Learning Rewrite Rules for Search Database Systems Using Query Logs

Monu Kedia
IBM Research - India
monu.kedia@in.ibm.com

Dinesh Garg
IBM Research - India
garg.dinesh@in.ibm.com

Sriram Raghavan
IBM Research - India
sriramraghavan@in.ibm.com

ABSTRACT

Recent literature on “search database systems” has introduced the notion of using query rewrite rules to influence the behavior of a search engine. Rewrite rules enable domain experts and search administrators to customize the search engine by providing a powerful rule-driven framework to transform user search queries. In this paper, we address the important problem of automatically learning such query rewrite rules from query logs using a novel Hidden Markov Model (HMM) formulation. Our formulation captures both the latent information present in the sequence of queries issued within a session as well as the explicit information in the form of user clicks on individual results. We have developed the notion of support and confidence for query rewrite rules and leveraged these concepts to harvest high quality rewrite rules from the trained HMM. We propose rigorous evaluation schemes to quantify the efficacy of a set of rewrite rules and demonstrate the effectiveness of our approach through extensive experiments using the publicly available AOL query log data set and the YAGO concept ontology.

Keywords
Rewrite rules, Hidden Markov Model (HMM), Query logs

1. INTRODUCTION

Despite the tremendous growth in the volume and importance of textual content within enterprises, search over enterprise Intranets and document collections has been unable to match the success of Web search engines. A number of reasons have been suggested for this phenomenon. In [13], the author points to the presence of multiple heterogeneous data sources (content systems, Web portals, emails, databases, etc.) and the difficulty of developing ranking functions that work across such sources. In [12, 7, 9], the authors mention the absence of significant link structure, the paucity of anchor text, the lack of economic incentives to make enterprise content searchable, and the presence of unique domain-specific jargon as key barriers to effective enterprise search.

To address some of these enterprise search challenges, a recent work on “search database systems” [10, 11] has introduced two key notions: (i) the idea of using an auxiliary domain ontology, external to the document collection, to convert keyword search queries into structured interpretations, and (ii) the use of rewrite rules to allow search administrators and domain experts to customize or otherwise modify end-user search queries for better relevance. To illustrate these ideas and motivate our work, we present the following example adapted from [11].

Example 1. Consider enterprise search scenario where users are looking for information about employees. Administrators can register rules that fire when users issue queries involving words like “office”, “contact”, “number”, etc. The rules will direct the search engine to augment the incoming query with additional transformed queries that are issued against an internal employee directory system to yield highly relevant results. The actual transformation accomplished by the rules can range from simple application of synonyms (e.g., map “fone” to “phone”) to more complex operations such as mapping keywords to concepts and mapping sets of keywords to relationships. Figure 1 illustrates a small domain ontology involving the types of entities and relationships that are typically present in an enterprise directory (persons, phones, addresses, and relationships amongst these concepts). Figure 2 shows a set of three rewrite rules that leverage this ontology to represent certain query transformations. The first rule in Figure 2 states that when a query consists of the name of a person followed by the word “contact”, the engine should transform the query into a search for the person name against the enterprise directory. The second rule transforms queries involving the name of a person followed by the phrase “office number” into an appropriate search against the peradd relationship that relates persons and their work addresses. Finally, Figure 3 shows...
how actual keyword search queries issued by users are transformed by these rewrite rules.

\[
\text{person} \quad X \text{ contact} \quad \Rightarrow \quad \text{person} \quad X
\]

\[
\text{person} \quad X \text{ Office Number} \quad \Rightarrow \quad \text{person} \quad \text{address} \quad X \quad \text{Office Number}
\]

\[
\text{person} \quad X \text{ Phone Number} \quad \Rightarrow \quad \text{person} \quad \text{phone} \quad X \quad \text{Phone Number}
\]

Figure 2: Rewrite rules

\[
\text{Alice contact} \quad \Rightarrow \quad \text{person} \quad Alice
\]

\[
\text{Alice Office Number} \quad \Rightarrow \quad \text{Alice Office Number}
\]

\[
\text{John Phone Number} \quad \Rightarrow \quad \text{person} \quad \text{phone} \quad John \quad \text{Phone Number}
\]

Figure 3: Rewrite rules

In prior work [10, 11], the authors proposed and formally studied the idea of using rewrite rules over auxiliary domain ontologies to influence a search engine’s behavior. The concept of {	extit{hedge expressions}} (a sequence of ordered directed trees) was introduced to represent the types of structures shown in Figures 2 and 3. Essentially, a hedge expression represents a particular “query template” described in terms of the concepts and relationships in the relevant ontology. A rewrite rule \(A \Rightarrow B\) is conceptually a binary relation over the set of all possible hedge expressions that can be constructed over this ontology. Based on this formalization, the authors in [10, 11] developed efficient hedge expression enumeration algorithms and presented theoretical results on convergence properties of sets of rewrite rules.

