Foundations of Session-Based Collaborative Search

Sebastian Straub

sebastian.straub@mailbox.tu-dresden.de

2016-12-16

Session-based search is the concept of harnessing the query history of a single search session to improve precision and recall of subsequent queries by making assumptions about the context of the current query. Collaborative search describes the participation of multiple users in the same search session in real-time, where results can be shared and the actions of one user may influence the ranking of the search results of another participant. The combination of these properties leads to a search engine that should be well suited for complex exploratory search tasks that can be solved efficiently by a small group of people in a combined effort.

This work lays the foundation for an upcoming master's thesis in which the concept of a session-based collaborative search engine will be investigated. A working prototype has been built as part of this work that implements session-based search on a collection of scientific papers.

Contents

1 Introduction

In the early years of the internet, search engines did not distinguish between their users: the same query generated the same search results for everyone, independent of any contextual information. This approach turned out to be somewhat limited in providing relevant results, as relevance often depends on contextual information about the user, like location, language or preferences, and so the idea of a personalized search engine [3] became a reality. In the following years, the amount of information that search engines could collect through tracking networks became so extensive and the processing cost became low enough that some engines began to generate search results tailored to the preferences of individual users based on their entire online footprint.

Despite this progress, the way we interact with a search engine has hardly changed over time: you submit a query, review the results and refine your query until the results are satisfying or nothing new comes up. While search results may be different for each user, the response to a search query remains the same for a single user, no matter what the context of that query might have been. If you've been researching snakes for the past hour and then enter "python" in a web search engine, it is likely that the search results will be about the programming language rather than the species of snake – the context of your recent query history has been completely ignored. In this example, the issue can easily be mitigated by adding "snake" as a second search term to clarify the intent, but this may be harder for complex research tasks where the user may not necessarily be aware of the relevant terminology.

The goal of session-based search is to support users in exploratory search tasks by identifying the common topic of the most recent queries and serving search results that match the current topic rather than the exact query terms. Instead of letting the user refine the query so that the search engine gets all the context it needs, we save the context in the search session and interpret subsequent queries under that prior. This way, a user does not only get more relevant results, it also allows the search engine to guide the user by suggesting topics that are relevant in the current context.

In this work we will investigate the foundations of a session-based collaborative search engine. In order to recognize the hidden topics in a search session, a topic model based on the underlying document collection is required. How a suitable topic model can be built and labeled will be discussed in Section 3. An introduction to session-based search will be given in Section 4, where we will investigate the structure of a search session, methods for the aggregation of the query history and the identification of relevant topics in the

current search context. As part of this work, the prototype of a session-based search engine has been built, which will be presented in Section 5. The subject of collaborative search will only briefly be discussed in the next chapter, a detailed examination must be deferred to future work. Section 6 lays out what else can be expected from an upcoming thesis.

2 Related Work

Topic Models are statistical models that help discovering the hidden topics in a document collection. Early work has been done by Deerwester et al. [22] and Hofmann [23]. The more recent works of Blei et al. [24, 25] will be discussed in Section 3. Fewer attention has been given so far to the related issue of automatic topic labelling, with occasional publications [29, 30, 31, 32, 33] since the appearance of this subject in 2007.

Session-Based Search as a concept has been discussed since at least 2003 [1], with first mention of the term "session-based search" in 2004 [2]. The definition of session-based search in this context is strictly limited to the idea of adjusting search results based on the last few queries of a user and not to be confused with the more general concept of a search session, which just describes the fact that users often submit related queries to a search engine in order to satisfy a search interest. It should also be distinguished from personalized search [3], which aims to learn a user's preferences over time and serve search results based on these preferences and other metadata.

Early work has been done by Shen and Zhai [1, 2], who mention the use of the query history to improve the ranking of search results. The topic has gained more attention since 2010, when the first *session track* [4] was held at the Text REtrieval Conference (TREC), though in some publications different nomenclature has been used, like *session search* [12] or *dynamic search* [5]. Notable research has been done by Yang et al. who model the search session as a Markov Decision Process [6, 7, 11, 12] with the user and the search engines as agents. Furthermore, query aggregation and the optimal weights of query terms in the context of a search session [8, 9, 10] have been investigated and an implementation was provided under the name *Dumpling* [5]. The works of Jiang et al. provided insights into user interaction and satisfaction with session-based search [13, 14].

Collaborative Search as defined in this work describes the participation of multiple users in the same search session at the same time. The idea to encourage collaboration

between users of a search engine is not new [16, 17], although the ways in which collaboration was achieved can be very different. One of the earliest works is the Community Search Assistant [15], which is a search engine that suggests related queries of other users while you are searching. Another example is SearchTogether [18], which supports groups of users in a search scenario by sharing queries and providing a chat application.

Morris [19] has shown that despite the fact that collaborative search tools do not have any notable market share, people are already searching collaboratively on a regular basis with the help of communication tools like chat applications or social media. These findings suggest a high demand for the right collaborative search tools, but so far there is no consensus about the best methods. Coagmento [20] is a collaborative information seeking framework that aims to support the search effort of small groups by providing a common dashboard where hyperlinks, text snippets and media files can be shared and reviewed. Querium [21] is another search engine that allows collaborators to rate search results and run similarity searches on individual results. However, none of these approaches incorporate a common search context that directly influences the search results of each collaborator, which is one of the methods proposed in this work.

