Sequence Clustering and Labeling for Unsupervised Query Intent Discovery

Jackie Chi Kit Cheung*
Department of Computer Science
University of Toronto
Toronto, ON Canada
jcheung@cs.toronto.edu

Xiao Li*
Facebook Inc.
Palo Alto, CA USA
xiaol@fb.com

ABSTRACT

One popular form of semantic search observed in several modern search engines is to recognize query patterns that trigger instant answers or domain-specific search, producing semantically enriched search results. This often requires understanding the query intent in addition to the meaning of the query terms in order to access structured data sources. A major challenge in intent understanding is to construct a domain-dependent schema and to annotate search queries based on such a schema, a process that to date has required much manual annotation effort. We present an unsupervised method for clustering queries with similar intent and for producing a pattern consisting of a sequence of semantic concepts and/or lexical items for each intent. Furthermore, we leverage the discovered intent patterns to automatically annotate a large number of queries beyond those used in clustering. We evaluated our method on 10 selected domains, discovering over 1400 intent patterns and automatically annotating 125K (and potentially many more) queries. We found that over 90% of patterns and 80% of instance annotations tested are judged to be correct by a majority of annotators.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*clustering*

General Terms

Algorithms, Experimentation

Keywords

semantic search, query intent discovery, clustering

*The work was done while both authors were with Microsoft Research, Redmond.

WSDM'12, February 8–12, 2012, Seattle, Washington, USA.

1. INTRODUCTION

In the traditional IR model of search, a search engine is responsible for returning documents relevant to a search query. Ranking algorithms using surface-level word statistics have been developed which have made keyword search the dominant form of web search today. Modern search engines are moving beyond this paradigm by more deeply analyzing the structure and semantics of search queries. Doing so can not only improve ranking results, it can also allow domain-dependent types of semantically enriched search results to be displayed in addition to the standard ranked list of relevant documents. For example, a search query that matches the pattern "[city] weather" would trigger a weather forecast for that city, while one that matches "[movie_title] showtimes [location]" would return an appropriate list of showtimes at cinemas near the specified location. These *instant answers* understand and directly address the information need of the user, obviating the step of clicking into and searching through an external document.

A related trend is the development of structured (or vertical/specialized) search, which allows precise search within a domain. Structured search requires a query understanding component to extract structured information that is matched with a back-end database. A query may be in the form of a keyword search, which is relatively unstructured with free word order, such as "hotel seattle february". Or, a query may be in a more natural language form, as is often the case with virtual assistant applications on mobile devices. These applications take a voice command from a user, such as "Book a reservation at an Italian restaurant for 7:30 tonight", then perform the desired action, or automate a substantial portion of it, such as by filling in an online reservation form.

One key issue for these applications is to connect the surface-level query to a structured data source according to the intent of the user. Doing so requires an intent understanding system, which makes use of a *schema*. We define a schema as consisting of a *domain*, a set of domain-dependent *intents*, and *slots* or *concepts*. Slots are often assumed to be shared among different intents in the same domain.

Given a schema, an intent understanding system consists of the following: (1) domain classification, (2) intent detection, and (3) slot filling. For example, "harry potter showtimes boston" should be classified into the *Movie* domain with the intent *FindShowtimes* and slots including *Title* = "harry potter" and *Location* = "boston". The first component can be viewed as identifying a coarse semantic class of query intent to limit the hypothesis space of the follow-

ing components. Next, intent detection and slot filling aim at finding more fine-grained user intents within the domain of interest and extracting the slots related to that intent. The last two components are tightly coupled and are often modeled jointly.

