CaSMoS: A Framework for Learning Candidate Selection Models over Structured Queries and Documents

Fedor Borisyuk   Krishnaram Kenthapadi   David Stein   Bo Zhao
LinkedIn Corporation
Mountain View, CA, USA
{fborisyuk, kkenthapadi, dstein, bozhao}@linkedin.com

ABSTRACT

User experience at social media and web platforms such as LinkedIn is heavily dependent on the performance and scalability of its products. Applications such as personalized search and recommendations require real-time scoring of millions of structured candidate documents associated with each query, with strict latency constraints. In such applications, the query incorporates the context of the user (in addition to search keywords if present), and hence can become very large, comprising of thousands of Boolean clauses over hundreds of document attributes. Consequently, candidate selection techniques need to be applied since it is infeasible to retrieve and score all matching documents from the underlying inverted index. We propose *CaSMoS*, a machine learned candidate selection framework that makes use of Weighted AND (WAND) query. Our framework is designed to prune irrelevant documents and retrieve documents that are likely to be part of the top-k results for the query. We apply a constrained feature selection algorithm to learn positive weights for feature combinations that are used as part of the weighted candidate selection query. We have implemented and deployed this system to be executed in real time using LinkedIn's Galene search platform. We perform extensive evaluation with different training data approaches and parameter settings, and investigate the scalability of the proposed candidate selection model. Our deployment of this system as part of LinkedIn's job recommendation engine has resulted in significant reduction in latency (up to 25%) without sacrificing the quality of the retrieved results, thereby paving the way for more sophisticated scoring models.

Keywords

Candidate selection; Learning query models; Personalized search and recommendation systems

1. INTRODUCTION

Real-time large-scale personalized search and recommendation systems play a key role at Internet companies like LinkedIn. At scale, these systems pose unique challenges, particularly where personalized search results or recommendations must be computed from extremely large populations of candidate items, and where both the items and user context data are highly dynamic. At LinkedIn, we have multiple such large-scale applications, and have developed methods for addressing these challenges using information retrieval and machine learning.

To meet business requirements, some personalized search and recommendation systems must meet tight constraints that they compute results in real time, offer a high degree of data freshness, and respond with low latency. An example of such an application at LinkedIn is the job recommendations product, which computes personalized sets of recommended job postings for users on the site based on the structured, context data present in the publicly available fields of their user profiles. We can train machine-learned scoring functions that predict whether an item (*e.g.*, a job posting) is a good match for a given user, but in large-scale use cases it is not feasible to compute the scores for all possible items in real-time to serve a page load. Instead, we use a two-stage retrieval process where we first invoke a computationally inexpensive model to select a small candidate set of relevant items from the set of all documents that match the query, and then perform a more computationally expensive second pass scoring and ranking on the obtained candidate set.

Search engines are known to be good at retrieving a small set of relevant documents matching a given query out of a huge document set, so for real-time large-scale recommenders it is a natural choice to model the recommendable items as structured documents and define a function to construct queries for selecting candidates. But making effective use of a search engine to produce viable candidate sets (having good precision and recall, and a small candidate set size) is non-trivial. One could naively construct large disjunction queries using all the publicly available keywords in a user's profile, for example, but such an approach would likely match too many items and fail to reject many of the irrelevant ones.

In some cases, ad-hoc heuristics and business rules can allow us to reduce somewhat the size of the candidate set; for example, for a job recommendations system, we could construct a query that selects the jobs in the same geographic region as the user, or the jobs that are in the same industry as the user. But these approaches would not work well in all industries or geographic locations: in some cases, the queries could still match too many jobs, while in some others, more relevant jobs that did not match the ad-hoc criteria could be rejected.

In this paper, we propose *CaSMoS*, a machine learning framework for constructing effective models for candidate selection in personalized search and recommendation systems. We present techniques for learning candidate selection models expressed using Weighted AND (WAND) Boolean predicates [9]. In our approach, we use a constrained feature selection algorithm to learn positive weights for feature combinations that are used as part of the weighted

KDD'16, August 13–17, 2016, San Francisco, CA, USA.
© 2016 ACM. ISBN 978-1-4503-4232-2/16/08. . . $15.00
DOI: http://dx.doi.org/10.1145/2939672.2939718

candidate selection query. We have implemented this system to be executed in real time using LinkedIn's Galene search engine, and deployed in production. We perform extensive evaluation with different training data approaches and parameter settings, and investigate the scalability of the proposed candidate selection model. Our online experiments demonstrate the efficacy of our framework in achieving significant drop in latency (up to 25%) without sacrificing the quality of the retrieved results, thereby paving the way for more sophisticated scoring models. We finally present the lessons learned from the deployment of our system as part of LinkedIn's job recommendation engine that serves millions of job recommendations to millions of users.

2. RELATED WORK

Query optimization has been well studied in relational databases for structured queries [10]. Information retrieval (IR) systems are different from traditional databases in that, inverted indexes are typically used to support keyword queries, and only top-k results need to be returned. One well known work for query optimization and candidate generation in IR systems is [9], in which the authors proposed the formulation of queries in the format of Weighted AND (WAND) predicates and described an efficient algorithm to execute such queries. There has been a lot of follow-up work: using SVM to learn WAND queries towards generation of candidates for document classification was proposed in [4]; the approach in [9] was compared with other algorithms for candidate generation in learning-to-rank systems and determined to be the most efficient among the compared algorithms in [6]; more aggressive pruning stategies on a per-query-basis were proposed in [16]. In this work, we propose methods to learn WAND queries for candidate generation for personalized search and recommendation systems, where both query and document data are semi-structured. The semi-structured data in our scenario is different from traditional web search engines in the sense that there are multiple fields of text in both queries and documents, and it is more important to learn which fields of queries and documents should match instead of which keywords should match. This is the main difference between our work and previous candidate selection methods in web search. Also, in this paper we focus on how to generate the query instead of query execution, since conceptually the underlying search engine can be treated as an independent component, where previous work on optimizing WAND query execution can be applied.

Other types of queries have also been investigated for candidate generation. An approach for constructing query modifications in the web search domain using corpus-based SVM models was introduced in [12]. Boolean models for obtaining the set of minimal terms using an approximate Markov blanket feature selection technique and a decision tree to build the corresponding Boolean query were proposed in [5]. Genetic algorithms were investigated in [3, 13] to find query alterations for faster execution in IR systems.

