Metaphor: A System for Related Search Recommendations

Azarias Reda
University of Michigan
azarias@umich.edu

Yubin Park
University of Texas at Austin
yubin.park@utexas.edu

Mitul Tiwari
LinkedIn
mtiwari@linkedin.com

Christian Posse
LinkedIn
cposse@linkedin.com

Sam Shah
LinkedIn
samshah@linkedin.com

ABSTRACT

Search plays an important role in online social networks as it provides an essential mechanism for discovering members and content on the network. Related search recommendation is one of several mechanisms used for improving members' search experience in finding relevant results to their queries. This paper describes the design, implementation, and deployment of *Metaphor*, the related search recommendation system on LinkedIn, a professional social networking site with over 175 million members worldwide. Metaphor builds on a number of signals and filters that capture several dimensions of relatedness across member search activity. The system, which has been in live operation for over a year, has gone through multiple iterations and evaluation cycles. This paper makes three contributions. First, we provide a discussion of a large-scale related search recommendation system. Second, we describe a mechanism for effectively combining several signals in building a unified dataset for related search recommendations. Third, we introduce a query length model for capturing bias in recommendation click behavior. We also discuss some of the practical concerns in deploying related search recommendations.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms, Design, Experimentation

Keywords

Query Suggestions, Recommender System, Log Analysis

1. INTRODUCTION

Search plays an important role in social networking sites such as LinkedIn as it provides an essential mechanism for discovering members and content on the network. By the time members have scrolled to the bottom of a search results page, it is reasonable to assume that they have not found what they are looking for, at least on the first page. This is when related search recommendations are most useful: they allow users to refine their searches or explore variants by providing alternate queries to their original search. Related search recommendations ("related searches") provide an important tool for improving members' search experience in finding relevant results to their queries and improving overall search engagement.

This paper describes *Metaphor*, the related searches system on LinkedIn. LinkedIn is a professional social networking site with over 175 million members worldwide, serving billions of searches every year. Search is particularly important in the professional context of the LinkedIn network, and represents one of the primary avenues for members and companies to find and connect with one another. Metaphor is an important addition to the set of tools used to guide members in their search experience. These related searches are displayed at the bottom of the search results page, as shown in Figure 1 for the query "Hadoop". In this paper, we will describe the design, implementation, and deployment of Metaphor—including scalability and practical considerations of mining billions of queries—that drives related search recommendations. At the time of writing, Metaphor has been in production for more than a year.

Metaphor builds on a number of signals and filters that capture several dimensions of relatedness in search activity, populating a unified dataset that drives related searches. Related searches are computed and periodically updated offline in a batch mode as a series of Hadoop MapReduce [18] jobs, which handles parallelization.

There are four main signals that go into building the search recommendation dataset. The first signal is based on collaborative filtering [1]: analyzing what members search for together. Using sessionized search logs, we establish sets of related query pairs based on temporal proximity. Each member is allowed a limited number of votes in associating query terms with each other, and the system penalizes generally popular terms.

The second signal is based on a result-anchored relationship of query terms. Using search logs for queries and results, the system constructs a click graph for searches. This results in a bipartite graph from queries to search results, with a many-to-many relationship between the vertices. Unlike previous applications of click graphs for related searches [14, 29], our approach benefits from the personalized nature of search on social networks, and in particular LinkedIn, where results are largely ordered by degree distance in the social graph from the searcher. For a given query, there are often several clicks resulting from the search, and for a given search result, there are several queries that lead to the result being clicked. This graph reflects the collective click behavior of members. Using this graph, Metaphor generates related searches by anchoring

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

on results that relate queries with each other. We call this signal a *query-result-query* signal.

The third signal leverages query term overlap. This is a commonly used approach in query suggestion on web search engines. We refine this approach to first identify terms within a query that have more significance to its meaning. Metaphor evaluates the frequency of terms in the query log, and how often the parts of the query occur together. When generic terms are used together with meaning-rich ones, the system identifies the terms that are more important to the query (for example, "Hadoop" in "Hadoop developer"). After priority of each term is established, Metaphor finds related queries that share important terms.

Related searches can help users to refine a given search query. For example, if a user searches for "Hadoop" to find Hadoop developers, we could suggest "Hadoop engineer" or "Hadoop developers", a refined query. The fourth signal in the recommendation pipeline helps with this problem by looking at the number of terms in a given query and its suggestions: suggestions with more terms are more likely to be refined queries. We built a statistical model for capturing click behavior on results based on recommendation length, which Metaphor uses as a bias to filter recommendations from the previous three signals.

After experimenting with recommendations from the individual signals on the production website for isolated evaluation, we settled on a step-wise union combination of related searches. In this approach, recommendations from collaborative filtering are backfilled with query-result-query recommendations and then partial overlapping recommendations. After each union step, we perform a length-biased filter based on our analysis of past clicks on recommendations.

Our evaluation shows that this approach results in 15% more clicks compared to collaborative filtering, the best signal in the ensemble, and a common technique for deriving such related search suggestions in large-scale production scenarios [24]. Metaphor scales to billions of queries and the recommendations can be computed using idle cluster resources.

The main contributions of this paper include:

- The description of the design, implementation, and deployment of a large-scale related search recommendation system that mines billions of queries,
- A mechanism for effectively combining several signals in building a unified dataset for search recommendation, and
- A novel approach of using query length for modeling click behavior and biasing search suggestions to reflect search intent

In addition, we enhanced signals to generate related searches, which included a variation of query-result-query click graph method and a partial matching technique that weighs in the specificity of each query term. Finally, we evaluated Metaphor by performing large-scale bucket or split tests involving millions of queries and members, an effective way of measuring the quality of a real recommendation system.

The rest of the paper is organized as follows. First, we discuss related work in Section 2. In Section 3 we describe the design principles, data flow behind Metaphor, and algorithms that comprise the various signals in the system. We then discuss the implementation and deployment of the system on the LinkedIn infrastructure in Section 4. We describe our evaluation and results in Section 5. Finally, we provide our conclusion and plans for future work in Section 6.

2. RELATED WORK

Related search recommendation has been an active area of research in the context of web search [9, 22, 25, 34, 41], and recently, e-commerce site search [24]. Metaphor is influenced by this research and by recommendation systems at large, and takes advantage of some popular recommendation techniques such as collaborative filtering [36] in addition to utilizing click graphs and partial matches.