3 Probabilistic Topic Models

An important prerequisite for efficient exploratory search is to have knowledge about the topics of a document collection, which are not inherently visible through keyword matching. The goal is to not only serve documents whose content matches a query string, but to better understand the search interest of the user and to find documents that are most relevant to that search interest. This however requires a consistent annotation of documents based on their contents, which is a huge effort when done manually by humans. However, with probabilistic topic models [25] it is possible to discover the abstract topics that occur in each document of a collection in an unsupervised learning process.

3.1 Topic Modeling

Topic modeling is the process of uncovering the thematic structure hidden inside a document collection. As a result, documents (or parts of them) are annotated with a set of topics. Each topic can be represented as a probability distribution over a

fixed vocabulary, therefore we can recognize a topic by matching an expected frequency distribution of key terms against the contents of a document.

A popular method to create such topic models is *latent Dirichlet allocation* (LDA), "a generative probabilistic model for collections of discrete data such as text corpora" [24]. LDA was influenced by *probabilistic latent semantic indexing* (pLSI) [23], which in turn is an enhancement of *latent semantic analysis* (LSA) [22]. The common underlying assumption of these models is that documents are the result of a generative process that has it's root in the specification of a set of topics. Therefore, the computational task can be seen as the reversal of this process, the inference of the hidden topics from a collection of observed documents. A detailed description of LDA is beyond the scope of this work, especially since a good introduction has been written by one of it's originators, David Blei [25].

Since it's introduction in 2003, the LDA model has been extended in various ways to mitigate some of it's shortcomings. An example is the nested Chinese restaurant process (nCRP), "a stochastic process which assigns probability distributions to infinitely-deep, infinitely-branching trees" [28]. This does not only solve the problem of having to define a fixed number of topics ahead of time, as otherwise required by LDA, but also creates a linked topic hierarchy instead of a set without any relations. The author-topic model [27] is another extension that concerns the inclusion of authorship information, which can be particularly useful for academic publications. The idea is that topics are distributed over authors instead of documents, therefore the topics of each document are modeled as the average of the contributing author's topics.

Due to the recent popularity of topic modeling in general and LDA in particular, there are plenty of implementations to choose from. In this work, the Python library Gensim [26] will be used, which provides a scalable implementation of LDA and some related models.

3.2 Topic Labeling

While LDA topic models can be very useful in identifying relations between documents based on the similarity of an underlying topic, they do not provide us with any means to actually define or name individual topics. Having a name for a topic can help in acquiring additional knowledge about that topic from other sources, which in turn can be used to improve search results. Furthermore it enables us to make judgments about the accuracy and quality of the topic model and helps humans, starting from a loose set of topics, to build an ontology that represents the relationships between entities of a

domain. A search engine can additionally benefit from such an ontology in that it can adequately visualize important topics to the user who can navigate through topics and filter results.

Each topic in an LDA model is represented by a vector that assigns a probability to each term that defines the topic. These marginal probabilities [24] can be used to find the most significant terms that identify a topic and it is possible, with some background knowledge, to derive a name for the underlying concept from these terms. This however requires manual review by humans with knowledge about the domain of interest, which is a time-consuming task.

To mitigate this issue, research has been done in recent years on the issue of automatic topic labeling. The general approaches are to use the terms with the highest marginal probabilities for each topic to either derive labels from relevant terms and documents of that topic [29, 33] or to use external knowledge sources to map the discovered topic to a known concept [30, 31, 32]. The knowledge-free approach is very flexible and able to derive labels even for new topics that just came up in news articles or social media. The knowledge-backed approach on the other hand allows us to link topics with other knowledge sources, which helps us to map a topic model to existing ontologies and to gain further knowledge of that topic. Therefore, topic labeling based on the methods described by [31, 32] seems to be the more promising solution for this work.

Still, the research so far has shown that fully automated topic labeling is not yet feasible. Hence the generated labels should additionally be screened by human reviewers, which is still a non-trivial task, but should be manageable for a customary amount of topics in the low triple-digit range.

4 Session-Based Search

A search session describes all interactions of a user with a search engine that are required to fulfill a single information need. This may be a simple question that can be answered with a single query (e.g. getting a weather forecast), or a complex task that requires careful review of the search results and a constant refinement of the search query (e.g. planning a holiday).

Contemporary web search engines improve their search results quite successfully by analyzing large query logs and identifying past search sessions with the goal to predict the most likely information need from an otherwise vague or ambiguous query. The

result of this analysis can be used to suggest terms while users are typing in their query or to expand the submitted user query to include more terms that the user likely would have chosen in the next query anyway. A dependable approach to identify and harness search sessions identified in query logs is the query-flow graph [35]. While large query logs can be a valuable resource for a search engine, there are often no query logs of sufficient size available for many search applications, usually due to a relatively small number of users or the high complexity and/or diversity of topics.

Session-based search, as defined in this work, aims to support the user in his search task without the need to have any knowledge about previous search sessions by other users. Instead, the best results will be calculated from just the data that is available in the current session.