Candidate generation is also related to blocking methods in the setting of record linkage and entity matching. A dynamic blocking algorithm to choose the blocking keys based on the data characteristics at run time, together with a MapReduce implementation, was proposed in [14]. The construction of blocking functions based on sets of blocking predicates was proposed in [8]. In that paper, the authors formulate the problem of learning an optimal blocking function as the task of finding a combination of blocking predicates. However, for blocking in entity matching systems, high recall is very important, since the goal is to try to match all records in one database with another. In personalized search and recommendation systems, the objective is very different, since we focus more on retrieving top-k relevant results instead of all relevant results.

Table 1: Structured representation of a hypothetical user profile

User field	Value of field
Title	Software Engineer
Company	LinkedIn Corporation
Industry	Internet
Location	San Francisco Bay Area, CA, USA
Skills	C++, Java, Linux, Machine Learning
Position summary	recommendation systems, professional content

Finally, the notion of contextual information has been investigated in varied disciplines such as psychology [7, 11] and computer science (see [2] and the references therein, for example). While these investigations pertain to defining and modeling different types of user context, we focus on the orthogonal problem of constructing effective candidate selection models in personalized search and recommendation systems, wherein the queries incorporate user context.

3. PROBLEM SETTING

Personalized search and recommendation systems form the backbone of several user-facing products at Internet companies such as LinkedIn. The underlying search / recommendation tasks can be modeled as information retrieval problems, wherein the query consists of the user context / interests expressed through user profile and (user provided) search keywords, if present, and the document corpus consists of millions of items of a particular type depending on the application (*e.g.*, the set of job postings in the case of personalized job search/recommendations). Due to the product and business requirements, we typically need to compute the results in real time, under strict latency constraints. While it is evident that the results need to be generated on the fly for search systems, one may wonder whether the results can be precomputed in the case of recommendation systems. However, in several recommendation scenarios, both the user context and the set of items are highly dynamic, necessitating the need for real time computation. On the one hand, relevant results need to be available as soon as a new user creates a profile, or a user changes the profile. On the other hand, the set of valid items may change frequently over time (*e.g.*, due to the arrival of new job postings or the expiration of job postings), and hence the freshness of results is crucial (*e.g.*, users would like to see recommendations of relevant jobs as soon as they are posted). Hereafter, we present in terms of recommendation systems, although our motivation and modeling are applicable for personalized search systems as well.

We next motivate the need for candidate selection as part of a two-stage retrieval process, especially considering the long, complex structured queries and the infeasibility of scoring millions of matching structured documents. Consider, for example, the task of recommending jobs to a hypothetical user who is currently employed as a Software Engineer at LinkedIn in San Francisco Bay Area, with expertise in C++, Java, Linux, and Machine Learning, and works on building recommendation systems for displaying professional content. The structured representation of this user's profile is shown in Table 1. Suppose that the items (jobs) are scored based on the following similarity features: (user_title, job_title), (user_skills, job_skills), (user_position_summary, job_skills). Even assuming such a simplistic scoring model and a relatively minimal user profile context, the number of jobs that need to be scored be-

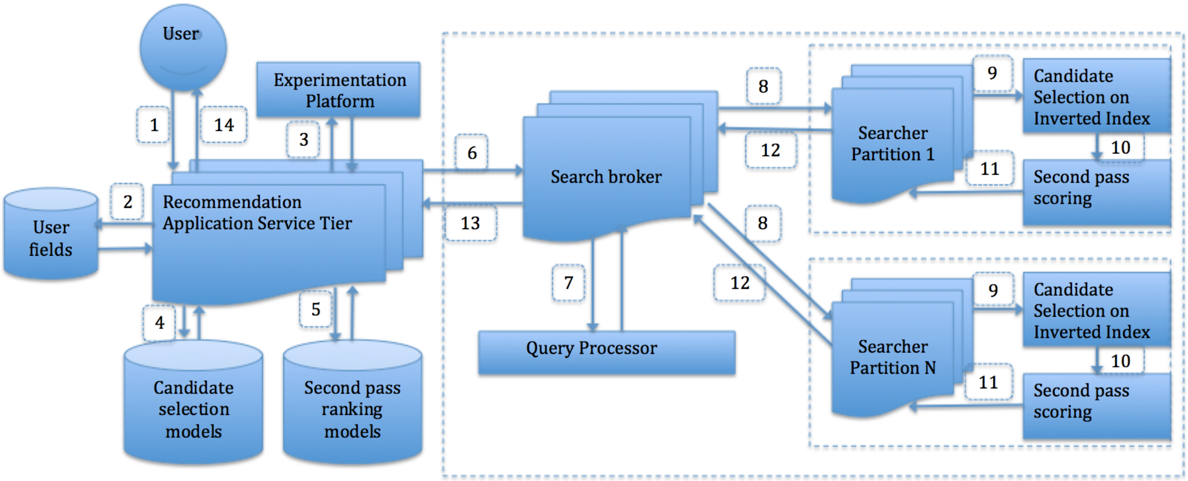

Figure 1: Online query processing and recommendation system.

comes very large, since any job whose structured representation includes one of {job_title: "Software Engineer", job_skills: "C++", . . ., job_skills: "Machine Learning", job_skills: "recommendation systems", job_skills: "professional content"} is a plausible recommendation. This problem is further compounded by the fact that the query may need to be expanded for robust match (*e.g.*, we can include titles similar to the user title, or locations similar to the user location). In practice, a rich user profile may consist of several structured attributes, and there could be hundreds of (user, item) features. Consequently, millions of items may need to be scored for a single user, which is not possible given the latency constraints, thereby highlighting the need for a two-stage retrieval process. Since our goal is to present a small set of most relevant results to the user, we would like to prune a large subset of the matching items that are not likely to be scored high, using a first stage, computationally inexpensive candidate selection model. Our problem can be stated as follows:

Given a long, complex structured query, select a small set of matching documents that are likely candidates for the top-k results for the query, while pruning matching documents that are not likely to be part of the top-k results.