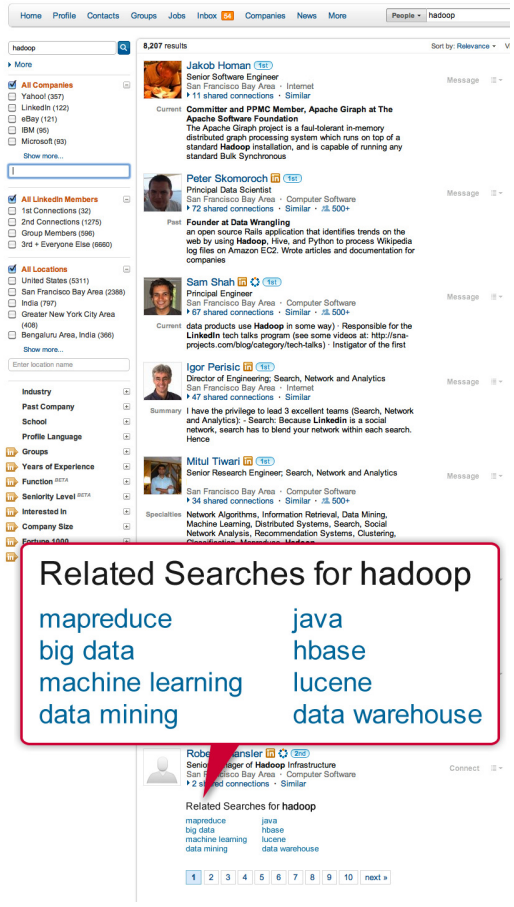

Figure 1: A screenshot of related searches in the context of a search for the query "Hadoop." The module shows a maximum of 8 results in a two-column layout and is available on every page of the search results.

Collaborative filtering based recommender systems have been successfully applied for movie recommendations [8] and product recommendations in e-commerce sites [24]. In Metaphor, query relatedness is implicit in query logs unlike movie rating data or historical purchases. Implicit relatedness among queries based on user sessions has been used for search recommendation in the context of web search engines [10, 11, 22, 25, 41]. Recently, query-flow graphs [10, 11] have been utilized to compute similarity between a pair of queries using the transition probabilities between them. Fonseca et al. [22] generate related queries by mining web search query logs and identifying queries that co-occur in the same session, while Zhang and Nasraoui [41] proposed finding related queries based on session and content similarity measures. A collaborative filtering approach based on implicit relatedness among queries works well (see Section 5), and forms an important component of Metaphor. However, in the context of a social network, collaborative filtering fails to discover related queries performed by different members, and it can be effectively combined with other

interesting signals that improve the overall performance of related search recommendation.

Utilizing clicked results for queries is another way of extracting related searches. Baeza-Yates et al. [7] introduced the idea of using clicks to infer related searches, which was then used to recommend queries by clustering searches leading to clicks on the same result. This approach was further developed to build an explicit bipartite graph with queries and results [4, 5, 14, 29]. The bipartite graph indirectly conveys the semantic relatedness of queries, which has been shown to be effective in search recommendation. Social networks provide a new context for click-based approaches that benefit from the underlying structure of members and the social graph. The personalized nature of search on LinkedIn, where results are largely ordered by degree distance, produces different results for different members, which often leads to different clicks for the same query. In aggregate, this allows us to discover diverse yet relevant recommendations. Furthermore, we have made several modifications to the algorithms and tailored them to the MapReduce framework as discussed in Section 3.

Partial matching techniques often can be found in query expansion research [15, 17, 38, 40]. Voorhees [38] shows lexically related words ameliorate mismatched vocabulary problems. Query refinement [26] and search shortcuts [13] are closely related to this concept. Our partial matching engine also suggests lexically overlapping queries to users. However, this process is significantly assisted by a prioritization step that identifies semantically important parts of a query. We utilize the frequency of terms and co-occurrence in the search log to rank the importance of terms within a query. This allows us to perform partial matches primarily based on terms that are essential to the query. We combine these approaches with a query length model that captures bias in recommendation click behavior. Based on statistical analysis of clicks on related searches, this length bias model is used to filter recommendations from the underlying signals. Although query length has been closely analyzed in the context of search characterization [3, 35], this is the first application of a length bias model in related search recommendation to the best of our knowledge.

Finally, ensemble methods such as bagging [12], boosting [32], stacking [39], and cascading [23] are widely studied in machine learning and statistics [20]. Some of these ensemble methods have been successfully applied in various recommendation scenarios such as movies [33] and music [21]. Most of these ensemble methods assume comparable performances in their base classifier engines, and expect overlap among datasets generated by each engine. However, each of our signals generates datasets with relatively small overlap and varied performance in different contexts. After experimenting with several alternatives, we devised a stepwise union model that combines the signals based on priority derived from empirical observations. As we will discuss in Section 5, this results in improved query coverage and clicks.

3. DESIGN

Metaphor is a data intensive system that relies on billions of queries in search activity logs to compute suggestions, which amounts to terabytes of search, result, and click data. This data is analyzed in building search recommendations based on a series of signals, notably correlating queries based on time, clicks, and the contents of the query string itself. Metaphor combines these underlying signals to generate a unified recommendation dataset for related searches. This section discusses these signals and the combination process in detail.

3.1 Collaborative Filtering

User	Query	Time (hh:mm)	Query pairs
John	q_1	17:04	(q_1, q_2), (q_1, q_3)
	q_2	17:06	(q_2, q_1), (q_2, q_3)
	q_3	17:07	(q_3, q_1), (q_3, q_2)
	q_4	20:52	-
Chris	q_5	21:34	(q_5, q_6)
	q_6	21:39	(q_6, q_5)
...			

Table 1: Query pairs based on temporal locality. q_4 is temporally distant from the other queries, thus it does not form a query pair. Note that each query pair is differently weighted according to the time between two queries.

The first signal we use to generate related search recommendations is based on collaborative filtering techniques. The collaborative filtering based signal uses temporal locality between queries as the primary factor for relating searches, that is, searches correlated by time are considered related. This is enabled by search logs that give us access to the search behavior of members in the social network.

Collaborative filtering in a social networking context can be enhanced by the strong sense of member identity as follows. We start by grouping together queries performed by the same member. We continue by identifying sessions within each member's search history using the inter-search time. We consider the member to have two sessions if the time elapsed between consecutive searches is above a threshold (currently set to 30 minutes). Using all the queries within a session, we form bidirectional (from, to) pairs of queries, which is a permutation rather than a combination. For each pair, we consider the time in between the two searches to assign a weight for the relationship between them. The farther apart the two searches are in the session, the lower the weight associated with the pair. Table 1 describes a brief example of the process.