4.1 Session Data

A search session S consists of one or more steps. Each step begins with the submission of a new query by the user and ends with the next query. A step in a session S is identified by it's index position in the list of steps $s_1, \ldots, s_n$. A single step is defined as

$$s_i = (q_i, u_i, d_i, s_{i-1})$$

where q is the query submitted by the user, u defines the user's actions during that step, d is the derived data that was generated by the search engine for that step and s_{i-1} is the step before the current step (or none, if $i - 1 = 0$). Similar definitions have already been suggested by Yang et al. [7, 12]. The definition used in this work was intentionally held abstract to give implementations of this concept the necessary freedom to choose which data of a session they want to harness. This is the definition of q, u and d in this work:

- query $q = (q_{f_1}, \ldots, q_{f_n})$, where each element of the tuple is a query string that is associated with a search function f_i. This will usually contain a full-text search function, but may also include more specific filters like the date of publication (before or after) or the name of the author of a publication. The selection of filters depends on the type of data that should be supported.
- user actions $u = (t, c)$, where t is the time stamp of the initial query submission and c is the list of clicked results, along with a time stamp for each click. This information can contribute to the ranking of future search results, because we can

assume that a user is interested in a link that was clicked, except if the time to the next activity (new query or another click) was very short.

- derived data $d = (e, r, m)$, where e is the expanded search query, r is the list of search results and m is the list of the currently best matching elements in the topic model along with their scores.

While q and u consist of user input that cannot be later restored, d can be derived at any later point in time, given a session S where at each step d_i has not been saved. Still, it can be useful to store d for various reasons: In session-based search, the state of a session is highly dependent on the state of all previous steps. Recalculating the entire query history during each step would slow down the search engine, while storing the state would only cost a justifiable amount of memory. A useful side-effect of this would be the possibility of efficient navigation in query history: the user could return to a previous step of the search session, review more results and move from there in a different direction.

4.2 Query Aggregation

An important feature of session-based search is the inclusion of terms from past queries into the calculation of the search result for the current query. A simple approach to achieve this is to join all past and the current query using the OR operator, but this may reduce the correlation of the search results with the most recent search query, which may be unexpected and even frustrating for the user. Therefore, it is advisable to give more weight to the most recent query compared to the previous ones.

This can be achieved by calculating the search results r for each query q in the current session individually. To form a unified list of search results r', the results of each search query $r_1, \ldots, r_n$ are joined in a single list and ordered by the score of each search result. Before the results are joined, the scores of each result list are multiplied by a weighting factor w_i, whose value depends on the position of the query in the query history. By selecting a larger w for more recent queries, the results matching these queries are preferred over the results of previous queries.

Guan and Yang [9] have investigated the optimal weight for each query depending on the distance of the query to the most recent one. Based on data from TREC 2010-2012 they evaluated the relative relevance of the search results in a session for all past queries in the session compared to the most recent one. Their findings show that relevance

generally decreases with increasing distance to the latest query, with the exception of the very first query, which always remains about as relevant as the second to last query.

These experimental findings can be roughly approximated by an exponential function with a base λ within about $[0.7 \ldots 0.9]$ and the exponent set to to $n - i$ for q_i (with an exception for the first query):

$$Weight\,(q_i) = \begin{cases} \lambda^{n-i} & for\ \ i > 1 \\ \lambda^{n-1} & for\ \ i = 1 \end{cases}$$

The implementation of this aggregation method has shown a decent balance for a value of $\lambda = 0.8$ (see Section 5.2).

4.3 Topic Identification

In this section we introduce *topic identification*, a method to decide, based on the most recent search results and other session data, which topics are most relevant to the user's search interest in general and the last query in particular, as well as to find a reasonable balance between these poles. This task is not related to topic detection [36], which is mainly concerned with text segmentation by recognizing known topics and identifying previously unknown topics, e.g. for news aggregation. Because topic identification has – to the best of the author's knowledge – not been discussed in previous publications, a novel approach to solve this issue will be presented. Experiences gained during the investigation of this topic will be described, but a scientific evaluation of these findings must be deferred to future work.

4.3.1 Topic Aggregation

As a result of the work in chapter 3 we can already rely on a set of documents which have been linked to one or more concepts in a topic model. The next step is now to calculate a list of the dominant topics for a single search query.

A list of search results r consists of documents $d_1, \ldots, d_n$, where each document has one or more topics $t_{i1}, \ldots, t_{in}$ assigned, where i is the index of the document d_i in r. A simple solution would be to count the number of occurrences of each distinct topic t in the first 10 documents and sort them in descending order (topics with the same number of occurrences are ordered by the highest score of the documents they occurred in). This

would provide us with an ordered list where the most frequent topics occur first, but there is a number of issues this solution does not account for:

1. *raw topic count*: a relevant topic that occurs in only few search results can be overruled by a topic that is more frequent but actually less relevant.
2. *distribution of topics*: popular topics that occur frequently in the document collection will generally overrule less frequent topics, notwithstanding their relevance.
3. *relations between topics*: closely related topics are counted as distinct entities, therefore a less relevant subject that is represented in the model as a single topic may overrule a more relevant subject that is represented by multiple topics.

Let's try to address these problems one by one.

The issue *raw topic count* can be mitigated by combining a set of different aggregation methods. T_{high} defines a list of topics ordered by the highest score of all documents they occur in. This will get us the topics of the most relevant documents, no matter how often they occur. A reasonable balance between score and frequency can be achieved by ordering the list of topics by the sum of the scores of all documents that the same topic occurs in (T_{sum}). And also the already mentioned raw frequency of topics in the result list can be a useful aggregation that adds a simple majority vote to the mixture (T_{count}). The combination of these aggregations is non-trivial, because the calculated scores are not comparable between different aggregations. Experience gained during this work indicates (non-representative) that a weighted average of the aggregations, after being scaled to the same value range, gives decent results.

$$T_{avg} = w_1 \cdot \frac{T_{high}}{\max(T_{high})} + w_2 \cdot \frac{T_{sum}}{\max(T_{sum})} + w_3 \cdot \frac{T_{count}}{\max(T_{count})}$$

A weighted average is proposed, because certain aggregations seem to provide more relevant results than others. Experience suggests that T_{high} is usually more relevant than T_{sum} and T_{count} seems to be the least relevant of the three aggregations.