There have been two major approaches to doing schema discovery to enable such an intent understanding system. The first is to use a grammar-based approach, defining rules in a domain and matching queries to these predefined rules, often based on the schema of the back-end database. While such an approach may work for simple domains and intents such as accessing weather reports, it is difficult to scale up to a larger number of domains. In addition, manually crafting grammar rules imposes a high cognitive load on humans and is often imprecise. Another approach is to use statistical models to classify queries by intent and to label sequences for slot filling. While this eliminates the need to specify rules, it still requires, in addition to the predefined schema, much annotation effort in the form of labeled data to train supervised machine learning models. Although semi-supervised methods [14] have been developed to reduce this cost, a reasonably large amount of labeled data is still needed as "seeds" to bootstrap learning. Furthermore, by relying on a predefined schema, both methods are susceptible to biases which can result in a mismatch to a user's real information need. For example, in the *Products* domain, databases often contain fields like "serial number" which are rarely queried by users or used as search criteria. Human-designed schemata generally do not cover all intents and their associated slots; search engines to date know how to semantically interpret a subset of user requests that are popular or important like "[city] weather", but there are many more that are yet to be understood and answered.

The goal of this work is to discover popular user intents and their associated slots, and to annotate query instances accordingly in a completely unsupervised manner. As in [1], we define *intent* as represented by a *pattern* or *template* consisting of a sequence of semantic concepts or lexical items. Continuing the above example, a pattern that describes "harry potter showtimes boston" would be "[movie title] showtimes [city]", and other queries that would be annotated with the same pattern might include "madagascar 2 showtimes sydney" or "planet of the apes showtimes london". We present a sequence clustering method for finding such clusters of queries with similar intent, and an *intent summarization* method for labeling clusters with such an intent pattern that describes the queries in the cluster. As such, our system not only discovers the patterns and slots in a domain, it also produces annotated instances of each pattern, and allows classification of new queries not used in clustering into the discovered patterns. Figure 1 shows the components of our system.

We evaluate our methods using Mechanical Turk judgments, and find that over 90% of patterns and over 80% of instances within a pattern are correct according to a majority of judges. Our approach discovers patterns that cover up to 20% of the traffic within a domain, and this number can be easily adjusted at the cost of precision. It can thus facilitate structured search, and is an important step toward fully automating query intent understanding.

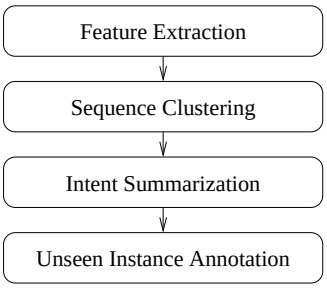

Figure 1: Components of our intent discovery system.

2. RELATED WORK

Query clustering has been applied to various tasks, such as to construct domains of similar documents or to find documents that may be interesting to a user [4], to semi-automatically improve identification of frequent natural language queries [28], or for query refinement and recommendation [2, 3, 24]. In contrast, this work aims to discover common query intent patterns.

There has been work on concept (slot) discovery, which aims to discover entities that belong to the same concept (according to the IS-A relation) and to label these concepts. One body of work discovers concepts by noting that they often appear in the same patterns in context [15, 17], stemming from work on hyponym and relation extraction [10, 23]. The related distributional hypothesis [9], which states that words with similar meaning appear in similar contexts, motivates other work [18, 20]. In [7], named entities in queries are classified into a list of predefined classes using a weakly supervised method. Whereas the above approaches discover flat concepts, ontology building places lexical items and named entities into a hierarchy [5, 26, 19]. Within the framework of the Semantic Web, the discovered ontologies can be used to facilitate inter-website communication [6, 12, for example].

The algorithm of Sarkas et al. [25] maps a query to a table of structured data and to attributes of the table. While they train an unsupervised model to disambiguate between possible mappings, their method requires a highly developed and structured domain schema, which may be available for some domains like the *Products* domain that they test their method on, but is not appropriate for discovering intents in new domains.

