search system using open source tools and publicly available data sets. These tools include typeahead.js from Twitter for the frontend and Lucene/elasticsearch for the backend. We present techniques for prefix-based retrieval as well as injecting custom ranking functions into elasticsearch. For the search index, we will use the dataset made available by Stackoverflow [2].

This tutorial is aimed at both researchers interested in knowing about retrieval techniques used for instant search as well as practitioners interested in deploying an instant search system at scale. The authors have worked extensively on building and scaling LinkedIn's instant search experience. To the best of our knowledge, this is the first tutorial that covers both theoretical and practical aspects of instant search.

SIGIR '16, July 17 - 21, 2016, Pisa, Italy

ISBN 978-1-4503-4069-4/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2911451.2914806

Keywords

Information Retrieval; Instant Search; Query Understanding

1. OBJECTIVES

Over the years search has become more focused on *getting results as you type*. Instant search has become a common part of the user experience in nearly all popular search engines, including web search engines (like Google), verticalized search engines (like Kayak) as well as social search engines (like Facebook, LinkedIn). The two broad categories of instant search include:

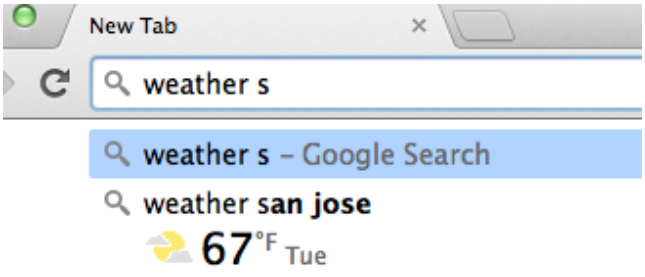
1. **Instant Results** (Figure 1). Here the actual result is provided while the query is being typed out.
2. **Instant Suggestions** (Figure 2). Here the user is guided to complete the query and the results will be presented after the suggestion is clicked.

For the rest of this proposal, **instant search** refers to both the above aspects unless otherwise stated. From an information need perspective, it is useful to think about queries as belonging to two categories:

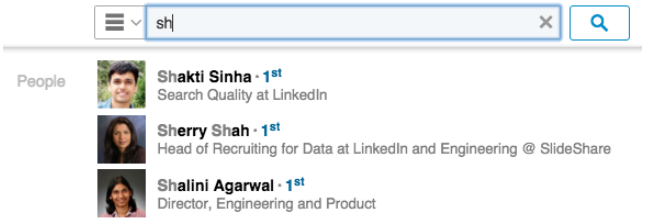
1. **Navigational search**. Here the user has a single entity in mind and there exists a single document which would satisfy their information need. Classic example of navigational query in social networks is name search.
2. **Exploratory search**. Here multiple results can satisfy the information need. Example: "software engineer new york" could have several good results.

The general approach for navigational search is to provide the result right away – most often in a typeahead dropdown box. As illustrated in Figure 1, this means providing the current temperature for San Jose, or presenting the people search results for the query *"sh"*. For exploratory queries where the need can be met by multiple documents, the goal is not to provide results in the typeahead box, but instead guide the user in framing their query before they end up on the search engine results page (SERP).

For a great instant search experience, we will need the following aspects to work in tandem:



(a) Example from Google search



(b) Example from LinkedIn

Figure 1: Instant result experience

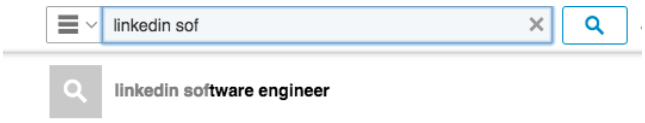


Figure 2: Instant suggestions

1. Frontend (client side) and decoration of search results
2. Offline data analysis
3. Offline indexing
4. Online retrieval within strict latency limits
5. Online Ranking within strict latency limits
6. Spell correction
7. Query suggestions

This tutorial is aimed at going over the theoretical and practical aspects of all the above issues. Our tutorial would be useful for both academics interested in expanding their knowledge as well as practitioners keen on putting together a working solution. We will make all our code available on GitHub and leverage the following tools - typeahead.js [4] from Twitter, Elasticsearch [1]/Lucene [14] for backend. We will use the Stackoverflow user-generated data [2] to construct the index and build an instant search experience on this dataset.

2. RELEVANCE TO INFORMATION RETRIEVAL AND RELATED WORK

Instant search is increasingly becoming an important part of the search experience. While there has been decades of research effort spent on optimizing the traditional "SERP" experience, building a scalable instant search experience poses its own unique set of challenges in terms of indexing, retrieval, and ranking. We believe the information retrieval researchers and practitioners would greatly benefit from an overview of these challenges, potential solutions and alternatives, as well as the various trade offs involved.