The second issue concerns the *distribution of topics* among all documents, which is likely to be non-uniform. To avoid the dominance of frequent topics over less frequent ones in a scenario where two topics should be equally important, it is proposed to add a factor to each topic based on the proportion of the topic's frequency in the result list compared to it's frequency in the entire document collection. The calculation of that factor is similar to *tf-idf* [34], where instead of the term frequency we count the topic frequency in the result list r. The inverse document frequency is defined as the logarithmically

scaled inverse fraction of the documents in the document collection D that are linked to a topic t. The resulting score is calculated as

$$\begin{aligned} \mathrm{tf}(t, r) &= |d \in r : t \in d| \\ \mathrm{idf}(t, D) &= \log \tfrac{|D|}{|d \in D : t \in d|} \\ \mathrm{tfidf}(t, r, D) &= \mathrm{tf}(t, r) \cdot \mathrm{idf}(t, D) \end{aligned}$$

$\mathrm{tf}(t, r)$ defines the number of occurrences of t in a document d from the result list r, $\mathrm{idf}(t, D)$ is calculated from the size of the document collection $|D|$ divided by the number of occurrences of the topic t in these documents and $\mathrm{tfidf}(t, r, D)$ is the product of the latter two.

Issue number three concerns *relations between topics*. It exists due to the non-uniform precision with which topics are assigned to documents (both by humans and machines), but also a possible imbalance when it comes to the granularity of the topic model itself. The proposed solution to this issue is to discover a common broader topic for related topics which is then temporarily added to all documents in the result list that contain one or more of the related topics. The discovery process works for (poly-)hierarchical topic models by finding matching ancestors between different topics. Because in a hierarchical model, all topics are related through the root node, there have to be strict limits for the range of discovery. Two important limits should be considered: No common ancestor must be selected from topics that are too close to the root node (the definition of "too close" depends on the size and structure of the topic model) and no common ancestor must be selected through too many recursive visits to a topic's parent (this limit defines which topics are considered to be "related"). The discovery of related topics should be applied recursively, until no more new topics are found or a recursion limit has been reached. If we apply this method to our previous problem, we will see that the more relevant related topics still score lower than the less relevant single topic, but the highest scoring topic will be the common ancestor of the related topics.

To bring it all together, the three proposed methods must be combined in reverse order: In a first step, we try to discover common ancestors for related topics and add them to the documents in the result list. Then, the modified version of tf-idf is calculated for all topics in the result list. Finally, T_{avg} is calculated and each topic's score is multiplied with it's corresponding tf-idf value.

4.3.2 Topic Shift

After we have determined the most relevant topics for the latest query, the next step is to use these results to adjust the topic centroid of the search session. The *topic centroid* is defined as a set of topics that are relevant for a search session and which will be preferred over other topics in subsequent queries. It can be stored as a list of tuples, each consisting of a topic t and a score s.

If there is no topic centroid yet, we can simply copy the results of the first topic aggregation. Otherwise, updating an existing topic centroid with the latest topic aggregation is a problem similar to the one discussed in 4.2 Query Aggregation: we want the centroid to adjust to more recent queries but still preserve the general topic of the search session in it's entirety.

Let's start once more with a simple solution and go from there: We can join the topic centroid with the latest topic aggregation by simply adding the new topics and summing up the scores of all matching topics. This would preserve all topics in a session and rank the most frequent and highly scored topics as most important, which are decent properties. Still, over a longer session a large number of topics may accumulate and it will get harder for new topics to replace existing ones, thereby locking the search results in on the dominant topics of the first few search queries.

To mitigate this issue, we multiply the score of each topic in the topic centroid by a fixed fraction $f_{cooldown}$ at each step in the session. When a new topic aggregation is added, each topic that is not part of the topic centroid is added as-is, while the score of each already existing topic is calculated according to this formula:

$$s = \max(s_{old},\, s_{new}) + w_{shift} \cdot \min(s_{old},\, s_{new})$$

where s_{old} is the existing score in the topic centroid, s_{new} is the new score in the topic aggregation and w_{shift} is a value between 0 and 1 that determines the rate at which scores will accumulate over time. This method will ensure that recurring topics have more weight than infrequent ones, but not too much to dominate the search session even if other topics become more important.

As an example, let's assume a topic model consisting only of t_1 with a score of 1, $f_{cooldown} = 0.8$ and $w_{shift} = 0.5$. Now a new aggregation is to be added, where $t_2 = 0.9$ and $t_1 = 0.7$. As t_2 was not part of the centroid before, we can simply copy it's value. The current value of t_1 is multiplied with $f_{cooldown}$, which results in 0.8 and the new

score is calculated: $s = \max(0.8,\ 0.7) + 0.5 \cdot \min(0.8,\ 0.7) = 0.8 + 0.5 \cdot 0.7 = 1.15$. Even though the score of t_1 was lower in the new aggregation, it's score in the topic centroid remains higher due to it's history, but not out of reach for the new topic t_2, which may take over after the next query.