4. RECOMMENDATION SYSTEM DESIGN AND ARCHITECTURE

We describe the overall design and architecture of the recommendation system deployed at LinkedIn, focusing on how candidate selection is performed. We first give an overview of LinkedIn's Galene search platform, followed by the description of the online recommendation system.

4.1 Galene Search Platform

In our deployment, we use LinkedIn's Galene search platform [15]. Galene uses Lucene-based inverted indexes at its core, and provides several functionalities desirable for our applications. The Galene platform supports periodic rebuilding of the entire index data offline in Hadoop, and offers a distributed service architecture with index partitioning and shard replication. The offline-built static base index when deployed in the online service is complemented by an online live-index tier, which consumes from a live Kafka [1] stream of document update events. The static base index and live index tiers together provide a single dynamic view of the search index to upstream callers of the service. Galene provides a rich query language that supports Weighted AND (WAND) queries [9]. Galene also offers a flexible API that allows for individual applications to define query expansion and query rewriting operations, as well as arbitrary scoring functions for ranking the matched documents. These APIs allow for easy integration of machine-learned scoring and candidate selection modules into Galene deployments, including those powering the methodologies discussed in this paper.

4.2 Online Query Processing and Recommendation System

Our online recommendation system uses a multi-tiered service oriented architecture (see Figure 1), and consists of a recommendation application service tier and a distributed search service tier. In this section, we provide an overview of the process by which these services respond to live requests for real-time recommendations. The description is annotated with references to the step numbers found in Figure 1.

4.2.1 Recommendation Application Service Tier

The recommendation application service mid-tier accepts requests from the front-end systems that are responsible for the user-facing LinkedIn web applications (step 1). Recommendation requests include the identifier of the entity for which recommendations are to be computed (in the example of job recommendations, this is the ID of the user visiting the site).

The recommendation service then retrieves the structured user fields data from a key-value store (step 2). It then obtains the experimental treatment for the user from an external A/B testing platform service (step 3). The experimental treatment identifies which machine-learned models should be used to compute the recommendations for the user. The recommendation service uses this information to retrieve the first-pass candidate selection model (step 4) and to retrieve the second-pass ranking model (step 5) from respective key-value stores. The models and the user data are then wrapped into a request object that is issued to the search broker service (step 6).

User field data is represented in a term-vector format, and consists largely of natural-language text-based fields including weighted uni-grams and bi-grams. Also included are non-text categorical derived fields (*e.g.*, which industries or geographic locations in our fixed taxonomy is a particular user associated with; the canonical version of user's current position title string, wherein the canonical version is obtained with respect to a large taxonomy of canonical titles; the set of skills for a user) along with corresponding weights.

4.2.2 *Search Service Tier*

The distributed search service tier consists of the following key components.

Search Broker: Search requests issued by the Recommendation Application Service are handled by the search broker service. The search broker applies the candidate selection model to the structured user fields data submitted with the request to synthesize a candidate selection query in the Galene query language (step 7). While the candidate selection model may be expressed in terms of (user, item) feature definitions, the constructed Galene query must be expressed in terms of item fields and values, possibly as a weighted Boolean predicate. For instance, for the task of recommending jobs to the hypothetical user in §3, the constructed Galene query could be a weighted Boolean predicate over literals such as job_title: "Software Engineer", job_skills: "C++", . . ., job_skills: "Machine Learning", job_skills: "recommendation systems", and job_skills: "professional content".

The request object is then decorated with the synthesized candidate selection query and is issued by the broker to the distributed set of search nodes, each of which contains one partition of the distributed index (step 8). Upon receiving the responses from the search nodes, the broker service merges the hits from the partitions and returns a list of the top-k items to the recommendation application service.

Searcher Node: A cluster of searcher node machines forms a partitioned and replicated distributed search index service. Each node runs as a stand-alone service, capable of answering queries against its own partition of the search index.

Upon receipt of the search request from the broker, the search node executes the Galene search query against its partition of the inverted index (step 9). In the typical example, the query is a Weighted AND query. As the matching documents are found via the information retrieval process, they are scored according to the second-pass ranking model provided with the request (step 10).

During second-pass ranking, interaction features are computed on-the-fly according to the second-pass ranking model, based on:

- the structured user fields contained in the request (*e.g.*, based on the user's profile data)
- the item (document) fields obtained via Galene's forward-index data

Documents whose scores exceed a defined threshold are inserted into a max-heap (step 11). When all the matching documents have been processed, the top-k documents in the max-heap are returned to the search broker (step 12).

The top documents returned by the searcher node (step 12) are subsequently merged with the results from other searcher nodes by the broker (step 13), and are passed back to the recommendation application service for post-processing. Post-processing may include applying filters and rules required for business reasons (for example, remove recommendations for ineligible jobs). Finally, the recommendation application service returns the finished list of recommended items to the front-end service for rendering on the site (step 14).

5. CaSMoS: CANDIDATE SELECTION MODEL LEARNING FRAMEWORK

We next present an overview of our candidate selection model, and describe the offline framework for learning this model based on user-item interaction log data. This offline framework is implemented in a distributed computing environment using Hadoop infrastructure, and makes use of LinkedIn's machine learning platform, based on Spark.

5.1 Candidate Selection Model Overview

We first present the intuition underlying our candidate selection model. As discussed in §3, for each user, millions of items could match with respect to each (user, item) feature used in the scoring model. However, the most relevant items are very likely to match on multiple (user, item) features. Thus, requiring an item to match on multiple features drastically prunes the number of items to be scored, while retaining the most relevant items. Further, it may be desirable to give different weights to different combinations of features: In an application scenario such as job recommendations, a match on the combination {(user_title, job_title), (user_skills, job_skills)} is more important than a match on the combination {(user_industry, job_industry), (user_seniority, job_seniority)}. Hence, we choose to represent the candidate selection model based on the Weighted AND (WAND) query operator [9].

Let F denote the set of (user, item) feature definitions. For $1 \leq i \leq k$, let C_i be a Boolean variable denoting whether an item matches on a combination $F_i \subseteq F$ of (user, item) features. Equivalently, C_i can be viewed as a conjunction over Boolean version of the corresponding features in F_i. Each clause C_i is associated with a positive weight w_i. An item is selected if the sum of weights associated with clauses that are true exceeds a threshold, θ. Thus, we specify the candidate selection model in terms of WAND Boolean predicate: For a given item, $WAND(C_1, w_1, \ldots, C_k, w_k, \theta)$ is true if and only if

$$\sum_{1 \leq i \leq k} w_i \cdot x_i \geq \theta,$$

where c_i is an indicator variable for C_i ($c_i = 1$ if C_i is true, and 0 otherwise).