On automating sequence labeling of queries, the work most related to ours is the use of supervised or semi-supervised machine learning models for semantic annotation. In [13], a distinction is defined between intent heads, which are slot names indicative of the intent such as "movie", and intent modifiers, which are slot fillers such as *rio* or *matrix*. Discriminative classifiers are trained on labeled data to recognize the distinction. In contrast, our unsupervised technique does not require making this decision and automatically decides which words should be generalized into a slot based on the clusters discovered. Semi-supervised and supervised methods to do query tagging using conditional random fields have also been proposed [14, 29], but these methods still require a predefined schema.

The work of Agarwal et al. [1] studies the importance of query templates and structured patterns. They find that a large majority of queries follow some query template in most domains they examined, and that most queries in a domain can be covered by a small number of templates (in the

hundreds). They also investigate the problem of template discovery. Starting from a seed set of in-domain queries, websites, or templates, a click graph, and at least a partial schema for a domain, their method produces a ranked list of templates relevant to that domain by an iterative graph walking method. Our work is completely unsupervised and does not require any seed annotations or schemata; while we do use a semi-supervised method to extract queries in a domain (Section 7.1), this is merely to speed up computation and facilitate analysis of the resulting clusters. Our method can be applied to the full query log, and we describe two algorithmic speed-ups to support this in Section 4.3. In addition to supporting structured search, query templates can also be used to improve query recommendations [27].

3. FEATURE REPRESENTATION

At the core of our approach is the use of feature vectors to represent queries. The key advantages of a feature vector representation are that semantic and lexical information about the queries can be encoded as features, and similarity scores can be computed between queries, as we will describe.

We let $q = (w_1, w_2, \ldots, w_M)$ denote a query consisting of M word tokens. We create a feature vector of N dimensions for each word token in the query, *i.e.*, $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,N})^T$, $i = 1, 2, \ldots, M$. We will discuss the features shortly. A query, therefore, is represented by an $N \times M$ matrix denoted as $X^q = (\mathbf{x}_1^q, \ldots, \mathbf{x}_M^q)$, where N is fixed and M varies based on query length. Given a large set of queries $\{q\}$, we would like to find query clusters based on X^q. To find semantically meaningful clusters, it is crucial to construct the features in such a way that semantically similar words are close to each other in the vector space. We leverage a knowledge base to achieve this goal.

3.1 Knowledge Base

Freebase is a large knowledge base with structured data harvested from many sources such as Wikipedia. We used the Freebase "Simple Topic Dump" data which consists of over 20 million entities assigned to over 13K concepts. There can be multiple concepts associated with one entity, and multiple entities corresponding to the same surface form. We created a list of concepts for each surface form by merging the concepts of all entities for that surface form. Although the Freebase concepts are hierarchical, we do not make use of the structure of the hierarchy, because it is very flat and uninformative, and because some of the concepts serve a Freebase-internal functional purpose, such as to group together topics defined by a particular Freebase user.

We removed five concepts that each appear in more than two million entities[1], as these are too common to be informative. This results in 13647 concepts.

3.2 Feature Extraction

Our feature vector space consists of $N = N_S + N_L$ dimensions, where N_S represents the number of semantic features and N_L is that of lexical features.

We extract semantic features based on the knowledge base data: if an n-gram $w_{t:t+n}$ in q matches an entity surface form in the freebase data, then for *each* $\mathbf{x}_i$, $i = t : t + n$, we set the value 1 to all the feature dimensions represented by the Freebase concepts that contain this entity surface form.

[1]"/common/topic" (which appears in every entry), "music/track", "book/book_edition", "/book/written_work" and "book/book"

	2010	*buick*	*regal*	*review*
[car]	0	1	1	0
[episode]	1	0	0	1
[model]	0	1	1	0
[season]	1	0	0	0
[year]	1	0	0	0
...	...	...	...	...
review	0	0	0	1
test	0	0	0	0

Figure 2: Sample query representation, abbreviated. Notice that *buick* and *regal* share similar feature vectors because they are found in a surface form *buick regal* in Freebase.