How exactly the topic centroid should influence the result list of the next query is not yet defined and will be part of future work. The general idea is to add documents about topics that are part of the topic centroid to the result list, even if they do not match the query terms, and to boost document scores, if their topics are part of the topic centroid.

5 Implementation

In order to put the theoretical benefits of a session-based collaborative search engine to the test, the prototype of a search engine was built as part of this work. At the time of writing, it allows session-based search on a small collection of scientific papers. This prototype will become the foundation for future work, in which the problems of collaborative search and the implementation of a more sophisticated topic model shall be approached.

5.1 Document Collection

Selecting a suitable document collection for the search engine was not a trivial task. It should not be subject to restrictive licensing and be available at low cost or free of charge, to make it as painless as possible for other researchers to reproduce the results of this work. This is a common issue with contemporary research on session-based search, which relies on data sets like those used in the TREC Session Track [37] or the HCIR challenge [38], which are either expensive to receive (e.g. shipping of hard drives) or only available to selected researchers under a non-disclosure agreement. Also, the document collection should be suitable for exploratory search, which means the topics must support non-trivial search interests and be diverse enough, so that the user has the chance to find more relevant answers after several query iterations.

A source that fulfills these requirements is *arXiv* [39], a digital repository of scientific papers in the fields of mathematics, physics, computer science and more. All papers submitted to arxiv.org are freely available to the public and dumps of the entire document collection (currently about 1.2 million papers) are available for a small fee that covers the cost of bandwidth. In the most prominent categories, the number of submissions is

probably large enough to give a representative sample of the research done in this field, which is an important prerequisite for exploratory search.

For the current prototype of the search engine, a full-text index of papers submitted to arXiv in the category *computer science* has been built. This subset consists of ~110.000 papers, which account for 67 GB of PDF documents or about 5 GB of plain text (uncompressed, utf-8).

arXiv provides metadata for it's records through a public API which conforms to the Protocol for Metadata Harvesting of the Open Archives Initiative (OAI-PMH) [40]. This allows us to reliably retrieve information like the title, abstract and list of authors that would otherwise have to be extracted from the PDFs and provides some additional metadata like the date of submission and a list of categories.

To handle these data sources, we have written a set of tools (see folder *index-builder* in the attached source code repository). It contains

- a crawler for the metadata-API that stores relevant information about all papers in a single JSON file.
- a PDF parser based on the *pdfminer* library [41] that extracts the content of each paper as plain text.
- a script that maps the categories of each paper to the ACM Computing Classification System of 1998 [42].
- a script that creates and populates the search index

The mapping of categories was necessary because arXiv uses two different classification systems for computer science papers (ACM'98 and CoRR) that are not compatible. These categories are used in this work as a preliminary alternative to a proper topic model, so the choice was made to map all categories to the ACM system, which is more detailed and hierarchically organized.

In this work, Elasticsearch [43] is used as search index. Elasticsearch is a distributed full-text search engine based on Apache Lucene that solves most of the standard information retrieval tasks that lay the foundation for a session-based search engine. The mentioned script initializes a new Elasticsearch index and stores each paper as as a record containing the full-text content along with all metadata.

5.2 Search Engine

The prototype of the search engine consists of a web application written in Python that uses the micro web framework Flask [44] to process queries and serve search results. The actual full-text search is being handled by Elasticsearch, which exposes it's REST API to the search engine.

Indexing The search index contains all documents and their metadata in raw and analyzed form. The raw form allows us to access each document by it's identifier and retrieve it's contents like in a regular DBMS, while the analyzed form enables us to efficiently run full-text queries on an inverted index and retrieve a ranked list of results [45]. Before text fields are added to the inverted index, they are usually transformed by an analyzer, which is an algorithm that typically consists of tokenization (splitting a string into terms) and normalization (character encoding, casing, etc.). For text fields, two analyzers will be used: The standard analyzer splits text on word boundaries, removes most of the punctuation and converts all terms to lowercase. Elasticsearch also offers a specific analyzer for the English language, which additionally removes a set of common stopwords and transforms words to their base form using the Porter stemming algorithm [47]. There is also the option not to analyze a field at all, which won't let us search it's content, but allows us to use exact string matching. If a query is executed against an analyzed field, the same algorithm will be applied to the query (otherwise the query terms may not match the analyzed tokens).

Each field in the index has zero or more analyzers assigned. Fields that are not supposed to be searchable (identifiers, URLs) were not analyzed, date fields got a specific date analyzer assigned and all text fields were analyzed with the English analyzer (title, authors, abstract, full-text). Some fields additionally have the standard analyzer applied (e.g. the authors, which often contain non-English names) or no analyzer to support exact matching (e.g. on document titles).

Query Execution Each query submitted by a user is executed in the context of a search session. This is also true for the first query of a user, although the session context will be empty. Figure 1 visualizes the steps that are executed by the search engine for each submitted query.

This section contains a few magic numbers that, if not noted otherwise, are purely based on experience and may change in future work with improved evaluation methods.