We give an illustration using a toy candidate selection model and the hypothetical user profile in Table 1. Suppose that the model is specified with a threshold of 0.5, and in terms of just four (clause, weight) pairs:

1. ((user_title, job_title) $\wedge$ (user_skills, job_skills), 0.55)
2. ((user_title, job_title) $\wedge$ (user_position_summary, job_skills), 0.35)
3. ((user_industry, job_industry) $\wedge$ (user_position_summary, job_skills), 0.25)
4. ((user_industry, job_industry) $\wedge$ (user_seniority, job_seniority), 0.05)

Here, a job posting (item) satisfies a clause if the underlying features will all be non-zero. For instance, a job posting would satisfy the first clause if its title matches the user's title and there is at least one skill in common between the job and the user. We can observe that, for a job posting to be selected, it must either satisfy the first clause or satisfy both the second and the third clauses. For the user in Table 1, a job posting for "Software Engineer" (title) with "Java" as one of the skills will satisfy the first clause, and hence will be selected. Similarly, a job posting for "Software Engineer" at a company in the Internet industry that lists "recommender systems" as

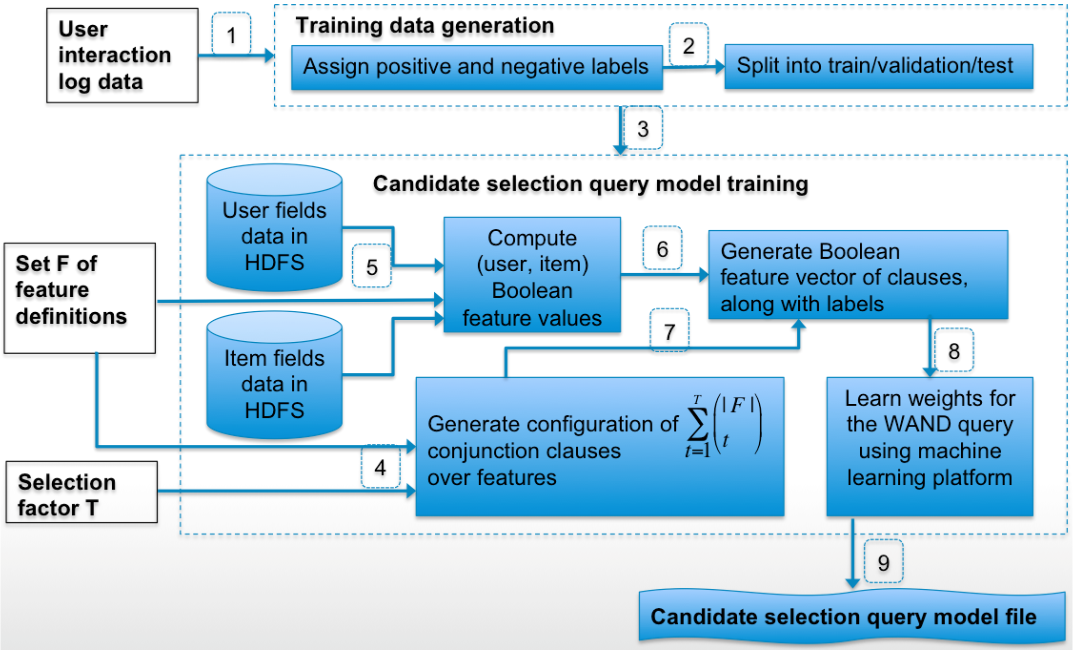
User interaction log data
1
Training data generation
Assign positive and negative labels
2
Split into train/validation/test
3
Candidate selection query model training
User fields data in HDFS
5
Compute (user, item) Boolean feature values
6
Generate Boolean feature vector of clauses, along with labels
Set F of feature definitions
Item fields data in HDFS
7
8
Generate configuration of $\sum_{t=1}^{T}\binom{|F|}{t}$ conjunction clauses over features
Learn weights for the WAND query using machine learning platform
4
Selection factor T
9
Candidate selection query model file

Figure 2: Offline framework for learning candidate selection query model.

one of the skills will satisfy both the second and the third clauses, and hence will be selected. However, a job posting for "Product Manager" at a company in the Internet industry will not be selected even if it lists any of the user's skills. Note that in this example, the fourth clause does not have any effect on whether a job posting will be selected, given the choice of the above threshold.

We next highlight the benefit of the above representation for the candidate selection model.

We are able to make use of WAND query operator, that is supported by LinkedIn's Galene search platform. Galene applies various optimization techniques as part of WAND query execution. For example, since the weights associated with the clauses are positive in WAND query, the retrieval system can select a document as soon as the sum of the weights for the satisfied clauses so far evaluated surpasses the threshold. In this case, there is no need to evaluate the remaining query clauses. This optimization would not be possible in case negative clause weights are allowed since the retrieval system would then need to evaluate all the clauses.

Algorithm 1 Algorithm for Learning Candidate Selection Model

Input: Set F of (user, item) feature definitions; User-item interaction log data; Selection factor, T.
Output: Candidate selection query model, specified as WAND query predicate.

```
1: Generate training data (labeled (user, item) pairs) from user-item interaction log data.
2: Create the configuration set of possible conjunction clauses, by taking combinations of up to $T$ (user, item) feature definitions.
3: Compute Boolean feature values for the (user, item) pairs present in the training data.
4: Generate Boolean feature vector of conjunction clauses, along with labels.
5: Learn weights for the WAND query.
```

5.2 Candidate Selection Model Training using User-Item Interaction Log Data

We next describe our offline system for learning the candidate selection model. Given a long, complex structured query, the goal of candidate selection can be viewed as achieving a separation between items that could be potentially in the top-k results and the items that are very unlikely to be in the top-k results. Hence, we adopt a supervised machine learning approach to train the candidate selection model. We present the components underlying our system in Algorithm 1, and the overall architecture in Figure 2. We elaborate on the key components below.