Since some users often search for similar things, we would like to discount the impact of repeated searches across several sessions of the same user. We accomplish this by considering only one weight (the highest) for each unique pair that was generated by the same user. At the end of this process, we have each pair associated with a weight, with potentially many users having the same pair with different weights. This allows us to calculate a TF-IDF (term frequency, inverse document frequency) [6] score that will eventually be used for recommendations.

To compute the TF-IDF score, we first aggregate all identical pairs, with the final pair weight given as sum of the individual weights from each pair. The TF component in TF-IDF score for each pair is the final pair weight. The IDF component is important because it dampens queries that are generally popular, which end up forming pairs with a large number of other queries. If such queries are not appropriately dealt with, we would have them as recommendations for an unproportionally large number of queries. For example, President Barack Obama is an active member of the site and is regularly searched out of curiosity; this penalization prevents searches for him from being incorrectly correlated with other queries.

We use a variant of the Jones-Robertson IDF measure [31] for our application as given in Equation (1). For a query q:

$$\text{IDF}(q) = \log \frac{d \cdot (N - D(q) + 0.5)}{D(q) + 0.5} \tag{1}$$

In this case, N is the total number of pairs in the dataset, $D(q)$ is the number of pairs that contain query q and d is a dampening factor, which was tuned using offline precision/recall evaluation. The resulting TF-IDF score is used to generate recommendations

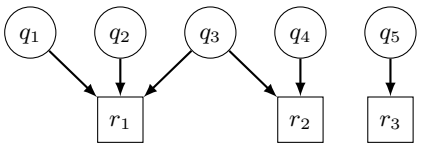

Figure 2: Query-result-query diagram. An arrow from q_i to r_j represents a click on result r_j for query q_i. Queries (q_1, q_2, q_3) are related through clicks on the same result r_1, and queries (q_3, q_4) are related through clicks on the result r_2.

from the collaborative filtering signal.

3.2 Query-result-query (QRQ)

The second signal we consider in constructing the related search dataset is relationship through member click activity. The basic idea in this signal is that by paying attention to similarity in click behavior for queries, we can discern some relatedness among queries. The LinkedIn search engine orders results largely by degree distance in the social graph from the searcher; thus, users searching for the same query will often get different results based on their social graph. Moreover, users are likely to be presented with intersecting result sets even as they are searching for different terms. This allows us to analyze click behavior from several members, and build relationships among different queries that prompt users to click on similar results. As a simple example, if several members searching for "Hadoop" and "MapReduce" end up clicking on similar results, then it might be a good idea to associate these searches in our related search recommendation. This allows us to recommend "Hadoop" in future searches of "MapReduce" and vice versa. Figure 2 shows the idea behind the query-result-query signal.

We begin data processing by identifying results that were clicked from the search results page in our logs. We then form (query, clicked-result) pairs by associating each query with the corresponding results that were clicked by the user. To prevent bias from a few members who often click on the same result, we document a pair only once per member. We then aggregate all similar pairs and attach a count to each unique pair, $C(q, r)$, which essentially describes the number of unique members who clicked on result r for query q.

The data processing then branches to two complementary components. On one hand, we aggregate all pairs with the same clicked-result r, say $G(r)$, and rank queries based on the pair count obtained in the previous step. With that goal in mind, we obtain $G'(r)$ from $G(r)$ with a few optimizations as follows. First, we remove clicked results that have too many unique inbound queries. Such results are too popular to form a meaningful relationship among the inbound queries. Second, we remove results that are clicked only for a single query since we require at least two queries to be related with each other. Finally, we sort the words in each query alphabetically, resulting in a bag of words model. This allows us to combine queries that would result in the same search result (for example, "Hadoop developer" and "developer Hadoop"). We define branching B as a measure that captures how many inbound branches a clicked result has. For a query q and a clicked result r, we define:

$$B(q, r) = \frac{C(q, r)}{\sum_{(q', r) \in G'(r)} C(q', r)} \tag{2}$$

In addition, we aggregate all pairs with the same query, and rank the results that were clicked for this query using the pair count. Let $K(q)$ represents all the clicks from query q across all searches. We obtain $K'(q)$ by the optimizations as described in the preceding paragraph. We define relevance R that captures how important a particular clicked result r was to a query q as follows:

$$R(q, r) = \frac{C(q, r)}{\sum_{(q, r') \in K'(q)} C(q, r')} \tag{3}$$

With both components—ranked queries and ranked clicks—available, we join the two datasets to infer relatedness among queries. For each result associated with a query, we identify a number of other queries that also lead to the result being clicked.

To combine these measures, we define a QRQ score for related queries. Let for a query q, a number of other queries $q_i, \ldots, q_j$ are related through a clicked result r. The QRQ score for each of those queries with respect to q is:

$$\text{score}(q, q_i) = \sum_r R(q, r) \cdot \log(1 + B(q_i, r)) \tag{4}$$

Here the sum is over each result r such that at least one member clicked on result r for queries q and q_i.

The intuition behind the QRQ score is that we consider both how important the result was to the query, and how important the potentially related query was to the result. From our empirical results, we find that we get better recommendations by emphasizing relevance over branching, hence the log scale for branching.

Finally, we deal with the issue of some queries that are generally more popular than others. Even with the measures taken to decrease bias, we find that certain queries can sometimes dominate recommendations because of their general popularity. To dampen those suggestions, we calculate TF-IDF scores, much in the way we did for collaborative filtering, using the QRQ score from above as the TF component, and the IDF component as computed in Section 3.1. We use these adjusted scores when generating recommendations for the query-result-query signal.

3.3 Partial Matches

The third signal we consider in building the related search recommendation set is the term overlap of queries (partial matching). Although the basic principle in this approach is straightforward, several adjustments must be made to obtain good recommendations. With millions of unique queries, term overlaps are rather common, and the difficulty is in identifying a set of queries that are "meaningfully" overlapping. To tackle this problem, we start out by grouping unique queries together and counting their occurrence. This results in pairs of form (query, count) that are used as input for further processing.

For each unique query, we then identify the tokens (terms) that make up the query. The tokenization process incorporates some optimizations. First, we remove stop words from the set that add little meaning to the query. In addition, we also remove very short tokens. While this can potentially remove some meaningful tokens, the optimization measurably improves overall relevance of the recommendations. Using the query pairs and their associated tokens, we generate a new set of tuples of the form (token, query, count). We then regroup the tuples using the tokens as key, which gives us a set of queries that share the token.