The specific contributions of this paper are as follows:

- A novel HMM-based model for capturing the latent user intent present in search logs and mapping the states of the HMM to hedge expressions.
- An extension to the HMM model and associated training procedure to incorporate “click events” (i.e., the event when a search user clicks on one of the search results) in terms of a distribution known as the Click Acquisition Probability (CAP).

- The notions of support and confidence of rewrite rules (expressed in terms of the parameters of the trained HMM) and a set of filtering criteria to mine high quality rewrite rules from the trained HMM.
- An efficient dynamic programming based algorithm to compute confidence scores for rewrite rules.
- Extensive evaluation and validation of our approach using a subset of the publicly available AOL query logs and a manually curated subset of the the YAGO ontology [15].

Note that even though the idea of using rewrite rules was primarily proposed from the viewpoint of building high quality enterprise search systems, we believe the techniques themselves are more broadly applicable. We have therefore chosen to use Web search logs (from AOL) and a generic ontology (such as YAGO) to demonstrate our technique, rather than using query logs and ontologies from specific enterprise search deployments. In addition, we have made the entire data set used for our evaluation publicly available for use by other researchers - https://sites.google.com/site/cikm2012dataset/.

2. RELATED WORK

The idea of using rewrite rules for domain-specific customization of search system was originally proposed by Fagin et al [11] as part of their work on search database systems for enterprise search. The primary focus of their work was on developing a formal framework to describe rewrite rules and study their convergence properties. The goal of our work is to develop an automated technique that leverages the user intent implicitly captured in search logs to assist domain experts and search administrators in crafting effective rewrite rules. Recently, Agarwal et al [1] have proposed an approach to automatically discover a library of query templates by analyzing query logs. While their notion of query templates is similar in spirit to the notion of hedge expressions used in this work, their approach does not involve finding pairs of hedge expressions that yield rewrite rules.

In addition to search database systems, there are three broad areas of work that relate to the work described in this paper: (1) learning to rank, (2) query suggestions, and (3) query rewrites for advertising.

2.1 Learning to Rank Search results

There is a large body of work on applying supervised or semi-supervised machine learning techniques to automatically learn search ranking algorithms, using information from search logs as training data [20, 5, 16, 4].

2.2 Query suggestions

Prior work in the area of automatic query recommendation includes techniques such as query expansion [22], query substitution [17], query clustering using query logs [3, 24], template based techniques for long-tail queries [23], and term-query graph based approach for long-tail queries [6].
2.3 Advertising

Query rewrites are used in advertising to reformulate user’s queries to facilitate better matching and identification of relevant ads from the available inventory ([26],[25],[21],[2]). From this literature, the model proposed by Cao et al [8] has commonalities with the HMM-based formulation proposed in this paper. However, unlike our work, their primary focus is on the scalability issues of the model rather than on the use of the HMM for learning rewrite rules.

3. HIDDEN MARKOV MODEL FOR RULE LEARNING

As mentioned in the introduction, our overall approach is based on building a first order HMM to model user intent and using a mapping from the HMM states to the hedge expressions to mine rewrite rules from the trained model. Table 1 summarizes the correspondence between different quantities in a typical search process and quantities in a typical HMM model.

<table>
<thead>
<tr>
<th>Search Process</th>
<th>HMM Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>User’s Search Intent</td>
<td>Hidden State</td>
</tr>
<tr>
<td>Search Query</td>
<td>Observation</td>
</tr>
<tr>
<td>Search Session</td>
<td>A Sequence of Observations</td>
</tr>
</tbody>
</table>

Table 1: Correspondence between search process and HMM model

To formalize our model, we use following notations.

- \( S = \{S_1, S_2, \ldots, S_N\} \) denotes the set of \( N \) hidden states of the model where each state corresponds to one possible search intent of a user.
- \( Q = \{Q_1, Q_2, \ldots, Q_M\} \) denotes the set of \( M \) observations where each observation corresponds to a search query fired by a user.
- \( A = \{a_{ij}\} \) is a HMM state transition probability matrix (aka state transition matrix).
- \( \pi = \{\pi_i\} \) denotes the initial state probability distribution.