Generation of training data: This component assigns positive and negative labels to (user, item) pairs, based on user-item interaction log data, and splits this data into train/validation/test sets. The log data contains events of user interactions with the recommendation application. The events could correspond to an item impression (indicating that the item was recommended to the user) as well as different types of interaction such as clicking (indicating the user clicked on the item snippet to view more details), saving (indicating that the user saved the item for later reference), and other application specific interactions (*e.g.*, applying for a job). We can infer positive and negative labels for (user, item) pairs in several ways, possibly depending on the specific application. For example, for a given user, items that were clicked by the user could be treated as positive examples and the items that were never shown to the user could be treated as negative examples. Our experimentation choices are described in §6.4.

Configuration of potential clauses for WAND Query: Given the set F of (user, item) feature definitions and the selection factor, T, we create the configuration set of possible clauses for WAND query by taking combinations of up to T feature definitions ($\sum_{1\leq t\leq T}\binom{|F|}{t}$ clauses in total).

Consider the job recommendation example discussed in §3, and assume that there are just three feature definitions: (user_title, job_title), (user_skills, job_skills), (user_position_summary, job_skills). If the selection factor equals 2, the set of potential clauses would be: {(user_title, job_title), (user_skills, job_skills), (user_position_summary, job_skills), (user_title, job_title) $\wedge$ (user_skills, job_skills), (user_title, job_title) $\wedge$ (user_position_summary, job_skills), (user_skills, job_skills) $\wedge$ (user_position_summary, job_skills)}. At first, the last three clauses containing two features each may seem redundant since any item that matches these clauses would have been matched by at least one of the first three singleton clauses. However, since a conjunction of two Boolean features is more selective (and usually more discriminative at identifying relevant items) than either feature, the conjunction could be assigned a significantly larger weight compared to the singleton clauses, and thus could determine whether an item gets selected or not. In our implementation, we set the selection factor, $T = 2$.

Computation of Boolean feature values: Given the set of feature definitions, we next generate the Boolean version of the corresponding features for the (user, item) pairs present in the training data. In the job recommendation example above, the feature (user_skills, job_skills) would evaluate to *true* if and only if there is at least one skill in common between the user and the job. To enable this computation, the system that generates the structured field data for the users writes a copy of this data to HDFS, in addition to populating/updating the user fields store (used by the online query processing recommendation system). Similarly, the system that generates the item fields data writes a copy of the data to HDFS, in addition to updating the corresponding search index online. User fields may include publicly available parts of a user's LinkedIn profile such as the current job title of the user, seniority, function, industry, skills, and key terms extracted from descriptive fields such as the position summary of the user, current job description, and past job descriptions. In the case of job recommendation application, the item (job posting) fields may include the job title, seniority, function, industry, skills, and key terms extracted from the job description.

Generation of Boolean feature vector of conjunction clauses: This component generates the dataset needed for the machine learning algorithm, based on the following inputs: the training data, the Boolean feature values, and the configuration set of conjunction clauses. Each row corresponds to a (user, item) pair. The columns correspond to the conjunction clauses of (user, item) features, with the last column indicating the label. A conjunction clause is set to 1 (*true)* if and only if the user and the item match on all the features present in the clause.

Learning WAND query model: We use LinkedIn's machine learning platform to learn the weights and determine the optimal threshold for the candidate selection model expressed as a Boolean WAND query. Since the weights associated with the clauses need to be positive in the WAND query, we cannot directly use off-the-shelf machine learning techniques. Instead, we formulate our learning task in terms of logistic regression with positive coefficient constraints.

To satisfy this requirement, we experimented with two different algorithmic implementations of our learning task. The first algorithm can be thought of as performing constrained feature selection, treating the logistic regression training procedure as a blackbox. The second algorithm modifies the internals of the training procedure towards achieving positive coefficient weights.

Algorithm 2 Constrained Feature Selection Algorithm

Input: Dataset comprising Boolean feature vector of conjunction clauses, along with labels.
Output: Candidate selection query model, specified as WAND query predicate.

```
Train the logistic regression model with all features (Boolean conjunction clauses).
repeat
    Remove features whose coefficients are below a small (positive) threshold.
    Retrain the model without the removed features.
until Desired number of conjunction clauses are left.
```

Algorithm 3 Non-negative Coefficient Constrained Boundary Algorithm

Input: Dataset comprising Boolean feature vector of conjunction clauses, along with labels.
Output: Candidate selection query model, specified as WAND query predicate.

```
repeat
    Perform a step of the gradient descent algorithm, and update the coefficients of the features (Boolean conjunction clauses).
    Assign negative coefficients to zero.
until The training procedure converges, or the number of iterations reaches a limit.
```

We describe the two algorithms formally below.

Constrained Feature Selection Algorithm: Algorithm 2 first trains the logistic regression model with all features (Boolean conjunction clauses), and then iteratively prunes features with coefficient weights below a small positive threshold. Thus, both features with negative coefficient weights and very small weights are removed. This process is terminated upon reaching the desired number of conjunction clauses. In our implementation, we tune the desired number of clauses based on the observed latency over a set of queries.

Non-negative Coefficient Constrained Boundary Algorithm: Algorithm 3 performs modification to the gradient descent algorithm so that the negative coefficients are set to zero after each gradient descent step. This process is repeated until either the training procedure converges, or the number of iterations reaches a limit.

We experimentally observed that Algorithm 3 performed similarly as Algorithm 2 in terms of quality metrics. Hence, considering the added complexity associated with modifying the internals of the training procedure, we decided to use Algorithm 2 in our deployed production system. We used precision-recall (PR) curve to select the threshold parameter, θ for the learned WAND query. PR curve can be constructed by measuring precision and recall with respect to different choices of the threshold. We experiment with four different choices of threshold, corresponding to recall values of 0.85, 0.90, 0.95, and 0.99 respectively, and discuss the effect of varying threshold on the performance of candidate selection in §6.

6. EXPERIMENTS

We next present an extensive evaluation of our candidate selection model learning framework. We evaluate both efficiency (query processing time / latency) and effectiveness (quality) of the proposed candidate selection method, in offline as well as online settings. We also investigate the effect of varying training data and features used on the quality of the resulting candidate selection model, as well as the effect of varying index size and traffic loads on the query processing time.