Given the volume of queries that share a token with each query, we make two important observations. The first is that not all tokens in the query are equally important to the query in question. For example, if we consider the query "Hadoop engineer", the term "Hadoop" is more important to the query than the term "engineer". Accordingly, it is more useful to make recommendations that look like "Hadoop developer" rather than "mechanical engineer", although both queries share a token with the original query. To measure importance of a token t in the query, we use a flavor of

an inverse document frequency as follows:

$$\text{IDF}(t) = \log \frac{M - Q(t) + 0.5}{Q(t) + 0.5} \quad (5)$$

Here, M is the total number of unique queries and $Q(t)$ is the total number of queries that contain the token t. The more common a token is across different queries, the less significant we consider it for any one of them.

The other observation is that we can apply a similar principle to filter out queries that are generally popular. Therefore, the final scores we assign to potential recommendations depend on both how important the token is to the query, as well as how popular the other queries that are associated with the token are. For a given query, we first identify all the queries that share a token with it using the (token, query, count) tuple, and then score each potentially related query as a multiple of the token significance measure and the TF-IDF measure of the related query in the corpus of queries. We use this score to generate recommendations based on partial matches.

3.4 Length Bias

Related searches can help members refine a given query. If a member has a clear objective when searching, they might benefit from query suggestions for refinement as the initial query might not directly map to their desired result. For example, if a member searches for "Hadoop" to find Hadoop engineers, related searches can suggest refined keywords such as "Hadoop engineer" or "Hadoop developer". We refer to this type of query suggestions as "refined" queries.

We developed a statistical biasing engine based on LinkedIn members' overall behavior. The analysis of the click behavior on displayed recommendations suggests a feature that captures the properties of refined queries. From our analysis, the number of words in queries is related to refinement type of queries, where longer queries tend to be more refined. Furthermore, we observed that members often click suggested queries that are slightly longer than the original query, but not too much longer.

We call this behavior "drift," which translates to a members' click preference for slightly longer recommendations. The additive drift score, which models the drift behavior, depends on the length of the previous query and the next query. The new score S' is computed by adding the drift score δ to the original score S:

$$S' \leftarrow S + \lambda \cdot \delta \quad (6)$$

where λ controls the strength of the length bias module. Since the length bias drift takes an additive form with a control parameter λ, this bias term is flexible in combining with multiple signals. By setting λ larger, longer queries tend to receive a higher score. The value of λ is determined by A/B testing to capture members' preferences. The additive drift score δ is computed using a parameterized equation of the form:

$$\log(\delta|q_p) \propto -\frac{\parallel l(q_n) - (\alpha \cdot l(q_p) + \beta) \parallel^2}{l(q_p)} \quad (7)$$

where q_p and q_n are the previous query and the next query respectively, parameters α, β are greater than zero, and for a query q, function $l(q)$ returns the length of the query q. The parameters (α, β) in Equation (7) are estimated using search query logs. Table 2 illustrates how the length bias module affects the recommendation result for a query "Hadoop". The length bias lifts longer recommendations to higher rankings, resulting in more "refined" queries in top-N recommendations. Parameter estimation and experimental results will be further explained in Section 5.2.1.

Recommendations	Length	Score	L-bias	Total
HBase	1	23.21	0.70	23.91
Hadoop developer	2	20.30	4.26	24.56
Cloud computing engineer	3	11.30	3.49	14.79

Table 2: Effects of the length bias module when $\alpha = 1.4$, $\beta = 1.0$, and $\lambda = 5$. Scores and L-biases represent original scores and length bias drifts, respectively. Longer recommendations tend to attain higher rankings after the length bias adjustment; for example, "Hadoop developer", which was second in original scores, becomes the first recommendation. Note that parameters and terms are arbitrarily selected to illustrate the mechanism.

3.5 Ensemble Approach

After generating related searches using the three separate signals and a length bias filter, the next step is to create a unified recommendation dataset that will be shown to members. To decide how to best combine signals together, we turned to empirical analysis. As we report in the evaluation section, we first segmented the LinkedIn population and ran the various signals individually on the website to get measures of performance. These measures include precision/recall, click-through rate, clicks, and coverage rate, as our evaluation will show.

Our first model was based on binary classification of the three signals with respect to click-through, which was not successful: we find there is minimal overlap across the clicked recommendations for related search suggestions. This led us to analyze the general characteristics of recommendations that were generated from the signals. As we see in the evaluation, the overlap is relatively small not only in the clicked recommendations, but also in the actual datasets that were generated from the three signals.

The minimal overlap of suggestions allows us to consider a different approach—step-wise unionization—for combining recommendations. Since our individual analysis has given us information on relative performance of signals, we are able to prioritize signals in the step-wise union. Accordingly, we first backfill collaborative filtering results with query-result-query recommendations. This is then processed through the length-bias filter to promote potentially better recommendations. We then backfill the result set with partial match recommendations, and once again run the length bias filter. In each of the union steps, we take measures to remove duplicate or near duplicate recommendations using a threshold minimum edit distance between recommendations. When it is time to display related searches, we choose the top-N results from this union set.

3.6 Practical Considerations

As a system designed for public consumption from the beginning, we had to consider a number of practical issues before deploying it to end users. For example, we have a very strong profanity filter that conservatively removes potentially offensive terms from recommendations. We apply the filter in the query preparation phase before using queries as an input for any of the signals. In addition, we incorporate a mechanism to deal with common misspellings in recommendations: in our recommendation dataset for a given query, we look at all suggested queries that are within a threshold of edit distances (currently set to less than or equal to 2) from each other. All such recommendations are collapsed to the most common entry in the set, which often removes many of the misspellings and typos in the collection.

Another important consideration was language. LinkedIn is an international website translated in a multitude of languages. We bucket search queries by language, and related searches are computed on these buckets. As LinkedIn has strong identities, these

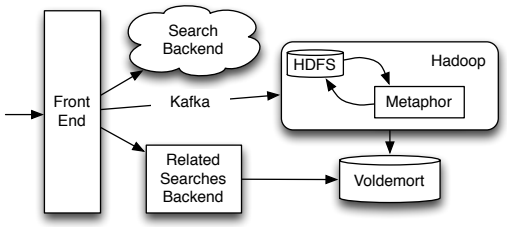

Figure 3: Dataflow in LinkedIn's production system. Tracking data is aggregated through Kafka, and after processing the data in Hadoop, related searches recommendations are stored in Voldemort to serve to end users.

buckets are determined from the locale of the member performing the search, which segments Metaphor's recommendations by language fairly well. This frees the system from performing language detection, which is difficult and often error-prone.