Next, we extract domain-independent lexical features from a corpus. Specifically, we extract all the word-types in the Wall Street Journal corpus that are not predominantly proper nouns or cardinal numbers (*i.e.*, the most common part-of-speech tag for that word-type is not CD, NNP or NNPS). In other words, for each w_i in q, we assign the value 1 to the feature dimension corresponding to the word identity. We removed such words because during intent summarization, we would like our algorithm to generalize these words into an appropriate semantic concept. For example, many clusters in the *Cars* domain contain a specific model year, like "2010", and it would be more useful to generalize this to the semantic concept [year] than to produce a pattern containing the specific year "2010". Figure 2 shows an example of a query's feature representation.

3.3 Inverse Entity Frequency

In standard information retrieval, it is well known that not all words are equally important. Similarly, not all semantic features extracted from Freebase are equally important in determining query similarity. We define a measure of *inverse entity frequency* that is analogous to inverse document frequency in information retrieval. Intuitively, the importance of a shared concept in determining query similarity is inversely correlated with the number of entities contained in that concept, so that sharing a very general concept like [album] provides only weak evidence of similarity, while sharing a specific concept like [sports car] provides stronger evidence.

We experimented with several specific instantiations of this idea, and we found that using the reciprocal entity frequency worked best. That is,

$$REF(y) = 1/size(y), \tag{1}$$

where y is a Freebase concept or lexical item, and $size(y)$ is the number of surface forms that concept y contains, or 1 if y is a lexical item.

4. CLUSTERING

We now introduce our sequence clustering method to discover groups of queries with similar intents. Sequence clustering is one of the most widely used techniques for pattern discovery from sequential data, and is used in virtually every field that deals with sequences, *e.g.*, gene sequences in biology or time series in telecommunications.

There are typically three strategies for clustering sequential data[2]. The first strategy relies on defining a similarity metric between a pair of sequences. Based on a similarity matrix constructed for the entire sequence collection, a good number of clustering algorithms can be applied, including agglomerative clustering such as single-link and complete-link algorithms, partitional clustering such as medoid-based method, and spectral clustering [16]. A second approach is to transform sequences into feature vectors of fixed dimensions, and then to perform any clustering algorithm that operates on feature vectors, *e.g.*, k-means. Note that since similarity can be computed from feature vectors, the algorithms mentioned above for the first strategy are applicable here as well. Finally, there is a class of methods based on generative models. Hidden Markov models, for example, are widely used for modeling sequential data. Clustering is done iteratively by alternately updating model parameters and computing the likelihood of a cluster generating a sequence. Non-parametric approaches are used to deal with the issue of the unknown number of concepts (topics). However, we expect a very large number of concepts in web queries, making such an approach infeasible.

Orthogonal to the above strategies, clustering can be performed in a bottom-up, top-down or partition-based fashion. In this work, we adopt a bottom-up approach based on a similarity metric. The reason is that the number of clusters is likely on the same scale with the number of queries, since there is a long tail of rare queries that each represent a unique intent. This prevents us from starting with K-partitions or applying a top-down approach. Specifically, we choose agglomerative clustering using a distance metric based on dynamic time warping (DTW) between a pair of sequences [21], which we will describe in more detail. While there might be other clustering strategies that could work well, but it is beyond the scope of this work to give a concise evaluation of different clustering algorithms.

4.1 Distance Metric

First, we let $d(\mathbf{x}_i, \mathbf{x}_j)$ denote a distance metric defined on a pair of static feature vectors. Let $\mathbf{r}_i$ be the REF-weighted vector for $\mathbf{x}_i$. That is,

$$\mathbf{r}_i = (REF(1) \times x_{i,1}, \ldots, REF(N) \times x_{i,N})^T. \quad (2)$$

Then, we define the distance as

$$d(\mathbf{x}_i, \mathbf{x}_j) = \log\left(1 - \frac{\mathbf{r}_i \cdot \mathbf{r}_j}{|\mathbf{r}_i||\mathbf{r}_j|}\right). \quad (3)$$