6.1 Experimental Setup

As described earlier, we implemented our techniques to be executed in real time using LinkedIn's Galene search engine, and deployed in production as part of LinkedIn's job recommendation system. Our offline learning framework is implemented in a distributed computing environment using Hadoop, and uses LinkedIn's machine learning platform, based on Spark.

We performed our experiments over two weeks of job recommendations log data, collected during July 2015. This data is part of the user-interaction log data, which contains events of user interactions with various LinkedIn applications. The following types of user-job interaction events were used to train the candidate selection models:

- job impression (whether a job was shown to a particular LinkedIn user),
- job click (whether the user clicked on the job and viewed the details of the job), and
- job application (whether the user applied for the job).

The training data was generated from this log data, and was divided into three subsets: training set, validation set, and test set. We trained the models using the training set, selected the best model based on the validation set, and computed the metrics on the test set. Our baseline model does not use candidate selection, and instead scores all job documents that match structured attribute keywords from the user profile. This model is implemented by constructing a Boolean disjunction query from the structured fields in the user profile to the relevant structured fields in the job documents.

Our online experiments were performed using LinkedIn's A/B testing platform [17]. LinkedIn users were partitioned randomly into different buckets, and different random buckets of users were presented with job recommendation results obtained using different models, over a period of several weeks. The users in the control group were shown results obtained without using candidate selection (our baseline model), while the treatment buckets corresponded to different candidate selection models. Through these online experiments, we compared different models (including the one with no candidate selection) with respect to both efficiency and effectiveness, as explained below.

6.2 Metrics

We used the following metrics for measuring the efficiency as well as effectiveness of our system. We report these metrics relative to the baseline model (for business confidentiality reasons).

Efficiency metrics: We used LinkedIn's internal online performance monitoring tools to measure the query processing time (latency) metric, which represents the time lag between a request to the our system and the response back. We use 90th percentile latency as the efficiency metric (the time taken to process a query is less than this latency 90% of the time). Lower latency corresponds to faster page load times for the users, and hence is desirable from both business objective and user experience/engagement perspectives.

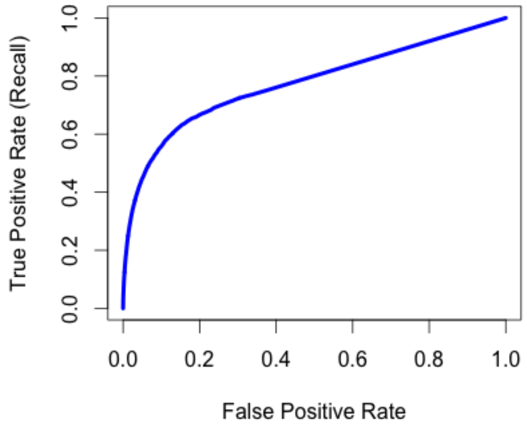

Figure 3: Receiver operating characteristic (ROC) curve for the candidate selection model.

Effectiveness metrics: We used both offline and online measures to quantify the effectiveness (quality) of the candidate selection models.

For offline evaluation, we computed the area under receiver operating characteristic curve (ROC AUC) and the area under precision-recall curve (PR AUC), both of which represent the quality of the candidate selection model (viewed as a binary classifier). The ROC curve is obtained by plotting the true positive rate (recall) against the false positive rate at various choices of the threshold, θ. The PR curve is obtained by plotting the precision against the true positive rate (recall) for various threshold choices.

For online evaluation, we computed the job application rate, which is defined as the ratio of the number of job applications (number of jobs applied to by users) to the number of job impressions (number of jobs presented to users). Higher job application rate corresponds to more relevant jobs being shown to users, and hence is desirable from both business objective and user experience/engagement perspectives.

6.3 Effectiveness and Efficiency of Learned Candidate Selection

We first present the results from offline analysis of the candidate selection model, viewing it as a binary classifier (type 2 training data approach is used for assigning positive and negative labels – see §6.4). Figure 3 shows the receiver operating characteristic (ROC) curve for the candidate selection model at various choices of the threshold. We can observe that a recall of 0.85 can be achieved with a false positive rate close to 0.65. Note that false positive rate is not a significant concern since the false positives can be eliminated as part of the scoring and ranking performed by the second pass ranking model. Further, note that our goal is to retrieve the top-k relevant results for each query, rather than to ensure high recall. For the online experiments, we selected four candidate selection models (varying just in the threshold) corresponding to recall values of 0.85, 0.90, 0.95, and 0.99 respectively, along with the baseline model with no candidate selection.

We measured the efficiency of the candidate selection models with respect to online production traffic. To ensure fair comparison, we allocated the same fraction of users (randomly partitioned)

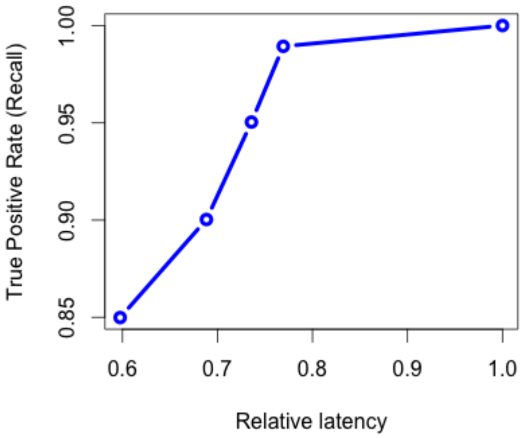

Figure 4: Effectiveness (recall) vs efficiency (relative 90th percentile latency).