Finally, LinkedIn uses a number of mechanisms to detect spam and tag suspicious accounts accordingly. Like all of our production systems, the related searches pipeline receives input from these spam filters to remove queries that were performed by tagged accounts.

4. IMPLEMENTATION

The Metaphor pipeline runs on Hadoop, an open source implementation of MapReduce [18]. MapReduce provides a framework for processing big datasets on a large number of commodity computers through a series of steps that partition and assemble data in a highly parallel fashion. It further simplifies the process of writing parallel programs by providing the underlying infrastructure, failure handling, and simple interfaces for programmers.

Figure 3 shows how data flows through Metaphor. Incoming query logs and other activity-based tracking data is aggregated from production services using Kafka [27], a publish-subscribe system for event collection and dissemination. This log data can be easily represented as a set of tuples, which is a natural fit for Hadoop's distributed computing paradigm. As of this writing, Kafka aggregates hundreds of gigabytes of data and more than a billion messages per day from LinkedIn's production systems.

Metaphor consists of a series of MapReduce jobs implemented in Java and Pig [30], a high-level scripting language on top of Hadoop. In total, there are over 50 MapReduce jobs, from initially pre-processing the query logs to computing and combining the requisite signals. Figure 4 shows a sketch of how data flows within these jobs.

There is some dependency among these jobs, where a job might require outputs from another job to continue. These dependencies are expressed using a workflow manager. Jobs such as query preparation need to be prioritized before all other jobs, while each individual signal can proceed independent of each other. In addition to dependency management, the workflow manager provides a set of useful services like resource locking, log collection, error reporting, simplified configuration, and job scheduling.

The pipeline scales well, as our evaluation shows (see Section 5.3), and handily processes the production data size of LinkedIn. Related searches need only be updated infrequently, meaning the computational cost can be amortized away by using idle cluster time. Once computed, the resulting recommendations are served to end users through Voldemort [37], an open-source key-value system, which is similar to Amazon's Dynamo [19].

5. EVALUATION

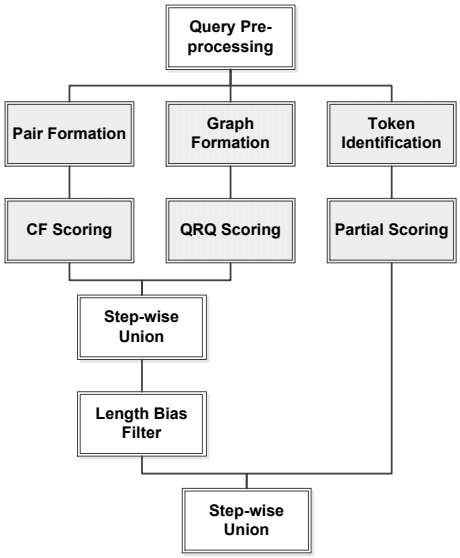

Figure 4: Dataflow in the Metaphor pipeline. Each stage consists of one or more MapReduce jobs that process loaded query and click data.

Iterative evaluation was an important part of the system development, and this section provides some of the characteristics and performance measures we used as we tuned related search recommendations. The first evaluation approach we used was an offline measure of precision/recall for the various signals in Metaphor. Second, we segmented the LinkedIn user base and conducted several A/B tests that were used to provide online measurements of performance. We used the results of these evaluations to derive the unified approach for the current recommendation set on production. Finally, we evaluated the runtime of the Metaphor pipeline.

Our evaluation answers the following questions:

- What is the performance of each of Metaphor's individual signals and its ensemble approach?
- What is the impact of length biasing?
- What is the system performance of the Metaphor pipeline and how does it scale?

5.1 Offline Evaluation

Offline evaluation provides a scientific and easily repeatable mechanism for establishing performance and tuning parameters. As a proxy for relevance, we measure the accuracy of predicting a member's future searches given their current search using precision and recall. In general, analysis of top-N recommendations shows that accuracy metrics such as precision/recall provide a better performance evaluation than error metrics [16].

We segment the query log into a training and test set. We start by generating the top-N recommendations for the training set from each signal of interest, where N is the number of recommendations that would be displayed to the user. To determine correctness in the test set, we use a moving time window of length K, which determines how far into the future from the current search we should consider in the member's search activity. This correct set is defined as the set of searches that were performed by the user in the following K minutes.

Precision and recall are defined as the mean of the average precision and recall, respectively, computed per member. A related

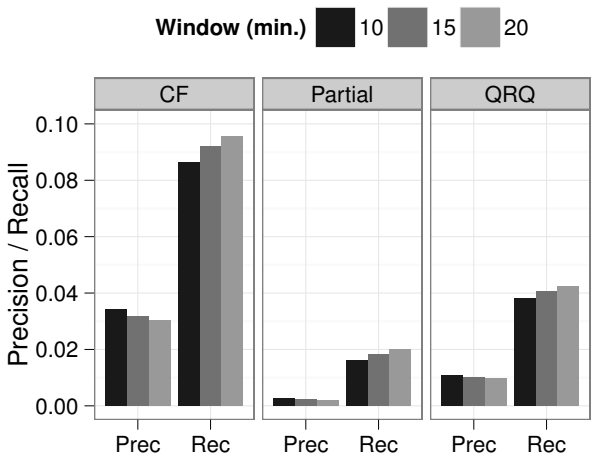

Figure 5: The mean average precision and recall measures for the top-10 recommendations as we vary the moving time window used to determine the correct set.

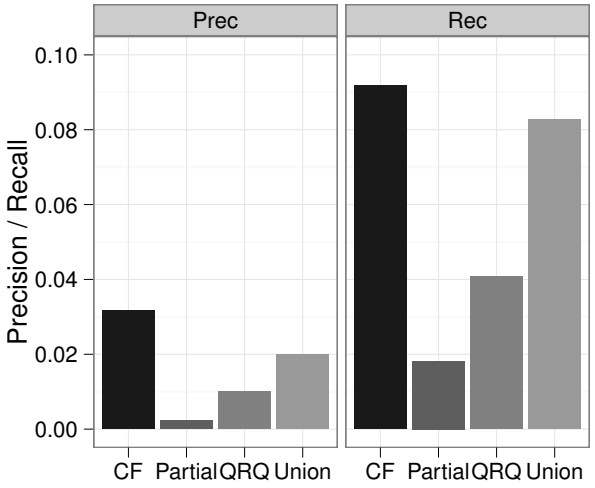

Figure 6: The precision/recall measures for the top-10 recommendations from the step-wise union-based ensemble approach. We use a 10 minute time window for the correct set.

search recommendation is relevant if the recommendation is in the correct set. Precision is the ratio of the number of relevant recommendations to the total number of recommendations N, while recall is the ratio of the number of relevant recommendations to the total number of searches in the correct set. This precision/recall measure indicates how well the recommendations reflect members' future search activity in the next K minutes.