Given a query pair (q, q'), we define an *alignment* as a set of edges E connecting the two sequences of nodes such that (1) every node has at least one edge, and (2) there are no crossing edges (monotonic assumption). The optimal alignment is the one that produces the smallest total distance computed by summing over all edges, *i.e.*,

$$E^* = \underset{E}{\operatorname{argmin}} \sum_{(i,j)\in E} d(\mathbf{x}_i^q, \mathbf{x}_j^{q'}). \quad (4)$$

The DTW distance can be efficiently computed by dynamic programming (Algorithm 1). We normalize the DTW distance by the average query length to compensate for the impact of sequence lengths on distance scores. So,

$$dist(X^q, X^{q'}) = \frac{\min_E \sum_{(i,j)\in E} d(\mathbf{x}_i^q, \mathbf{x}_j^{q'})}{(|q| + |q'|)/2}. \quad (5)$$

Algorithm 1 Dynamic Time Warping

1: $DTW(0,0) = 0$
2: $DTW(i,0) = DTW(0,j) = \infty$, $i = 1..|q|$, $j = 1..|q'|$
3: **for** $i = 1$ to $|q|$ **do**
4: **for** $j = 1$ to $|q'|$ **do**
5: $DTW(i,j) = d(\mathbf{x}_i^q, \mathbf{x}_j^{q'}) + \text{Min}(DTW(i-1,j), DTW(i,j-1), DTW(i-1,j-1))$
6: **end for**
7: **end for**
8: return $DTW(|q|, |q'|)$

4.2 Agglomerative Clustering

Agglomerative clustering starts by treating each data point (a sequence in our case) as a cluster, and then iteratively merging pairs of clusters with the lowest distance (equivalently, highest similarity) until a stopping criterion is satisfied. After each merging step, the cluster distances are recalculated as necessary.

Different versions of agglomerative clustering exist depending on their definition of cluster distance. In single-link clustering,

$$dist(C, C') = \min_{q\in C, q'\in C'} dist(X^q, X^{q'}), \quad (6)$$

while in complete-link clustering,

$$dist(C, C') = \max_{q\in C, q'\in C'} dist(X^q, X^{q'}). \quad (7)$$

If D is the number of input data-points (sequences), single-link clustering has time complexity $O(D^2)$, whereas complete-link clustering has time complexity $O(D^2 \log D)$ [16]. In practice, single-link clustering is faster as well. In our experiments, we evaluated both single- and complete-link agglomerative clustering. Figure 3 shows a sample output cluster in the *Cars* domain using single-link clustering.

There also exists average-link agglomerative clustering, which takes the average distance between cluster instances to be the cluster distance, but it is computationally more expensive and its behavior would simply be intermediate between single- and complete-link, so we did not use it in our experiments.

The stopping criterion can be determined in several ways. Two popular options are to threshold the minimum inter-cluster distance of the remaining clusters, or to set a fixed number of merging iterations. We opt for the former, because the number of queries with similar intent that should be merged is unknown, and this number is different for each domain. At this point, all singleton clusters are removed, and the remaining non-singleton clusters are passed to intent summarization. After the sequence clustering step, when we say cluster we refer to non-singleton clusters.

4.3 Speed-up and Distributed Computation

The main computation bottleneck is to compute the similarity matrix between sequences, which is quadratic in the number of sequences. Since each similarity calculation is

[2]See [11] for a general overview of clustering algorithms, and [8] for one specific to clustering sequential data.

Queries			
2007	audi rs4	review	
2007	infiniti m45	review	
2004	audi s4	review	
2010	jaguar xj	review	
2011	jaguar xj	review	
1987	bmw m6	review	
2010	pontiac gto	review	
09	acura rl	review	
2010	audi tt	review	video
2010	buick regal	review	
2010	mazdaspeed 3	review	
2010	acura zdx	review	
Pattern			
[year]	**[model]**	**review**	

Figure 3: Sample query cluster, with alignment and pattern produced by intent summarization.

independent of other calculations, the matrix computation can be naturally parallelized across the cells of the matrix.