Our goal in this tutorial is to provide an overview of the challenges involved, the various pieces of technology and relevance components that need to work together, and then walk the audience through the process of building a scalable system end-to-end. **To the best of our knowledge, such a tutorial has not been presented before in the information retrieval community.** That said, there's prior research that attempts to address various challenges posed by instant search – although in a piecemeal manner.

2.1 Related Work

In the area of indexing and retrieval, Bast et al. [9] proposed a block-based index to improve retrieval speed by reducing random accesses to posting lists. But this does not address the problem of efficiently finding candidate completions of the partially typed query. Li et al. [16] describe a trie based approach to represent prefixes, with posting lists at the leaf nodes. While such an approach works well for query autocomplete systems with few million entries, it's much harder to scale further without additional retrieval techniques like early termination [7, 21, 22].

Early termination is a retrieval technique that avoids exhaustively scoring all matching documents; this is achieved by reorganizing the index such that documents with high prior probability of relevance (generally referred to as "static rank") appear first. Commonly used scores include within-document term frequencies [18], or some notion of document quality or popularity [10, 17]. Needless to say, the effectiveness of early termination hinges on the choice of "static rank", which in turn is highly dependent on the problem domain.

Fuzzy search is an area of research that aims to build retrieval systems that are tolerant to minor spelling errors or variations. While full word spelling correction is a well-studied problem [15, 13, 6], fixing spelling errors in partial queries as the user is typing is a relatively new area of research [11, 12].

In the area of personalized query suggestions, Bar-Yossef et al. [8] propose an approach that biases query completions towards previous queries in the same session. But this does not address single-query sessions, which are very common in navigational search use cases. Weber et al. [20] focused on how query likelihoods differ by demographics. Milad [19] provides a machine learning framework that can combine demographic and user features for generating personalized query completions.

In terms of open source technologies, there are several JavaScript libraries for implementing client side autocomplete, e.g. typeahead.js [4] and Bootstrap [3]. For the search backend, Apache Lucene [14] is a search engine library with support for full text search via a fairly expressive query language, extensible scoring, and high performance indexing. Elastic Search [1] is a search server based on Lucene that provides the ability to quickly build scalable search engines. It provides a distributed, multitenant-capable search engine with a HTTP web interface.

In this tutorial, we aim to bring together the best ideas from prior research and leverage open source technologies to put together a working end to end instant search experience over a publicly available dataset.

3. FORMAT AND DETAILED SCHEDULE

All code and scripts needed will be made available on GitHub

1. Quick hitter – End-to-end instant search experience using Python and typeahead.js (20 min)
   - This will serve as an ice-breaker.
   - We will show how easy it is to put together a basic autocomplete service within 40 lines of python.
   - The time will also be used to make sure the developer environment is set up.
2. Literature and tools survey. Cover various aspects of instant search referenced in Section 2.1. These include:
   - Performance and Retrieval (30 min).
     - Using indexing techniques like early termination and static rank to retrieve and rank fewer documents while maintaining recall.
     - Using FST or Trie based retrieval. Theory covering aspects of Lucene FST [5].
     - Prefix indexing based retrieval.
     - Trade offs between prefix indexing and FST.
   - Tolerance to Errors. We will cover various aspects of fixing common spelling errors in autocomplete (20 min).
   - Relevance (20 min).
     - Ranking autocomplete suggestions using session data.
     - Personalizing suggestions.
     - Using custom scoring function inside Lucene.
3. Walking through a real example (90 min)
   (a) Overview and an End-to-End Prototype (15 min)
   - Understanding and familiarizing ourselves with the dataset
     - Essential fields
     - Tokenization needs
   - Understanding the tools (elasticsearch [1], python etc.)
   - Given a particular open data set and a schema, create an search index using elasticsearch [1]. Sample curl queries would be provided to test if the index is working (specific queries with expected results)

   (b) Designing Prefix Index (30 min)
   - Retrieval and Ranking. Given an open data set.
     - Decide which fields to index and create an index which can handle prefixes
     - Send sample prefix query and get results
   - Decide fields to index for retrieval (Analyzers, N-Grams etc.)
   - Decide fields to be used for ranking

   (c) Building front end (15 min)
   - Integrate front end components to build end-to-end experience
   - Use typeahead.js [4] to query the backend built using tools built in previous sections