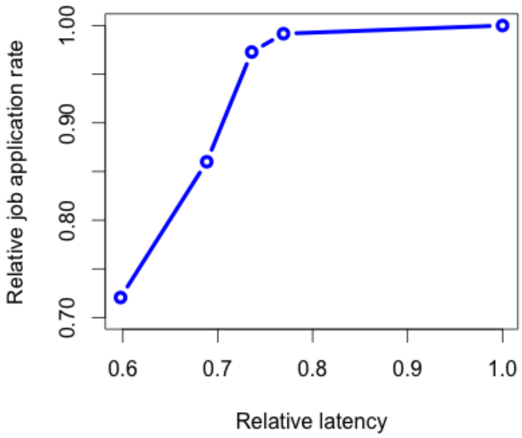

Figure 5: Effectiveness (relative job application rate) vs efficiency (relative 90th percentile latency).

to each of the five models above, so that each model is subject to the same amount of live traffic load. Figure 4 shows the plot of recall vs. 90th percentile latency for the four candidate selection models, relative to those of the baseline model. The top right data point (with relative recall and latency both equal to 1) represents the baseline model. As we increase the threshold parameter of the model, fewer documents are selected, thereby resulting in improved query processing time (latency) but reduced recall. However, we observe that the introduction of the candidate selection model with (a threshold corresponding to) recall value of 0.99 reduces latency by more than 20%. Choosing the models with recall values of 0.95 and 0.90 reduce latency by more than 25% and 30% respectively. These results demonstrate the potential to significantly reduce latency without sacrificing quality.

We next present the trade-off between effectiveness and efficiency for the candidate selection models, focusing on the metrics that directly impact user experience and business objectives. Figure 5 shows the plot of the job application rate vs. 90th percentile latency for the four candidate selection models, relative to those of the baseline model (the top right data point in the plot). We observe that the best balance can be achieved by choosing the candidate selection model that achieves more than 25% latency reduction with less than 3% drop in the job application rate. Comparing Figures 4 and 5, we observe that a small reduction in (offline) recall does not translate to as much drop in the (online) job application rate. On the other hand, a more aggressive candidate selection model (with slightly larger reduction in recall) has a disproportionately worse job application rate when deployed online.

6.4 Effect of Varying Training Data and Choice of Features

We experimented with two methods for generating training data from the user-job interaction logs.

- Type 1. Jobs with clicks + random negatives
  - Positive labels are assigned to (user u, job j) pairs where user u clicked on job j.
  - Negative labels are assigned to (user u, job j) pairs where j is a randomly sampled job that was never shown to u.
- Type 2. Job impressions + random negatives
  - Positive labels are assigned to (user u, job j) pairs where job j is part of the top recommendation results (referred as "impressions") shown to user u using the baseline model.
  - Negative labels are assigned to (user u, job j) pairs where j is a randomly sampled job that was never shown to u.

The intuition underlying type 1 training data is that we want to directly optimize for retrieving jobs that users are more likely to click. Type 2 training data is based on the premise that it is sufficient to optimize for retrieving jobs that would be returned and ranked at the top by the baseline model. The underlying assumption is that this task should be easier than predicting jobs that will get clicked, which would instead be taken care of by the more complicated second pass scoring and ranking model.

Table 2 presents the effectiveness (in terms of ROC AUC and PR AUC) of the models trained with Type 1 and Type 2 training data respectively. We observed better performance using Type 2 training data compared to Type 1 training data (ROC AUC of 0.775 vs. 0.742). Type 2 training data approach also resulted in better online A/B testing performance. Hence, we chose this approach for reporting other experimental results, and also in the deployed production system.

We also measured the impact of using only features with positive coefficient weights in the candidate selection model. From the last two rows in Table 2, we notice that the offline effectiveness (ROC AUC) with this restriction is 0.775, compared to 0.812 when using all features. However, this restriction is needed for the optimization techniques performed by Galene as part of WAND query execution (see §5.1).

Table 2: Dependence of candidate selection model performance on the training data and choice of features.

Method	ROC AUC	PR AUC
Type 1: Clicks + random negatives + only features with positive coefficients	0.742	0.927
Type 2: Impressions + random negatives + only features with positive coefficients	0.775	0.933
Type 2: Impressions + random negatives + all features with positive and negative coefficients	0.812	0.943

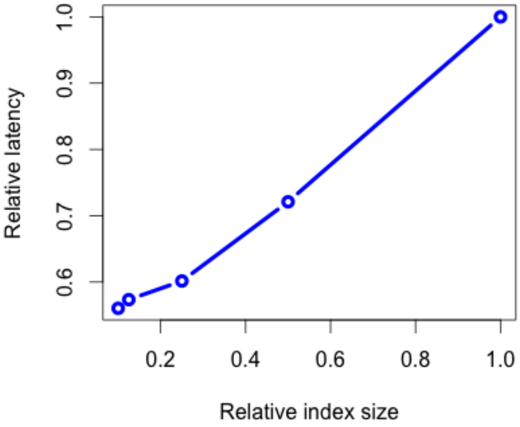

Figure 6: Efficiency (relative 90th percentile latency) vs relative index size.

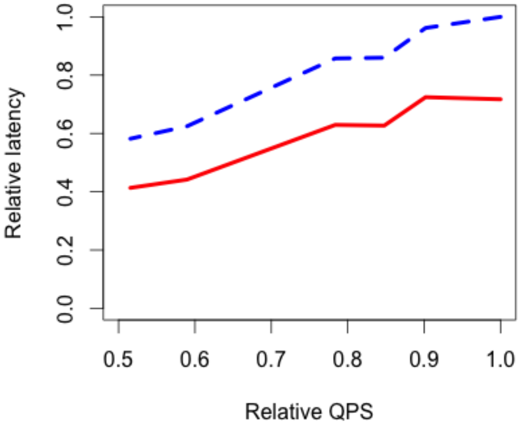

Figure 7: Efficiency (relative 90th percentile latency) vs. relative QPS. Dashed blue line represents baseline, red line represents learned candidate selection model.

6.5 Effect of Varying Index Sizes and Traffic Loads

We also investigated the scalability of our framework by varying index sizes and traffic loads and measuring the impact on the efficiency (latency) in the online system. For these experiments, we used the candidate selection model with (offline) recall value of 0.90.

Figure 6 shows the plot of relative latency vs. relative index size. We observed roughly linear increase in latency as the index size was increased (keeping other parameters such as the number of machines and the traffic load fixed). Since the inverted index is distributed across multiple machines in our architecture, the problem of growing index size can be mitigated by increasing the number of searcher nodes.

Figure 7 presents the effect of increasing traffic load (QPS) on latency. For both the baseline model (dashed blue line) and the candidate selection model (red line), we observed about 65% to 70% increase in latency when QPS was doubled. In particular, the extent of latency reduction for the candidate selection model compared to the baseline remained about the same, demonstrating that the efficiency gain of our proposed framework is robust under different traffic loads.