Admittedly, this captures only a narrow dimension of recommendation quality by introducing a strong visibility bias: without members having seen possible recommendations, the precision/recall measurement only tells us the ability of our signals to predict future search behavior, not the system's ability to find other novel or serendipitous searches. Thus, this measure is not necessarily a strong indicator of recommendation performance, but it provides a reasonable metric for fast, iterative offline evaluation.

Figure 5 shows a comparison of mean average precision and recall for collaborative filtering, partial matches, and query-result-query. The precision and recall values are rather low as there are very diverse searches on LinkedIn, making predicting future searches quite difficult. As expected, recall increases as we increase the time window K, while precision decreases. At all cutoffs, the collaborative filtering approach has an advantage in both precision and recall.

Figure 6 compares mean average precision and recall for Metaphor's union ensemble approach. To make comparisons easier, we show performance for a fixed recommendation size and time cutoff, with the window size K set to 10 minutes and a top-N of 10. The results are generalizable across other time window and top-N values: the precision and recall values for the union approach are less than the best performing signal, collaborative filtering. While this is initially surprising, our discussion of the online evaluation reveals why the union approach performs better in practice because of a higher coverage of queries.

Furthermore, we evaluate the utility of related searches beyond the metrics of precision and recall. While large gains in mean average precision are detectable to the user, nominal improvements remain inconclusive [2]. Moreover, precision and recall in this context simply captures the ability of the recommendation engines to predict future search behavior, and is thus an incomplete proxy for relevance. We would like a more robust measurement that more directly evaluates a perception of quality.

5.2 Online Evaluation

We performed several isolated tests of the various signals in our model on independent segments of the LinkedIn population, commonly known as A/B or bucket testing. When combined with offline statistical analysis of recommendation signals, A/B tests allow a deeper understanding of user engagement. We focused on a few key metrics for this evaluation:

- *Coverage*. Not all typed queries have recommendations. The coverage rate is the fraction of queries that have recommendations for a given signal. The higher the coverage, the better members' experience if the recommendations are of high quality.
- *Impressions*. Impressions describe the number of times recommendations were displayed and are analogous to the trigger rate of the module on the search page. It does not make sense to track impressions at the per recommendation level (that is, 8 impressions as seen in Figure 1), as a member can at most click on one result.
- *Clicks*. Related searches recommendations might not always be useful to the user. A click by the user represents a vote that this particular recommendation is relevant.
- *Click-through rate (CTR)*. The click-through rate is the fraction of clicks over impressions.

For each of these measures, we took care to ignore page reloads and the browser back button to avoid incorrectly inflating the above metrics.

Related searches are not the main focus of the search result page (see Figure 1), but rather means for users to pivot to a more useful, novel, or serendipitous search. It is tempting to consider CTR as the sole basis for evaluation, but more careful consideration is needed. It would make sense to count each individual search *result* on the page as an impression—changing the number of results has a dramatic effect on the utility of the search—but for a small module on the bottom of the page (a maximum of 8 results in a two-column layout), absolute clicks is an additional important metric to consider as it encapsulates the notion of coverage as well as quality. As a pedagogical example, consider a recommender that has a single suggestion: it recommends "Hadoop" when given the typed query "MapReduce." Based on our query logs, this recommender

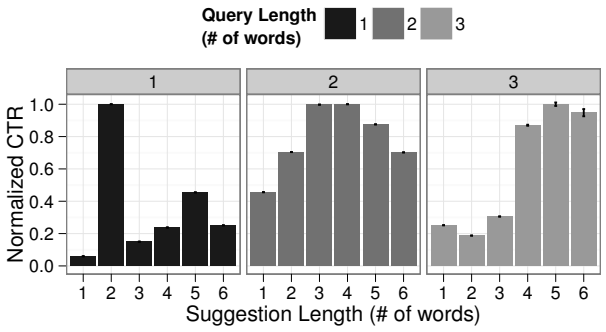

Figure 7: Suggestion length (# of words) compared to normalized CTRs for a given query length (# of words). Higher CTRs are exhibited for longer suggestions than the original query.

would have a very high click-through rate, but very low coverage, a handful of clicks, and ultimately little usefulness. Simply put, increased clicks from users indicate the recommendations are more useful.

As part of our online evaluation, we generated four datasets to run split or A/B tests based on collaborative filtering, query-result-query, partial matches, and step-wise union as described in Section 3. We segmented the LinkedIn population into four separate buckets of 10% each, serving one dataset to one of the members buckets for 2 weeks. We first evaluate the effect of length biasing, then gauge Metaphor's ensemble approach.

5.2.1 Length Bias

In our initial analysis, we observe a key feature that is closely related to users' clicks. This feature is the length difference between clicked suggestions and typed queries. Figure 7 shows the CTRs, which are normalized by the highest CTR in the given query length. Somewhat surprisingly, the highest CTR appears on suggestions one term longer than the original query, indirectly suggesting that users tend to click more refined queries.

From our analysis, we observed that we can improve "future" CTRs by only showing refined queries. For example, suppose a user types a single word query, and we show only two word suggestions. This removes 21% of the total suggestions for a single word query, but increases the CTR by 23% ($p<0.01$). For two and three word queries, similar results are observed. Simply suggesting longer queries comparatively improves the CTR, but reduces coverage, impacting overall clicks and hurting performance.

Our length bias filter can be regarded as a concise formulation of this idea that utilizes users' query selection bias; in other words, we can softly adjust the length of the suggestions, as described in Section 3.4. From our A/B testing, the scale parameter λ is set as ≈ 20, which depends on the scale of the original score. Note that λ determines the strength of length bias as noted earlier. Equation (7) in Section 3.4 approximates the behavior in Figure 7, where $\alpha \approx 1.5$ and $\beta \approx 1$ is estimated by minimizing KL-divergence [28] between the parameterized equation and the observed CTR graph. Accordingly, we can reflect our users' length bias into the resultant query recommendation distributions with a simple maneuver, which would guarantee higher CTR values.