Another possible algorithmic speed-up is to use the inverted index trick to filter out calculations for cells which have no common features. To do so, an inverted index is first constructed from the dimensions of the feature vectors to the queries for which this dimension is non-zero. Then, to compute the similarity between a query q and all of the other queries, the inverted index is consulted for each non-zero dimension in q to obtain a set of queries with at least one overlapping feature. The similarity score calculation is then only performed on this subset of queries, as the similarity scores for the other queries are zero. We did not implement this method, as we found computing the entire similarity matrix to be feasible, but we mention this method to show that our method can be scaled to many more query instances if so desired.

5. INTENT SUMMARIZATION

After clustering, groups of queries with similar intent are constructed. The next step is to produce a pattern (*i.e.*, a label sequence of slots and lexical items) that describes the intent of the cluster, which we call *intent summarization*. We denote a pattern by $p = (y_1, y_2, \ldots, y_L)$, where L is the length of the label sequence. This length is less than or equal to the lengths of the queries being summarized, because one semantic concept may consume multiple query words.

Intent summarization is useful for several reasons. First, if these automatically discovered intents are to inform development of structured search, a human-readable and interpretable pattern would be highly useful to connect the intents to a structured database, or to construct new knowledge bases or tools based on the pattern. Second, the patterns produced by intent summarization would allow generalization of the intents to new query instances, as we will demonstrate in Section 6.

In this work, we use Freebase concepts as slots. Doing so allows us to avoid manually building domain-dependent lists of slots. We use the head word of the name of the Freebase concept as the slot name, unless it is the generic word "topic", in which case we use the concept's parent concept in the Freebase hierarchy. The head word is found by using an in-house chunker with head-finding heuristics. So, "[city with dogs]" would become "[city]".

Intent summarization consists of the following steps. First, the queries are segmented and aligned. Each alignment represents one slot in the pattern. Then, for each position in the pattern, we select a Freebase concept or lexical item to represent the segments at that position. We now describe these steps in more detail.

5.1 Segmentation and Alignment

We segment each query based on the feature vector representation of the words in the query. As a result of the feature extraction procedure discussed in Section 3.2, words that belong to the same entity in Freebase have very similar feature vector representations; *e.g.*, "san" and "francisco" because of the city name "san francisco". We thus segment the query greedily by considering the cosine similarity of adjacent feature vectors, merging words into a segment if the similarity is below a low threshold.

After segmentation, all segments at the same absolute position are aligned and passed on to the concept selection step. There may be errors in clustering or segmentation that lead to some queries having more segments than others, in which case we take the floor of the median to be the length of the pattern. This is, however, rare, because the dynamic time warping algorithm in the clustering step usually results in low similarity in these cases. Also, our concept selection method is robust to many such errors, as we will describe.

5.2 Concept Selection

Suppose we have a cluster of K queries. We define a function, $y_i = generalize(\mathcal{S}_i = s_i^1, \ldots, s_i^K)$, which, given a multiset of segments $\mathcal{S}_i$ at a fixed position i, returns a Freebase concept or lexical item y_i that summarizes the input segments. Note that an arbitrary order is assigned to $\mathcal{S}_i$ so that all of its segments can be enumerated (as indexed by the superscript), and that each segment comes from a different query in the cluster. We assume that the function is independent of the position, so for clarity, we will omit the subscript i in this section.

One natural way to do generalization is to pick the concept that contains the most input segments according to Freebase. This approach, however, turns out not to be robust. Noise, whether from the clustering procedure, or, more commonly, from the Freebase concept omissions and ambiguity of the segments can lead to incorrect generalizations.