We further augment this HMM by introducing the notion of a click event at each query observation. For every observation, a click event happens when the user clicks on one of the search results produced for the corresponding query. Inclusion of this event makes the augmented HMM a triply stochastic process where each observation is now a pair (query, click event). In other words, an observed query \( Q \) now becomes either \((Q,0)\) or \((Q,1)\) based on the presence or absence of a corresponding click event. To model click events, we introduce a probability distribution called click acquisition probability (CAP). The CAP distribution, denoted by \( C = \{c_i|Q_j|\} \), \( 1 \leq i \leq N \) and \( 1 \leq j \leq M \), is defined as

\[
c_i(Q_j) = P[\text{Click} | \text{Observing } Q_j \text{ at } t \text{ and } X_t = S_i]\]

Thus, our augmented HMM is completely described by the 5-tuple

\[
\lambda = (S, A, B, C, \pi)
\]

In what follows, we discuss how to train such an HMM from a search query log and develop a novel dynamic programming algorithm to extract high quality rewrite rules from this trained HMM.

3.1 Training HMM Using Query Logs

To train the HMM, we assume that we have \( L \) training sessions \( O = \{O^{(1)}, O^{(2)}, \ldots, O^{(L)}\} \), where each training session \( O^{(k)} \) is a sequence of queries with associated click events. For a single training session \( O^{(k)} = \{Q_{1(k)}, Q_{2(k)}, \ldots, Q_{M(k)}\} \), the task of training the HMM is a widely studied topic with well known algorithms such as the Baum-Welch algorithm.\(^1\)

However, since we have multiple training sessions (and thus multiple observation sequences available), we leverage the extension to the Baum-Welch algorithm proposed by Li et al [19] for learning \( A, B, \) and \( \pi \) of our HMM.

The remaining parameter of our augmented HMM is the CAP distribution \( C \). We will estimate this distribution at the level of the individual states of the HMM, i.e., \( C = \{c_i\}, 1 \leq i \leq N \), where

\[
c_i = P[\text{click} | X_t = S_i] = \sum_{j=1}^{M} c_i(Q_j)b_i(Q_j)
\]

The quantity \( b_i(Q_j) \) will be estimated as part of normal HMM training and \( c_i(Q_j) \) can be computed through simple frequency counts over the training data.

3.2 The Problem of Inferring Rewrite Rules

In our HMM framework, a rewrite rule is any statement of the form \( S_i \Rightarrow S_j \) where \( S_i, S_j \in S \) (\( S_i \) and \( S_j \) are respectively the antecedent and consequent of this rewrite rule). To identify high quality rewrite rules from the set of all possible rewrite rules, we develop a set of scoring metrics based on the notions of support and confidence of a rewrite rule.

**Definition 1** (Confidence). Let \( \lambda = (S,A,B,C,\pi) \) be an augmented HMM and let \( \gamma \) be any sequence of states of length 0 or more. The confidence score \( \sigma(S_i \Rightarrow S_j) \) of the rewrite rule \( S_i \Rightarrow S_j \) is defined as

\[
\sigma(S_i \Rightarrow S_j) = \text{maximize}_{\gamma} \ P[S_i \gamma S_j | \lambda]
\]

Thus, the confidence score of a rewrite rule is the highest probability of transitioning from the antecedent of a rewrite rule to its consequent, potentially through a sequence of intermediate states. The path \( S^*_i \) that yields this highest probability is called the optimal sequence for the rewrite rule \( S_i \Rightarrow S_j \). Thus,

\[
S^*_i = \text{argmax}_{\gamma} \ P[S_i \gamma S_j | \lambda]
\]

\(^1\)In our implementation, we used the publicly available UMDHMM implementation [18].
sequence of states of length 0 or more. The CAP-based confidence score of a rewrite rule \(S_i \Rightarrow S_j\) is defined as

\[
\sigma^\gamma(S_i \Rightarrow S_j) = \maximize_{\gamma} P[S_i \gamma S_j | \lambda](1 - c_i) \prod_{k=1}^{l}(1 - c_{(k)})
\]

where \(c_i\) is as defined in Equation (2).

Essentially, the extra multiplicative term \((1 - c_i)c_j \prod_{k=1}^{l}(1 - c_{(k)})\) represents the probability of no click events anywhere in the transition sequence except at the very end in state \(S_j\). The path \(S_i^\gamma\) that yields this highest probability is known as the CAP-optimal sequence for the rewrite rule \(S_i \Rightarrow S_j\).