5.2.2 Ensemble Approach

Metaphor constructs three underlying signals for use in its recommendations. We observed that there was minimal overlap in the clicks among various signals. In other words, different queries seemed to be clicked for different recommendation approaches.

Signal	Jaccard index
CF and Partials	0.36
CF and QRQ	0.27
Partials and QRQ	0.24

Table 3: Overlap of recommendation signals. The Jaccard index is a statistic for comparing the similarity between sets and is defined as: $J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$ for sets A and B. As shown in the table, there is little overlap between the Metaphor's various signals.

Signal	# of recommendations		
	1–4	5–8	>8
CF	74%	10%	16%
QRQ	62%	18%	20%
Partials	27%	7%	65%

Table 4: The number of results for each signal. Collaborative filtering and query-result-query have relatively few recommendations for a given typed query.

This led us to analyze recommendation overlap in the underlying dataset for the various signals. Although there was a significantly higher number of recommendations for partial matches (about three times the volume of the other signals), this dataset was inflated by recommendations for tail queries that are not often searched for. As a result, the difference in the coverage of recommendations for the queries issued during our experiment was not that high.

Table 3 captures the set similarity among signals using the Jaccard index between the recommendation datasets, which simply gives the ratio of the intersection to the union. That is, for sets A and B, $J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$. The results indicate minimal overlap across the various signals.

This suggests that we could combine recommendations in a step-wise union. To further understand the implications of using an ensemble, we looked at the depth of recommendations for each signal: Table 4 shows the number of recommendations for each signal. We found that collaborative filtering and query-result-query also have a small number of recommendations for each query (less than 4 suggestions for the majority of queries). Combined with low overlap, this means a step-wise union can safely capture recommendations from multiple methods.

To establish priority in our union approach, we looked at the individual performance of the signals. Figure 8 shows the normalized click, impression, and CTR for each individual approach. On average, recommendations from collaborative filtering receive almost twice as many overalls clicks and twice the CTR of partial matches and query-result-query-based recommendations. Naturally, Metaphor should show collaborative filtering-based results first if available. Query-result-query matches are the second best performing signal and should be shown second. Finally, recommendations based on partial matches should be shown.

We also A/B tested this step-wise union ensemble, the results of which are also shown in Figure 8 and validate this step-wise union ensemble approach. While the CTR of the union approach is virtually identical to collaborative filtering, coverage increases by over 20% in the union model. As Metaphor's suggestions are able to span more typed queries, the absolute number of clicks improves by about 15% compared to collaborative filtering ($p<0.01$), the best signal in the ensemble. These clicks represent increased engagement from users and are indicative of overall higher quality recommendations.

These results differ from our offline evaluation because of a visibility bias. In the absence of ground truth, the offline evaluation

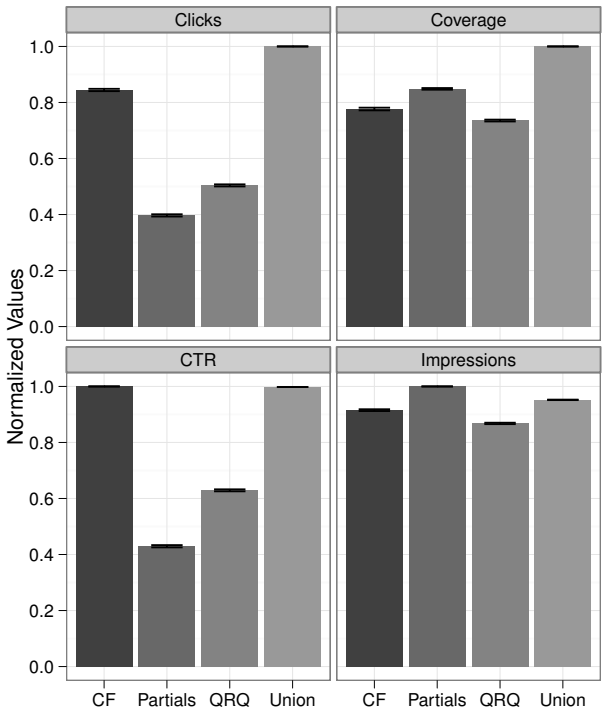

Figure 8: A/B results for clicks, coverage, CTR, and impressions with the individual signals and the union model. Each metric has been normalized to the maximum value within that metric.

uses subsequent searches as its gold standard, which penalizes discoverability. Here, precision and recall are only capturing future search behavior and are not necessarily indicative of recommendation quality and engagement in a live system.

5.3 Runtime Performance

Finally, we evaluated the runtime performance of the Metaphor pipeline. To understand the runtime behavior with respect to the number of input queries, we ran the pipeline on a Hadoop test cluster in isolation. This cluster consisted of 80 2x quad-core Nehalem-based Intel Xeon 5500 Sun Fire x4275 machines at 2.533 GHz with 24 GB RAM and 8x1 TB SATA drives configured in a 6:6 map:reduce ratio for a total of 480 available map and 480 available reduce slots. We ran several experiments by varying the number of input queries from 1 million to 3 billion and measured the completion time for the pipeline, as shown in Figure 9. Each experiment was run five times, and the confidence interval is indicated on the figure.

The pipeline scales quadratically due to the underlying nature of our algorithms. There are some optimizations that can be made. The partials signal is the most computationally intense as it generates more than 3 times the amount of output as the other signals. This signal could be computed asynchronously and refreshed less frequently than the rest of the pipeline.

More importantly, related search recommendations stay fresh for a relatively long period of time and can be refreshed rather infrequently. To test this, we segmented the LinkedIn user population into 3 A/B test buckets (10% each), refreshing recommendations at a weekly, biweekly, and monthly frequency for each respective bucket. Clicks and CTR were virtually indistinguishable between

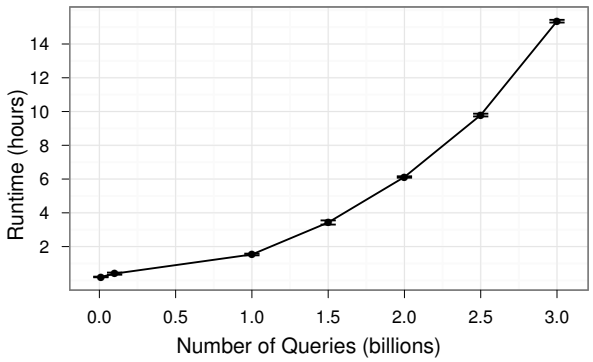

Figure 9: Runtime of the Metaphor pipeline while varying the size of the input query log. The pipeline was run on a dedicated 80-node Hadoop test cluster for 5 trials each.

the weekly and biweekly refresh buckets, with a 1.2% drop in clicks ($p<0.01$) and a 2.1% drop in CTR ($p<0.01$) in the monthly refresh bucket. This makes intuitive sense: these recommendations only change at the introduction or shift in query terms, which moves at the speed of the change in the professional lexicon.