   (d) Ranking and Query Rewriting (30 min)
   This will be the creative section of the tutorial where attendees have the opportunity to extent what had been built in the last one hour towards their own custom search experience. Examples would be provided. Sample problems handled:
   - Blending results between different query types like mixing query suggestions and results for the same prefix. Example: query like "ha" can have a mix of results like "Tag: hadoop" (suggesting to search via tag "hadoop") as well as instant results like the question "How to learn Hadoop".
   - Integrating metadata into ranking (like number of upvotes for a Q&A site)

4. PRESENTER INFORMATION

Ganesh Venkataraman currently leads jobs relevance at LinkedIn. His contributions at LinkedIn include, leading end to end re-architecture of job search, machine learned ranking for people search typeahead, introducing machine learned ranking towards skills search at LinkedIn. He co-authored a paper on personalized ranking which won the best paper award at the IEEE Big Data Conference 2015. He holds a Ph.D. from Texas A&M in Electrical & Computer Engineering where he was the recipient of the Dean's graduate merit scholarship.

Abhimanyu Lad Abhimanyu (Abhi) Lad leads query understanding efforts at LinkedIn. His contributions include major improvements to query intent prediction, query suggestions, spelling correction, name search and leading a major re-architecture of LinkedIn people search stack. He holds a Ph.D. in Language and Information Technologies from Carnegie Mellon University, where he was the recipient of the Yahoo! Ph.D. Fellowship. He has several publications in the field of information retrieval and machine learning with over 130 citations.

Viet Ha-Thuc leads machine learning efforts for improving search quality at LinkedIn. He has played a key role in designing and implementing machine learned ranking for personalized search and federation across several verticals at LinkedIn. His work on LinkedIn search has been published at conferences such as CIKM, Big Data and WWW. One of the publications received the Best Application Paper Award at 2015 IEEE Big Data. Prior to LinkedIn, he was a scientist in the Content Understanding group at Yahoo! Labs, where he developed a machine learning system for extracting relevant entities and concepts in text documents. The system was deployed to annotate every email and news article in the Yahoo! ecosystem. He received a Ph.D. in Computer Science from the University of Iowa in 2011.

Dhruv Arya currently leads job search quality at LinkedIn. His goal is to apply machine learning and data mining approaches to build talent matching algorithms that connect job seekers to the most relevant jobs. Apart from this, he has made key contributions to query understanding and rewriting, whole page optimization, as well as personalized federated search, which was presented at CIKM 2015. He received a Master's degree in Computer Science from University of Pennsylvania in 2013.

5. REFERENCES

[1] Elasticsearch documentation, https://www.elastic.co/guide/index.html.

[2] Stackoverflow creative commons data dump, http://blog.stackoverflow.com/2009/06/stack-overflow-creative-commons-data-dump/.

[3] Twitter bootstrap library. http://getbootstrap.com.

[4] typeahead.js library. https://twitter.github.io/typeahead.js.

[5] Using finite state transducers in lucene, http://blog.mikemccandless.com/2010/12/using-finite-state-transducers-in.html.

[6] F. Ahmad and G. Kondrak. Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 955–962. Association for Computational Linguistics, 2005.

[7] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 35–42. ACM, 2001.

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

[9] H. Bast and I. 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. ACM, 2006.

[10] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine, 1998. In *Proceedings of the Seventh World Wide Web Conference*, 2007.

[11] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 707–718. ACM, 2009.

[12] H. Duan and B.-J. P. Hsu. Online spelling correction for query completion. In *Proceedings of the 20th international conference on World wide web*, pages 117–126. ACM, 2011.

[13] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun. A large scale ranker-based system for search query spelling correction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 358–366. Association for Computational Linguistics, 2010.

[14] E. Hatcher and O. Gospodnetic. Lucene in action. 2004.

[15] M. D. Kernighan, K. W. Church, and W. A. Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the 13th conference on Computational linguistics-Volume 2*, pages 205–210. Association for Computational Linguistics, 1990.

[16] G. Li, J. Wang, C. Li, and J. Feng. Supporting efficient top-k queries in type-ahead search. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 355–364. ACM, 2012.

[17] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 129–140. VLDB Endowment, 2003.

[18] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *JASIS*, 47(10):749–764, 1996.

[19] M. Shokouhi and K. 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. ACM, 2012.

[20] I. Weber and C. Castillo. The demographics of web search. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 523–530. ACM, 2010.

[21] H. Yan, S. Shi, F. Zhang, T. Suel, and J.-R. Wen. Efficient term proximity search with term-pair indexes. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1229–1238. ACM, 2010.

[22] F. Zhang, S. Shi, H. Yan, and J.-R. Wen. Revisiting globally sorted indexes for efficient document retrieval. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 371–380. ACM, 2010.