To define the support of a rewrite rule, we first define the support of an HMM edge \(\langle S_{(i)}, S_{(j)} \rangle\) as the number of times a pair of adjacent queries \(\langle Q_x, Q_y \rangle\) appear in query logs such that \(Q_x\) and \(Q_y\) correspond to the hidden states \(S_{(i)}\) and \(S_{(j)}\), respectively. The following definitions now follow naturally:

**Definition 3 (Support).** The support of a rewrite rule \(S_i \Rightarrow S_j\), denoted by \(\text{supp}(S_i \Rightarrow S_j)\), is the minimum of the edge supports across all the edges of the optimal sequences of transitions from \(S_i\) to \(S_j\), given by \(S_i S_i^* S_j\).

**Definition 4 (CAP-based Support).** The CAP-based support of a rewrite rule \(S_i \Rightarrow S_j\), denoted by \(\text{supp}^\gamma(S_i \Rightarrow S_j)\), is the minimum of the edge supports across all the edges of the CAP-optimal sequence of transitions from \(S_i\) to \(S_j\), given by \(S_i S_i^* S_j\).

The definitions of support and confidence along with their CAP-based variants yield four natural filtering strategies for identifying high quality rewrite rules:

1. **Support Filter Confidence Rank (SFCR):** Filter out all rewrite rules for which the support is lower than some threshold \(\tau_s\) and pick the top-\(k\) rules from the survivors based on their confidence score.
2. **Support Filter Confidence Rank with CAP (SFCR-CAP):** Variant of SFCR that uses the corresponding CAP-based variants for support and confidence.
3. **Confidence Filter Support Rank (CFSR):** Analogous to SFCR but with the roles of support and confidence swapped.
4. **Confidence Filter Support Rank with CAP (CFSR-CAP):** Variant of CFSR that uses the corresponding CAP-based variants for support and confidence.

To make these filtering strategies practical, we require an efficient algorithm to compute the confidence scores and optimal sequences for any pair of states in a trained HMM (note that support values can be computed efficiently once the optimal sequences are known). The following section addresses this problem.

### 3.3 Computing Confidence Scores

We present a dynamic programming based algorithm to compute an optimal sequence for every possible rewrite rule \(S_i \Rightarrow S_j\). We first state (without proof) the following lemma

**Lemma 1.** Let \(S_i^* = \langle S_{(1)}, S_{(2)}, \ldots, S_{(l)} \rangle\) be the optimal sequence for the rewrite rule \(S_i \Rightarrow S_j\). Let \(S_x\) and \(S_y\) be two different states in the following sequence \(S_i S_i^* S_j = \langle S_i, S_{(1)}, \ldots, S_{(l)}, S_y \rangle\) such that \(S_x\) appears before \(S_y\). Then, the subsequence \(S_x^*\) must be the optimal sequence for the rewrite rule \(S_x \Rightarrow S_y\). In other words,

\[
S_x = S_x^* = \argmax_S P[S_x S_y | \lambda]
\]

From the lemma, it is apparent that the problem of determining optimal sequences exhibits a structure that is amenable to a dynamic programming approach. However, rather than developing a custom algorithm for this task, we first show that the problem of computing optimal sequences can be reduced to the all pairs shortest path problem for which well known dynamic programming algorithms such as the Floyd-Warshall or Johnson algorithm are available. We will first show this reduction for non-CAP confidence scores and then extend the reduction to incorporate CAP-based confidence.

Consider Expression (3) that defines the confidence score of a rewrite rule. It is easy to see that

\[
\sigma(S_i \Rightarrow S_j) = \maximize_S P[S_i S_j | \lambda] = \maximize_{S=(S_{(1)}, S_{(2)}, \ldots, S_{(l)})} a_{i(1)} a_{j(l)} \prod_{k=1}^{l} a_{(k)(k+1)}
\]

where Equation (7) follows from the Markov property. The maximization problem in equation (7) can be reformulated as minimization of the log-likelihood, thus yielding:

\[
\minimize_{\langle S_{(1)}, S_{(2)}, \ldots S_{(l)} \rangle} \left[ \log(a_{i(1)}) + \log(a_{j(l)}) + \sum_{k=1}^{l} \log(a_{(k)(k+1)}) \right]
\]

Expression (8) has the following natural graph theoretic interpretation: a directed graph \(G = (S, E)\) with a vertex set corresponding to the states of the Markov chain and a directed arc from every vertex \(S_i\) to every other vertex \(S_j\) with an arc weight of \(-\log(a_{ij})\). The log-likelihood minimization problem (8) is equivalent to the task of finding a shortest directed path from vertex \(S_i\) to \(S_j\) in graph \(G\). To accommodate CAP-based confidence scores, note that the extra multiplicative terms in Equation (5) can be easily accommodated by modifying the arc weights to \(-\log(a_{ij}) - \log(1 - c_i)\) instead of \(-\log(a_{ij})\).