7. DEPLOYMENT LESSONS

We have deployed our candidate selection strategies as part of the job recommendation system at LinkedIn. Deployment procedure was incremental, where we rolled out the model in several steps to a progressively larger proportion of LinkedIn users. As the amount of traffic to our online system varies depending on the day of the week (for example, relatively large traffic on Mondays compared to the weekends), we carefully monitored the load on the production system upon any deployment changes and gradually increased the traffic allocation on a weekly basis. We noticed that the freshness of the data was very critical to the relevance quality of our system, and hence scheduled major index updates to occur at least daily.

Our experience also highlights the crucial need to leave a certain percent of traffic (*e.g.*, 5%) served by the baseline model, which has highest recall. This traffic allocation is needed to generate unbiased training data for retraining and refreshing the candidate selection model in the future. Otherwise, false negative results from the current model will never be shown to users and hence cannot serve as positive examples during future model training.

During deployment, we also measured the quality of retrieved results in different user segments. A large fraction of the requests for job recommendations comes from active job seekers on LinkedIn. Typically such users tend to have richer and more complete profiles. As a result, the constructed Galene query associated with an active user typically is longer, and consists of Boolean clauses over a significantly large number of literals (item field/value pairs). As discussed in §6, our candidate selection query model favorably helped to serve job recommendations to active users with reduced latency, without sacrificing the quality of the results. At the same time, there are also users on LinkedIn with relatively less rich and complete profiles, for whom we need to serve job recommendations. During initial stages of deployment, we noticed that lesser number of job documents were retrieved for users with lower profile completion when the learned candidate selection model was used, thereby potentially resulting in lower quality of recommendations. Towards better understanding and quantification of this issue, we compared the number of users that are served job recommendation

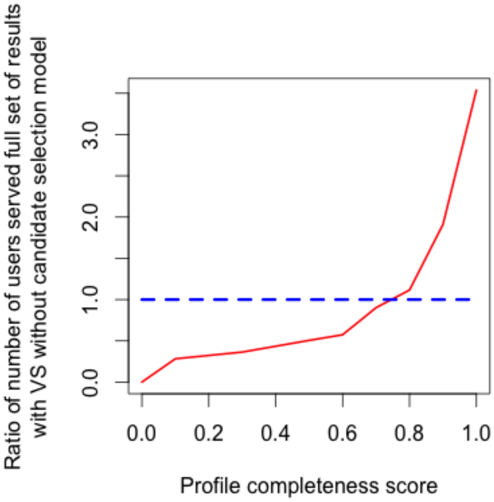

Figure 8: Dependency of ratio of number of users served on profile completeness score.

results with and without learned candidate selection models as a function of a score indicating the extent of richness/completion of a user's profile ('profile completeness score'). More precisely, for each score value, we computed the ratio of (1) the fraction of users within a narrow range of this score that are being served job recommendations using candidate selection model to (2) the fraction of users within a narrow range of this score that are being served job recommendations without candidate selection model. In Figure 8, the X-axis denotes the 'profile completeness score' and the Y-axis denotes the above ratio. We observe that for users with relatively low profile completeness score, the candidate selection based model resulted in a reduced chance of getting a full set of job recommendations (represented as the region where the red line is below the dashed blue line). An explanation is that our initial model was trained with data mostly from active users with relatively high profile completeness score.

Our solution to address this problem was to target users differently depending on their profile completeness score, by using different candidate selection models and/or lowering threshold for the candidate selection model, and in particular, to train a less strict candidate selection model for users with low profile completeness scores.

8. CONCLUSION

Motivated by the requirements of real-time personalized search and recommendation systems at LinkedIn, we studied the problem of building candidate selection models over structured queries and documents. We proposed *CaSMoS*, a machine learned candidate selection framework that makes use of Weighted AND (WAND) query. We developed and deployed our framework as part of the production system that powers LinkedIn's job recommendation engine, resulting in significant reduction in latency (up to 25%) without sacrificing the quality of the retrieved results. We showed the efficacy of the proposed framework through extensive experiments on real datasets using offline metrics as well as online A/B testing evaluation. We also presented the design decisions and tradeoffs faced while building and evaluating our framework, and highlighted the lessons learned through the production deployment of our system.

9. ACKNOWLEDGMENTS

The authors would like to thank Parul Jain, Kaushik Rangadurai, and Lance Wall for their help in the implementation and deployment of our system as part of LinkedIn's job recommendation engine, and David Hardtke and Liang Zhang for insightful feedback.

10. REFERENCES

[1] Apache Kafka. http://kafka.apache.org/.

[2] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*. Springer, 2015.

[3] E. Al Mashagba, F. Al Mashagba, and M. O. Nassar. Query optimization using genetic algorithms in the vector space model. *International Journal of Computer Science Issues (IJCSI)*, 8(5), 2011.

[4] A. Anagnostopoulos, A. Z. Broder, and K. Punera. Effective and efficient classification on a search-engine model. In *CIKM*, 2006.

[5] Y. Aphinyanaphongs and C. Aliferis. Learning Boolean queries for article quality filtering. In *MEDINFO*, 2004.

[6] N. Asadi and J. Lin. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *SIGIR*, 2013.

[7] M. Bazire and P. Brézillon. Understanding context before using it. In *CONTEXT*, 2005.

[8] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, 2006.

[9] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, 2003.

[10] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS*, 1998.

[11] P. Dourish. What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1), 2004.

[12] G. W. Flake, E. J. Glover, S. Lawrence, and C. L. Giles. Extracting query modifications from nonlinear SVMs. In *WWW*, 2002.

[13] J.-T. Horng and C.-C. Yeh. Applying genetic algorithms to query optimization in document retrieval. *Information processing & management*, 36(5), 2000.

[14] N. McNeill, H. Kardes, and A. Borthwick. Dynamic record blocking: Efficient linking of massive databases in MapReduce. In *QDB*, 2012.

[15] S. Sriram and A. Makhani. LinkedIn's Galene Search engine, 2014. https://engineering.linkedin.com/search/did-you-mean-galene.

[16] N. Tonellotto, C. Macdonald, and I. Ounis. Efficient and effective retrieval using selective pruning. In *WSDM*, 2013.

[17] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin. From infrastructure to culture: A/B testing challenges in large scale social networks. In *KDD*, 2015.