In practice, the runtime of the pipeline is more than sufficient. Hadoop provides horizontal scalability so it is easy and cheap to add nodes, and due to the infrequent nature of this refresh, the computational cost is completely amortized away by using idle cluster resources.

6. CONCLUSIONS AND FUTURE WORK

This paper discussed the design, implementation and deployment of Metaphor, the related search recommendation system at LinkedIn. With millions of searches served every day, related search recommendation is an important addition to the suite of tools used at LinkedIn to improve member search experience. There are four main signals that went into building the recommendation dataset for Metaphor. One signal is based on collaborative filtering, and analyzes what members search for together based on temporal proximity. Another signal takes advantage of search personalization in the social network to derive relationships based on clicks. Additional signals look at partial matches among queries and the length differential between queries and suggestions for modeling click behavior. We use a step-wise union with priority for combining signals into an ensemble. Metaphor has been in production for more than a year.

We plan to work on two additional approaches to improve related search recommendations in the context of the social network. The first is to incorporate and evaluate a user feedback loop for displayed suggestions. A feedback loop can be useful because it allows us to experiment with new recommendations that would not otherwise have made it to the top-N suggestions displayed to the user. This can be done by replacing some of the recommendations with randomly selected choices from lower in the dataset. If a new replacement is clicked often, we can then promote it to a higher position in the recommendation list. Second, we also plan to work on personalizing related searches based on member searches and profiles, much like the core search experience that provides personalized results to members.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the

state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, 2005.

[2] James Allan, Ben Carterette, and Joshua Lewis. When will information retrieval be "good enough?". In *Proceedings of the SIGIR*, 2005.

[3] Avi Arampatzis and Jaap Kamps. A study of query length. In *Proceedings of the SIGIR*, 2008.

[4] Ricardo Baeza-Yates. Applications of web query mining. In *Proceedings of the ECIR*, 2005.

[5] Ricardo Baeza-Yates. Graphs from search engine queries. *LNCS*, 4362:1–8, 2007.

[6] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[7] Ricardo A. Baeza-Yates, Carlos A. Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In *Proceedings of the EDBT Workshops*, 2004.

[8] James Bennett and Stan Lanning. The Netflix prize. In *KDD Cup and Workshop*, 2007.

[9] Sumit Bhatia, Debapriyo Majumdar, and Prasenjit Mitra. Query suggestions in the absence of query logs. In *Proceedings of the SIGIR*, 2011.

[10] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, Aristides Gionis, and Sebastiano Vigna. The query-flow graph: model and applications. In *Proceedings of the CIKM*, 2008.

[11] Paolo Boldi, Francesco Bonchi, Carlos Castillo, Debora Donato, and Sebastiano Vigna. Query suggestions using query-flow graphs. In *Proceedings of the WSDM*, 2009.

[12] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2): 123–140, 1996.

[13] Peter D. Bruza and Simon Dennis. Query reformulation on the internet: Empirical data and the hyperindex search engine. In *Proceedings of the RIAO*, 1997.

[14] Carlos Castillo, Claudio Corsi, Debora Donato, Paolo Ferragina, and Aristides Gionis. Query-log mining for detecting spam. In *Proceedings of the AIRWeb*, 2008.

[15] Paul A. Chirita, Claudiu S. Firan, and Wolfgang Nejdl. Personalized query expansion for the web. In *Proceedings of the SIGIR*, 2007.

[16] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the RecSys*, 2010.

[17] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Query expansion by mining user logs. *TKDD*, 15(4):829–839, 2003.

[18] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the OSDI*, 2004.

[19] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41:205–220, 2007.

[20] Thomas G. Dietterich. Ensemble methods in machine learning. *LNCS*, 1857:1–15, 2000.

[21] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. Recommending music items based on the Yahoo! music dataset. In *KDD-Cup*, 2011.

[22] Bruno M. Fonseca, Paulo B. Golgher, Edleno S. de Moura, and Nivio Ziviani. Using association rules to discover search engines related queries. In *Proceedings of the LA-WEB*, 2003.

[23] João Gama and Pavel Brazdil. Cascade generalization. *Machine Learning*, 41:315–343, 2000.

[24] Mohammad Al Hasan, Nish Parikh, Byanit Singh, and Neel Sundaresan. Query suggestion for E-commerce sites. In *Proceedings of the WSDM*, 2011.

[25] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *Proceedings of the WWW*, 2006.

[26] Reiner Kraft and Jason Zien. Mining anchor text for query refinement. In *Proceedings of the WWW*, 2004.

[27] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, 2011.

[28] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.

[29] Qiaozhu Mei, Dengyong Zhou, and Kenneth Church. Query suggestion using hitting time. In *Proceedings of the CIKM*, 2008.

[30] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig Latin: a not-so-foreign language for data processing. In *Proceedings of the SIGMOD*, 2008.

[31] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60(5), 2004.

[32] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the IJCAI*, 1999.

[33] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-weighted linear stacking. *CoRR*, abs/0911.0460, 2009.

[34] Yang Song, Dengyong Zhou, and Li-wei He. Query suggestion by constructing term-transition graphs. In *Proceedings of the WSDM*, 2012.

[35] Amanda Spink, Dietmar Wolfram, Major B. J. Jansen, and Tefko Saracevic. Searching the web: The public and their queries. *Journal of American Society for Information Science and Technology*, 2001.

[36] Xiaofei Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in AI*, 2009:4:1–4:19, 2009.

[37] Roshan Sumbaly, Jay Kreps, Lei Gao, Alex Feinberg, Chinmay Soman, and Sam Shah. Serving Large-scale Batch Computed Data with Project Voldemort. In *Proceedings of the FAST*, 2012.

[38] Ellen M. Voorhees. Query expansion using lexical-semantic relations. In *Proceedings of the SIGIR*, 1994.

[39] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[40] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *Proceedings of the SIGIR*, 1996.

[41] Zhiyong Zhang and Olfa Nasraoui. Mining search engine query logs for query recommendation. In *Proceedings of the WWW*, 2006.