### 4. IMPLEMENTATION AND RESULTS

#### 4.1 Implementation Details

Figure 4 shows the high level flow of the rule learning system we have implemented and evaluated. Below, we discuss each step of this flow in detail, followed by a discussion of the evaluation schemes and experimental results. All our results are based on a 0.5 million size subset of the publicly available AOL query logs (the overall log includes about 20 million web queries from 0.65 million users). Each entry in this query log is a 5-tuple \((u, q, t, cl, p)\) consisting of Anonymous user identifier, Query issued by the user, Time

\[\text{The additional term} \ - \log(c_j) \ \text{that would be present if we directly took the logarithm of the expression in Equation (5) does not affect the all pairs shortest path computation as it is a constant for a given destination vertex.}\]
Figure 4: Experimental setup for evaluation of proposed approach to learn rewrite rules

4.1.1 Concept Identification

Our first step involves tagging each query log entry with one or more concepts from YAGO [15] ontology. To accomplish this, we process the log entries one at a time. For each entry, we view the query string $q$ as a set of query tokens $\{w_1, w_2, \ldots, w_n\}$. For each possible sub-sequence from this token list, we lookup the YAGO ontology and record the matching concepts. While the YAGO ontology includes concepts from the Wikipedia categories along with the WordNet hierarchy, we observed that the WordNet concepts were too broad and high level to be of use in generating rules. We therefore retained only the Wikipedia categories. Furthermore, given two sub-sequence of tokens from $q$ where one sub-sequence is strictly contained within another, the set of concepts from the smaller sequence were filtered out as long as there was at least one concept matching the larger sequence. Let $T(e)$ denote the set of all concepts associated, through this process, with a log entry $e$.

4.1.2 Discovery of Session Boundaries

Our sessionization procedure consists of the following steps:

1. Group the log entries based on their user identifier field to produce per-user log sequences.
2. Sort each sequence in ascending order of the timestamp field.
3. In a single pass over each user’s query log, identify a session boundary between two successive entries $e_1$ and $e_2$, if
   
   (a) $t_1 - t_2 \geq \Delta$, where $t_1$ and $t_2$ are the time stamp fields for entries $e_1$ and $e_2$, respectively. In our experiments, we have set $\Delta$ to 12 minutes, as recommended in [14].
   
   (b) $J_0(T(e_1), T(e_2)) \geq \Theta$, where $J_0(\cdot)$ is the Jaccard distance between two sets. Logically, we assume that if there is significant difference between the concept sets for two different queries, the user has likely moved onto a different information need and we must therefore view the previous session as having terminated. Based on some initial experiments and manual verification of session quality, we set $\Theta$ to 0.4.

4.1.3 Domain Identification and Ontology Refinements

It is natural for a generic web query log to span queries from heterogeneous domains. Mining rewrite rules using a single HMM across query sessions spanning many different domains is not meaningful as it would involve working with too many concepts which would in turn lead to numerous hedge expressions (i.e., a large number of HMM states) and a corresponding data sparsity problem. A sequence of cleansing and filtering steps were therefore essential to identify a more manageable domain ontology and corresponding query set that could be used to train the HMM. To drive this cleansing process, we first chose a subset of the query logs consisting of 0.2 million queries and attempted to identify the high level topics involved in those sessions. To this end, we associated each query session with a concept set by combining the concept sets for all the queries in the session (i.e., the union of all the $T(e)$ sets within each session). Using such concept sets as features, we performed standard k-means clustering and after some experimentation with cluster parameters, identified two major cohesive topics with significant membership: Media and Entertainment and Literature.

Even after restricting our focus only to the concepts present in these two clusters, we were still left with a relatively large set of 7216 concepts, mainly due to the fine granularity concepts returned by YAGO (e.g., numerous concepts such as wikicategory_1845_poems, wikicategory_1849_poems etc.). A series of manual and semi-automatic heuristics were applied to this concept set to merge related concepts (e.g., create a single high level category called poems) and finally arrive at a set of 629 high-level concepts that formed the ontology for our learning process. Finally, for our subsequent experiments, we retained only those query sessions in which each query in the session was related to at least one concept in our cleansed and refined ontology.

4.1.4 HMM Training

Armed with a cleansed ontology and a corresponding set of query sessions, we are now ready to train the HMM $(S, A, B, C, \pi)$ described in Section 3.