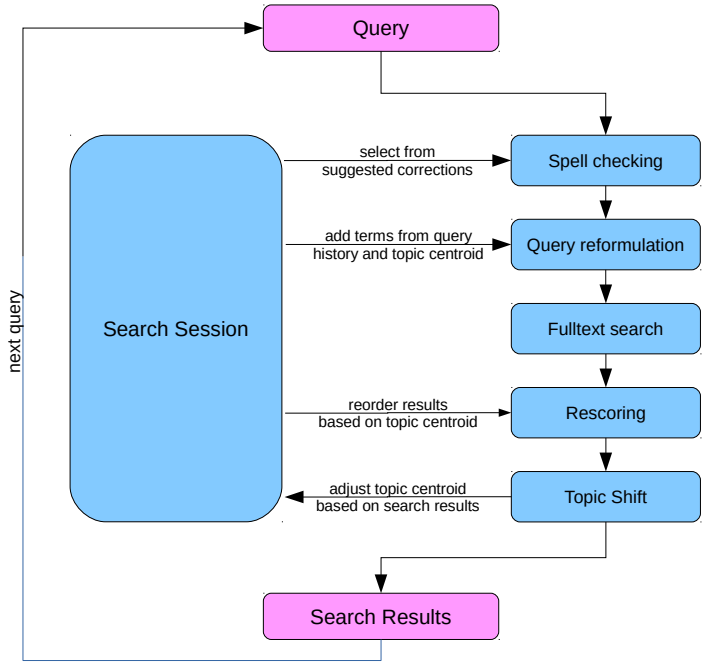

Figure 1: query formulation and execution in a search session

Correcting spelling errors is the first step, because it allows us to search for what the user meant, not what might have been typed accidentally. Spell checking is done using Hunspell [46] backed by an English dictionary, which generates decent results, but it just provides us with a list of words whose spelling is similar to the misspelled one and not with the most likely correction. Also, the dictionaries available for Hunspell often don't contain contemporary technical terms, which results in lots of false positives for expert document collections. In future work, it is planned to include much larger dictionaries based on word frequency lists generated from the document collection and the Google Books Ngram Corpus [48], which should allow us to pick the most relevant correction and significantly reduce the amount of false positives.

The next step is query reformulation, where the actual query is built that gets submitted to the search backend. Two methods are applied for full-text search: boolean query and phrase matching. The boolean query finds all documents that contain at least 67% of the terms in a query. This gives us a good recall, but more specific results can be retrieved using phrase matching, which is much more restrictive in that it requires all terms to appear in the same order as in the query, but therefore allows us to find contingent sequences of terms in a document. The requirement of having a strict ordering can be

slightly relaxed by allowing Elasticsearch to move query terms for a certain distance, which will increase recall at the cost of precision. Both query types are connected using the OR operator, and because the phrase query returns much fewer, but more relevant results, it's score will be boosted by a factor of 3.

For both query types, matching occurs on different fields with different weights w, relative to the weight of a match on the analyzed document text which is defined to be $w = 1$. A match in the document's abstract has a weight of 2 and $w = 3$ is given to a match on one of the authors or the analyzed title of a document. An exact match on a document title however results in weight of 4. Therefore, a while a boolean match on the document text would result in a weight $w = 1$, a phrase match on the title of a document would benefit from both the boost for phrase matching (3) and the document title (3), resulting in $w = 9$.

Now that we have defined a way to formulate a single query, we can apply the same method to terms from the query history. As described in Section 4.2, we concatenate all queries in the history using the OR operator and apply the weight function to determine the influence of each query to the overall result. In future work, it is also planned to include the most relevant terms from documents that are part of the current topic centroid.

The formulated query is converted to a format that can be interpreted by the REST API of Elasticsearch, which then executes the actual full-text search. The result is a list of documents matching the search criteria, in descending order of their match score.

In step four, the scores of the resulting documents are adjusted based on the current topic centroid. The goal is to prefer documents that are about one or more of the topics that are currently considered to be most relevant, even if their score as a result of the full-text search is relatively low. This is achieved by multiplying the score of each document that occurs in one of the topics from the topic centroid with a factor that is derived from that topic's current score.

Finally, the topic centroid must be updated (or created) based on the re-scored search result. In Section 4.3 this process has already been discussed in detail, but it has not yet been implemented in this prototype.

User Interface The prototype of the search engine provides a minimalist web interface that comes in the form of a traditional search engine results page (SERP) [49], sans advertisements. Figure 2 shows it's main contents:

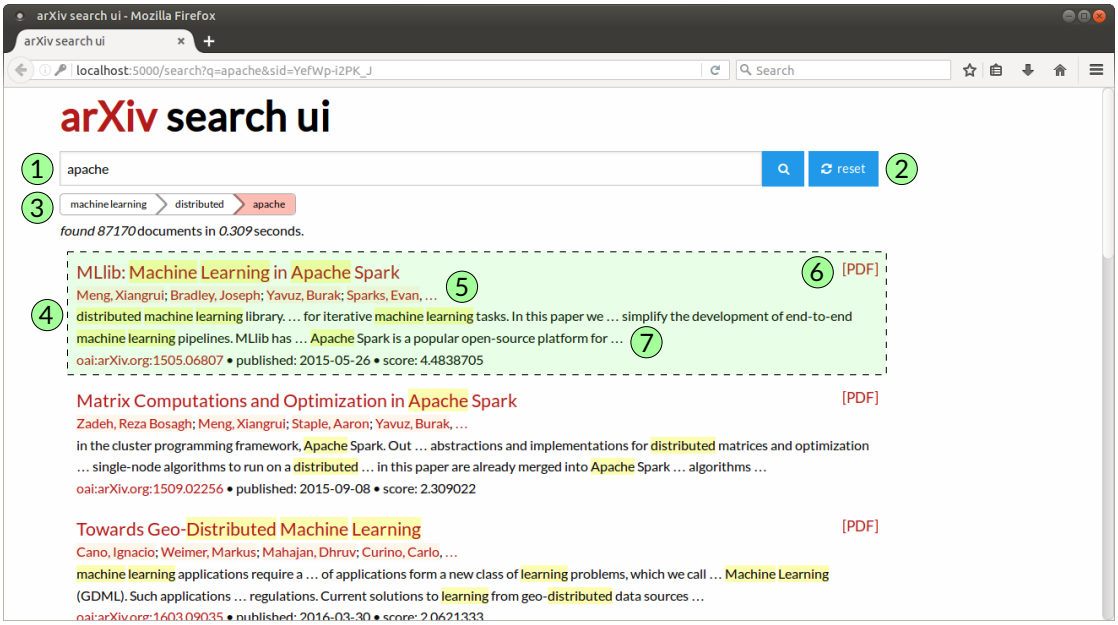

Figure 2: screenshot of the prototypical search engine results page

1. query box: a form field where subsequent queries can be submitted (contains the last query by default)
2. reset button: discards the query history and starts a new session
3. query history: all recently entered queries that contribute to the search results
4. the first search result (dashed box with green background)
5. title of the search result (links to the paper's summary page on arxiv.org) and list of authors (each name links to a search for other papers by this author)
6. direct link to the PDF version of this paper on arxiv.org
7. text snippets from the paper with highlighted search terms (also including terms from past queries in this session)

This user interface will be the foundation for upcoming work, in which a visualization of the topic centroid and social features for collaborative search are planned.

6 Future Work

The prototype of the search engine relies on the categorization of a document by it's submitter, which tends to be inconsistent and not very detailed. We plan to implement

LDA topic models as described in Section 3 using the Gensim [26] library. To approach the issue of automatic topic labelling, additional knowledge resources must be harvested. Therefore it is intended to build a full-text index of the English Wikipedia that reflects the link structure between articles and to implement functions to map from topics to Wikipedia articles or categories [31, 32].

The implementation of session-based search must be adjusted to support LDA topic models. We intend to add support for topic identification, as described in Section 4.3, which will also require some experimentation to find reasonable values for the parameters of various functions. Furthermore, it is planned to harness more of the available session data like the information about clicked search results. As there are no off-the-shelf solutions for these problems, the implementation is likely to be more time-consuming than the topic modeling task.

Collaborative search is an issue that has been neglected in this work so far. We intend to give users of the search engine the option to start a joint search session with other users, where a common topic centroid will be calculated from the query history of all participants. This leaves some challenging questions: What is the best way to derive topics from multiple distinct query histories and how does a common topic centroid influence the search results of participants? Collaborative search will require an intuitive user interface that allows participants to share results and help them with their research, but which at the same time does not distract them from their actual research task. The implementation of the UI as well as the actual server-side implementation of the collaborative search will require a considerable effort.

Furthermore, there are a few subjects concerning the search engine that should be addressed: the document collection should be expanded to cover all papers submitted to *arXiv*, which will remove the bias that was introduced by selecting a small subset of the entire collection. The frontend of the search engine is currently very conservative, which helps with usability, but fails to make use of the topical knowledge of the search engine. Therefore it is planned to visualize the current topic centroid and give users the option to use the suggested topics to navigate and explore new topics.

7 Conclusion

We have surveyed session-based search, an information retrieval method that treats each query in the context of a search session and provides results based on recent queries and user interaction with search results. From past queries and search results we discover

the search interest of the user and serve relevant results that are not strictly required to match the query terms. From an unlabeled document collection we build a topic model using latent Dirichlet allocation, which helps us to discover the abstract topics that are hidden in the documents and to build an ontology that mirrors the structure of our knowledge base. We have presented the prototype of a session-based search engine that allows us to find publications that have been submitted to the online-repository *arXiv*. It was our intention to make the results of this work easily reproducible, therefore the prototype and all tools required to build the document index have been disclosed. In future work, we plan to add support for more sophisticated topic models, to evaluate the performance of the session-based search in practice and to implement functions that allow multiple users to collaborate in the same search session.

References

[1] Shen, X., & Zhai, C. X. (2003). Exploiting query history for document ranking in interactive information retrieval. In *Proceedings of the 26th annual international ACM SIGIR conference*.

[2] Sriram, S., Shen, X., & Zhai, C. (2004). A session-based search engine. In *Proceedings of the 27th annual international ACM SIGIR conference*.

[3] Teevan, J., Dumais, S. T., & Horvitz, E. (2005). Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th annual international ACM SIGIR conference*.

[4] Kanoulas, E., Carterette, B., Hall, M., Clough, P., & Sanderson, M. (2010). Overview of the TREC 2010 session track.

[5] Zhou, A. J., Luo, J., & Yang, H. (2015). DUMPLING: A Novel Dynamic Search Engine. In *Proceedings of the 38th International ACM SIGIR Conference*.

[6] Yang, H., Guan, D., & Zhang, S. (2015). The Query Change Model: Modeling Session Search as a Markov Decision Process. *ACM Transactions on Information Systems (TOIS)*.

[7] Luo, J., Zhang, S., Dong, X., & Yang, H. (2015). Designing states, actions, and rewards for using POMDP in session search. In *European Conference on Information Retrieval*.

[8] Luo, J., Dong, X., & Yang, H. (2014). Modeling Rich Interactions in Session Search. Georgetown. In *Notebook of the Text REtrieval Conference 2014*.