For example, suppose we would like to generalize the first position of the queries "2004 audi s4 review", "2010 buick regal review", and "09 acura rl review" . The Freebase concept for [year] contains "2004" and "2010", but not "09". On the other hand, all three segments belong to the concept [tv series episode], so this simple approach would incorrectly select this concept to be the generalization. Furthermore, there is usually no unique concept that maximizes the number of contained segments, and a procedure is needed to select among these features.

To solve these problems, we use a generative model to implement the generalization function with REF scores to initialize the parameters to the model. The intuition for choosing this approach is that we want to prefer the most specific concept (*i.e.*, the concept with the highest REF score) that contains all of the segments according to Freebase, but we

want to be flexible and account for the fact that Freebase concepts are noisy and may contain omissions.

Assuming that each of the segments in the cluster can be modeled independently of the others, the following model gives the joint probability of the segments and the concept:

$$P(\mathcal{S}, y) = P(y) \prod_{k=1}^{K} P(s^k|y). \quad (8)$$

The parameters of the model $P(s|y)$ are initialized using REF scores with linear discounting for smoothing purposes:

$$P(s|y) = \begin{cases} \gamma * REF(y) & \text{if concept } y \text{ contains } s \\ \dfrac{1-\gamma}{F - size(y)} & \text{otherwise.} \end{cases} \quad (9)$$

F represents the total number of surface forms in Freebase. This initialization corresponds to the following scenario for generating a segment from a concept. With probability γ, we randomly select a segment from the concept using a uniform multinomial distribution according to Freebase. With probability $1-\gamma$, we select a segment not contained by that concept according to Freebase (of which there are $F - size(y)$). We set γ to be 0.5. The prior probabilities are initialized uniformly. Inference means finding

$$\begin{aligned} y^* &= \operatorname*{argmax}_y \log P(\mathcal{S}|y) + \log P(y) \\ &= \operatorname*{argmax}_y \sum_{k=1}^{K} \log P(s^k|y) + \log P(y). \end{aligned} \quad (10)$$

Given the parameters, the maximized expression is simple to calculate for each concept and lexical item. It can be sped up by considering only those concepts which contain at least one of the input segments.

While this initialization prefers concepts with higher REF scores and lexical items, it is not ideal because it is susceptible to overly-specific concepts. For example, different subsets of countries would generalize to different concepts, such as [breed origin] or [olympic participating country], whereas in most domains, the desired generalization is simply the more natural [country] concept. To refine the parameters to prefer such "natural" features, we reestimate the prior probabilities using hard EM.

In the E-step, we use the above model to infer y^* for each aligned segment in the cluster. Then in the M-step, we reestimate model parameters, *i.e.*, the prior probabilities, in the maximum likelihood sense:

$$P(y) = \frac{count(y)}{\sum_{j=1}^{N} count(j)}, \quad (11)$$

where $count(y)$ is the number of times y appeared in the E-step's classification, and the denominator is the total number of classifications made. These two steps are alternated until convergence. In our experiments, we simply conduct one iteration of the algorithm and empirically found the results satisfactory, as will be shown in Section 7.2.

Finally, as a further refinement, any clusters that have the same pattern after intent summarization are merged. Figure 3 shows the output of intent summarization on the example query cluster. We could further merge clusters based on semantic intent, as an intent may be expressed by several different, but related patterns, such as "[city] map" and "map of [city]", but we leave this to future work.

	[year]	[model]	review
[car]	0	0	0
[episode]	0	0	0
[model]	0	1	0
[season]	0	0	0
[year]	1	0	0
...	...	...	...
review	0	0	1
test	0	0	0

Figure 4: Pattern representation for instance annotation.

6. INSTANCE ANNOTATION

The procedure above produces a set of patterns within a domain and classifies query instances into these patterns. When encountering new query instances, we would like an efficient method to classify them into one of the existing patterns without having to perform clustering again, which is expensive and would not yield substantially different results. Fortunately, the feature representation in our method suggests a natural way to do this.