Recall that $S$, the set of states of our HMM, is simply the set of all possible hedge expressions. To generate this set, we make a single pass through all the query sessions and for each query, generate all the possible hedge expressions that correspond to that query. Note, with a flat ontology, a hedge expression is merely a sequence of interleaved concepts and tokens. Thus, a specific hedge expression for a query $q = \langle w_1, w_2, \ldots, w_n \rangle$ with $n$ tokens can be generated by replacing one or more non-overlapping subsequence of tokens with a corresponding concept (e.g., a possible hedge expression for query $q$ is $\langle \gamma_1, \gamma_2, \ldots, w_k, w_{k+1}, \ldots, w_m, \gamma_{m+1}, \ldots, \gamma_n \rangle$, where $\gamma_i$’s are concepts). After this exhaustive enumeration process, to further reduce issues due to data sparsity, we further pruned the state space by retaining an HMM state only if:

3In the interest of space, we do not delve into the details of this ontology refinement process.
and (ii) there were at least five queries that yielded this hedge expression. Intuitively, we wish to retain rules only if there is enough critical mass of evidence in the query logs to believe that the state could yield rewrite rules.

The transition probability matrix $A$ and the state probability $\pi$ are both initialized to uniform distributions. To initialize emission probabilities (distribution $B$), for each hidden state $S_i$, we adopt the following smoothing strategy: we uniformly distribute 0.1 probability mass across all those queries which do not match the hedge expression corresponding to $S_i$. The remaining 0.9 probability mass is distributed across queries that match the hedge expression, in proportion to the number of times the queries occurred in the the training data.

### 4.1.5 Rule Inference

All our experiments on rule inference are reported on a training data set of 0.3 million queries (7223 sessions) and a test data set of 0.2 million queries (4817 sessions). Using the filtering criterion described in the previous section, our training data resulted in an HMM with 688 states. We used all four strategies (SFCR, SFCR-CAP, CFSR and CFSR-CAP) to mine rewrite rules. We observed that the rules produced by our techniques broadly fall under four categories. Below, we describe each category of rules along with actual anecdotal examples (in the following, let $A$ be the set of concepts and $x$ be the set of all query tokens):

1. **$A \Rightarrow Ax$**: Semantically, this is like adding context sensitive keywords to transform the query. An example of this kind of rules is wiki_novelist $\Rightarrow$ wiki_novelist biography, where “wiki_novelist” is the concept and “biography” is the context sensitive keyword added by the application of this rule.

2. **$Ax \Rightarrow A$**: Semantically, this is like dropping context sensitive noise words. An example of this kind of rules is wiki_songs lyrics $\Rightarrow$ wiki_songs, where “wiki_songs” is a concept and “lyrics” is the keyword.

3. **$Ax \Rightarrow Ay$**: This is an interesting class of rules. This is similar to synonym expansion, but “$x’$” may not necessarily be the synonym of “$y$”. A good example of this class of rule is wiki_actors galleries $\Rightarrow$ wiki_actors links, where “wiki_actors” is a concept, and “galleries” and “links” are keywords but not synonym of each other.

4. **$Ax \Rightarrow Ax’$**: This is basically spell correction which has little semantic relevance. Although, mined rules do contain examples from this class but there are more established approaches in the literature to accomplish this task. So, we don’t consider this class in “interesting” rules.

### 4.2 Evaluation Framework

To the best of our knowledge, this is the first attempt to automatically discover query rewrite rules in the context of a search database system. As such, there are no existing evaluation schemes for measuring the quality of the extracted rewrite rules. Obviously, the real long term impact of such rules is best measured by comparing the overall improvement of search quality over an extended period of time before and after the rules are deployed. However, we anticipate that a system such as ours would be used in conjunction with a human expert who would examine and judge the recommendations from our system before finally deploying it against a search system. Therefore, the evaluation schemes described below are geared towards measuring how well the metrics (SFCR, SFCR-CAP, CFSR and CFSR-CAP) perform with respect to identifying and prioritizing the most useful rules to present to such an expert.