[9] Guan, D., & Yang, H. (2014). Query Aggregation in Session Search. In *Proceedings of the 3rd Workshop on Data-Driven User Behavioral Modeling and Mining from Social Media*.

[10] Guan, D., & Yang, H. (2014). Is the First Query the Most Important: An Evaluation of Query Aggregation Schemes in Session Search. In *Proceedings of the Asia Information Retrieval Society Conference 2014*.

[11] Luo, J., Zhang, S., & Yang, H. (2014). Win-win search: Dual-agent stochastic game in session search. In *Proceedings of the 37th international ACM SIGIR conference*.

[12] Guan, D., Zhang, S., & Yang, H. (2013). Utilizing query change for session search. In *Proceedings of the 36th international ACM SIGIR conference*.

[13] Jiang, J., Hassan Awadallah, A., Shi, X., & White, R. W. (2015). Understanding and predicting graded search satisfaction. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*.

[14] Jiang, J., He, D., & Allan, J. (2014). Searching, browsing, and clicking in a search session: changes in user behavior by task and over time. In *Proceedings of the 37th international ACM SIGIR conference*.

[15] Glance, N. S. (2001). Community search assistant. In *Proceedings of the 6th international conference on Intelligent user interfaces*.

[16] Smyth, B., Balfe, E., Briggs, P., Coyle, M., & Freyne, J. (2003). Collaborative web search.

In *IJCAI*.

[17] Smyth, B., Balfe, E., Freyne, J., Briggs, P., Coyle, M., & Boydell, O. (2004). Exploiting query repetition and regularity in an adaptive community-based web search engine. *User Modeling and User-Adapted Interaction*.

[18] Morris, M. R., & Horvitz, E. (2007). SearchTogether: an interface for collaborative web search. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*.

[19] Morris, M. R. (2013). Collaborative search revisited. In *Proceedings of the 2013 conference on Computer supported cooperative work*.

[20] Shah, C. (2010). Coagmento-a collaborative information seeking, synthesis and sense-making framework. *Integrated demo at CSCW*.

[21] Golovchinsky, G., & Diriye, A. (2011). Session-based search with Querium. In *HCIR conference proceedings*.

[22] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*.

[23] Hofmann, T. (1999). Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference*.

[24] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*.

[25] Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*.

[26] Rehurek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*.

[27] Rosen-Zvi, M., Griffiths, T., Steyvers, M., & Smyth, P. (2004). The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*.

[28] Blei, D. M., Griffiths, T. L., & Jordan, M. I. (2010). The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM*.

[29] Mei, Q., Shen, X., & Zhai, C. (2007). Automatic labeling of multinomial topic models. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*.

[30] Lau, J. H., Newman, D., Karimi, S., & Baldwin, T. (2010). Best topic word selection for topic labelling. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*.

[31] Lau, J. H., Grieser, K., Newman, D., & Baldwin, T. (2011). Automatic labelling of topic models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*.

[32] Hulpus, I., Hayes, C., Karnstedt, M., & Greene, D. (2013). Unsupervised graph-based topic labelling using dbpedia. In *Proceedings of the sixth ACM international conference on Web search and data mining*.

[33] Cano Basave, A. E., He, Y., & Xu, R. (2014). Automatic labelling of topic models learned from twitter by summarisation. *Association for Computational Linguistics*.

[34] Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

[35] Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., & Vigna, S. (2008). The query-flow graph: model and applications. In *Proceedings of the 17th ACM conference on Information and knowledge management*.

[36] Allan, J. (2002). Introduction to topic detection and tracking. In *Topic detection and tracking*.

[37] B. Carterette, E. Kanoulas, M. M. Hall, and P. D. Clough. Overview of the TREC 2014 session track. In *TREC, 2014*.

[38] Capra, R., Golovchinsky, G., Kules, B., Russell, D., Smith, C. L., Tunkelang, D., & White, R. W. (2012, January). HCIR 2011: the fifth international workshop on human-computer interaction and information retrieval. In *ACM SIGIR Forum*.

[39] arXiv is an e-print service in the fields of physics, mathematics, computer science, quantitative biology, quantitative finance and statistics. https://arxiv.org

[40] Van de Sompel, H., Lagoze, C., Nelson, M., & Warner, S. (2001). The Open Archives Initiative Protocol for Metadata Harvesting. https://www.openarchives.org/OAI/openarchivesprotocol.html

[41] Shinyama, Y. PDFMiner: PDF parser and analyzer. http://www.unixuser.org/~euske/python/pdfminer/

[42] The 1998 ACM Computing Classification System. http://www.acm.org/about/class/1998/

[43] Elasticsearch: RESTful, Distributed Search & Analytics. https://www.elastic.co/products/elasticsearch

[44] Flask: A Python Microframework. http://flask.pocoo.org/

[45] Gormley, C., & Tong, Z. (2015). Elasticsearch: The Definitive Guide.

[46] Hunspell. https://hunspell.github.io/

[47] Porter, M. F. (1980). An algorithm for suffix stripping. *Program*.

[48] Lin, Y., Michel, J. B., Aiden, E. L., Orwant, J., Brockman, W., & Petrov, S. (2012). Syntactic annotations for the google books ngram corpus. In *Proceedings of the ACL 2012 system demonstrations*.

[49] Höchstötter, N., & Lewandowski, D. (2009). What users see–Structures in search engine results pages. *Information Sciences*.