#### 4.2.1 Reduction in Query Rewrites (QRR score)

This scheme measures the total reduction in query rewrites over the test data using learned rules. Formally, suppose $R$ is the set of rewrite rules output by some learning algorithm. Let $O^{(k)} = \{Q_1^{(k)}, Q_2^{(k)}, \ldots, Q_k^{(k)}\}$ be some user session in the test data. For every rule $R \in R$, and for every query $Q_j^{(k)}$ in the user session, we compute a non-negative integer score, namely $QRR(Q_j^{(k)} \mid R, O^{(k)})$ score. This score essentially measures the effect of having the rule $R$ applied to the query $Q_j^{(k)}$ in the same session $O^{(k)}$. If the use of a rewrite rule $R$ on the query $Q_j^{(k)}$ would have resulted in a query that is present downstream in the same session $O^{(k)}$, this metric measures the savings in user burden as $1 + \text{number of intervening queries}$. The overall QRR score at a session level is defined as follows.

$$QRR(O^{(k)}) = \sum_{R \in R, Q_j^{(k)} \in O^{(k)}} QRR(Q_j^{(k)} \mid R, O^{(k)})$$

The quality of the set $R$ is defined as the sum of QRR scores over all the sessions. Intuitively, this means that by incorporating the learned rules in the system, we would save QRR rewrites. Logically, it is a measure of the reduction in burden of the users to get their information need satisfied by the search system.

#### 4.2.2 Clicks on Query Rewrites (CLC score)

This scheme is designed with the idea that if the rules are frequently generating queries that are being clicked in the session then the rules are performing well. Formally, CLC score is defined as total number of clicks earned by the queries that are rewritten by a rule set $R$ over all the sessions in the test data.

### 4.3 Evaluation Results

As discussed earlier, a rule $A \Rightarrow B$ is defined by a pair of HMM states. We observe that all such pairs cannot result in valid rules. A rule $A \Rightarrow B$ is valid only if the set of concepts present in the consequent state $B$ is a subset of the set of concepts present in the antecedent state $A$. The reason behind this is simple - when we apply a rewrite rule to a given query, we need to bind every concepts in the consequent state present downstream in the given query in order to generate the rewritten query. If consequent of the rule contains any additional concept then the binding for such concept is unknown and rewritten query cannot be generated.

For our evaluation purpose, we let $R$ denote the set of all valid rules that can be obtained using our 0.3 million size training data of AOL log. In our setting, we had 10368 valid rules. Further, we let $QHR(R)$ and $CLC(R)$ denote the QRR and the CLC scores, respectively for the rule set $R$ when evaluated on our 0.2 million size test data of AOL log. Now we apply four different filtering strategies, namely, SFCR, SFCR-CAP, CFSR and CFSR-CAP, on the rule set
and extract out a smaller size subset of the high quality rules. For each of these filtering strategy, Table 2 reports the QRR and the CLC scores of the filtered rules set relative to the original rule set \( R \) (that is, \( QRR(R) \) and \( CLC(R) \), respectively). The rows in the table correspond to the different filtering strategies. Recall, \( \tau_s \) is a threshold parameter for the strategies CFSR and CFSR-CAP; and \( \tau_c \) is a threshold parameter for the strategies SFCR and SFCR-CAP. We have experimented with two different values for each of these threshold parameters, namely \( \tau_s = \{0, 2\} \) and \( \tau_c = \{0, 0.5\} \). The column number 2 through 5, in Table 2, correspond to the relative QRR and CLC scores of the filtered rule set under each of these filtering strategies. The second and the third column corresponds to the scenario when we have \( \tau_s = 0 \) and \( \tau_c = 0 \). Under this scenario, all valid rules meet the threshold limit during the first phase of any of these four filtering strategies. In the subsequent stage, we pick top 25\% rules. The fourth and the fifth column correspond to the scenario when we have \( \tau_s = 2 \) and \( \tau_c = 0.5 \). Under this scenario, a large number of rules get filtered out in the first phase itself for any of these four strategies. In the subsequent phase of each of these strategies, we do not perform any further filtering and instead just pick all the rules. The number of selected rules are being reported (in bracket) along with each result entry in the table.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Threshold Parameters</th>
<th>( \frac{\tau_s \geq 0, \tau_c \geq 0}{% QRR, % CLC} )</th>
<th>( \frac{\tau_s \geq 2, \tau_c \geq 0.5}{% QRR, % CLC} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFCR</td>
<td>33% (2592)</td>
<td>37% (126)</td>
<td>45% (126)</td>
</tr>
<tr>
<td>SFCR-CAP</td>
<td>27% (2592)</td>
<td>41% (126)</td>
<td>50% (126)</td>
</tr>
<tr>
<td>CFSR</td>
<td>66% (2592)</td>
<td>7% (94)</td>
<td>8% (94)</td>
</tr>
<tr>
<td>CFSR-CAP</td>
<td>70% (2592)</td>
<td>0.6% (8)</td>
<td>0.4% (8)</td>
</tr>
</tbody>
</table>

Table 2: Evaluation Results

4.3.1 Discussion

Several interesting observations can be made from the results.

1. The percentage numbers in the Table 2 measure the efficacy of the filtered rule set under various schemes vis-à-vis efficacy of all valid rules which get generated by our trained HMM.

2. The interesting observation here is that by choosing an appropriate non-zero value for the threshold parameter \( \tau_s \), one can reduce the rule set size significantly without any degradation in the performance, especially under the strategies SFCR and SFCR-CAP. For example, by setting \( \tau_s = 2 \), SFCR-CAP strategy gives just 126 high quality rules whose QRR and CLC scores are 51\% and 72\% higher relative to the corresponding scores for some other set of 2592 rules which were obtained by setting \( \tau_s = 0 \). Further, comparing this with SFCR reveals the importance of click acquisition probabilities in our model. This suggests that if the value of the parameter \( \tau_s \) is chosen in an appropriate manner, the SFCR-CAP filtering strategy could turn out to be extremely effective.

3. For \( \tau_s = 0.5 \) in our experiments, the strategies CFSR and CFSR-CAP do not seem to be performing very well. This suggests that a lots of high quality rules got filtered out in the first phase itself when we filtered based on the confidence scores of the rules. This suggests that the confidence of a rule is very sensitive parameter when it comes to identifying high quality rules and hence the support becomes an important complementary metric.

4. In practice, the search administrator typically does not embed all the valid rules into the system due to the overheads resulting by applying these rules to the queries at the run time and also the presence of noise in the learned rules. This is precisely where our filtering strategy such as SFCR-CAP can help him pick a desired size subset of very high quality rules that would meet the specific requirements of improving system’s performance.

5. Finally, when we eye-balled the rule set output by the SFCR-CAP strategy, we found several semantically sensible rules as shown in the anecdotal evidence section below.

4.3.2 Anecdotal Evidence

Below is a list of top 10 quality learned rules in our experiments which we picked based on eye-balling the results of SFCR-CAP strategy.

1. \( \text{wiki}_\text{novelists} \Rightarrow \text{wiki}_\text{novelists biography} \)
2. \( \text{wiki}_\text{singers} \Rightarrow \text{wiki}_\text{singers pics} \)
3. \( \text{wiki}_\text{musical_groups} \Rightarrow \text{wiki}_\text{musical_groups lyrics} \)
4. \( \text{wiki}_\text{companies} \Rightarrow \text{wiki}_\text{companies jobs} \)
5. \( \text{wiki}_\text{films} \Rightarrow \text{wiki}_\text{films movie} \)
6. \( \text{wiki}_\text{albums} \Rightarrow \text{wiki}_\text{albums} \)
7. \( \text{wiki}_\text{country_singers video clips} \Rightarrow \text{wiki}_\text{country_singers clips} \)
8. \( \text{wiki}_\text{actors pictures} \Rightarrow \text{wiki}_\text{actors on tv} \)
9. \( \text{wiki}_\text{female_models nude} \Rightarrow \text{wiki}_\text{female_models naked} \)
10. \( \text{wiki}_\text{music_groups lyrics} \Rightarrow \text{wiki}_\text{music_groups lyrics} \)

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have addressed the problem of automatically mining rewrite rules from historical search logs. The solution to this problem can significantly reduce the manual efforts involved in rule authoring. We have formulated this problem as a Hidden Markov Model where hidden states of the Markov chain correspond to the latent search intents of the users and the observations correspond to the queries fired by them. We believe that this is natural and first-of-a-kind modeling framework for this problem. Building on the existing body of rich literature, we have also described a method to train the resulting HMM. The other novelty of this paper lies in proposing the notions of support and confidence of rewrite rules (by expressing them in terms of
the parameters of the trained HMM) and a set of filtering criterion to mine high quality rewrite rules from the HMM.

For this, we have proposed four different scores of a rewrite rules and shown that these scores can be computed by simple algorithms such as all-pair shortest path algorithms. We have further described four different filtering strategies, namely SFCR, SFCR-CAP, CFSSR, and CFSSR-CAP, to output a set of good quality rewrite rules. These strategies are based on the four different scores of the rewrite rules. Finally, we have proposed two evaluation schemes for measuring the quality of the rewrite rules learned by any algorithm. This paper opens up plenty of avenues for further investigations, the primary one being better evaluation schemes for the learned rules, fast algorithms for training underlying HMM, testing the proposed HMM framework in other problem context such as automatic query suggestion.

6. REFERENCES