For each discovered pattern $p = (y_1, y_2, \ldots, y_L)$, we define $\mathbf{z}_i$ to be a feature vector in the same space as the word tokens that represents the ith element in the pattern. It thus has dimension N, and the interpretation of the dimensions is the same as before. Each $\mathbf{z}_i$ is defined to be a zero vector, except at the dimension corresponding to y_i, where it has value one. So,

$$\begin{aligned} \mathbf{z}_i &= (z_{i,1}, z_{i,2}, \ldots, z_{i,N})^T \\ z_{i,j} &= \begin{cases} 1 & \text{if } y_i = j \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (12)$$

A pattern is then represented by a matrix of size $N \times L$, $Z = (\mathbf{z}_1, \ldots, \mathbf{z}_L)$. See Figure 4 for a sample pattern representation.

Given this representation of a pattern, a new instance can be classified into one of the existing patterns, Z^*, by finding the pattern that is the most similar to the new query, using the same log REF-weighted cosine and DTW algorithm as before to calculate similarity:

$$Z^* = \begin{cases} \operatorname*{argmin}_Z dist(X^q, Z) & \text{if } dist(X^q, Z) < \tau \\ \text{undefined} & \text{otherwise.} \end{cases} \quad (13)$$

If the minimum distance is above a threshold τ, the query is judged to be not similar enough to any of the existing patterns and not classified into one. We set this threshold to be the same as the termination threshold in agglomerative clustering. Note that the inverted index speed-up applies here as well, and would likely result in a great speed-up because the pattern representations are very sparse.

7. EXPERIMENTS

Evaluating unsupervised methods is a challenging problem, especially if there is little labeled data available, as is the case here. As a first evaluation, we elicited human judgments on the output of the algorithms (Evaluations 1 and 2). We also measured the domain coverage of the extracted patterns (Evaluation 3).

Domains		10
Sampled queries		∼101K
Single-link clusters	*total*	1069
	min	13
	max	203
	std	63.5
	avg. size	4.22
Complete-link clusters	*total*	1466
	min	18
	max	279
	std	85.4
	avg. size	2.68

Table 1: Data and cluster characteristics. The attributes *min*, *max* and *std* refer to the number of non-singleton clusters discovered over the domains. *avg. size* is the average number of queries in non-singleton clusters.

We are interested in evaluating two aspects of our system: the patterns that result from intent summarization and the query instances that are classified into each cluster. For each aspect, we can measure the precision; *i.e.*, the quality of the output of our system, or the recall; *i.e.*, whether we have extracted all of the useful patterns and instances that fit a pattern. Naturally, recall is much harder to evaluate in this case because there is no clear way to obtain all of the possible patterns in a domain or query instances that belong to a certain pattern. While one could approximate pattern recall evaluation by comparing to the patterns in a human-labeled dataset, no such labels are available for our dataset, and the labeled datasets that are available, even in related domains, are not appropriate due to differences in schema and data creation. In particular, the schema in a domain is often manually predefined, and queries that do not follow the schema are not included, thus not being representative of actual user needs in a domain.

7.1 Data set

We extracted search queries and segmented them into domains using a click graph. The nodes in the click graph are queries and URLs, while the edges connect queries to URLs based on search engine click-through data. We extracted queries belonging to a particular domain by specifying a number of seed URLs from each domain, and taking the queries that are linked to these URLs. We created ten domains in this manner: *Cars*, *Celebrities*, *Gaming*, *Health*, *Real estate*, *Sports*, *Travel*, *Jobs*, *Movies*, and *Education*.

We sampled at least 10K queries[3] from each domain and ran our sequence clustering and intent summarization algorithms on these samples as described above. When calculating the similarity matrix for sequence clustering, we split computation into fifteen processes, finishing all calculations within ten minutes, and we set the stopping criterion for agglomerative clustering to be a threshold of 2.0 on the minimum cluster distance. While we could have compared our methods to a a bag-of-words baseline, we did not do so because the outcome of our work is to produce grammar rules in which word order is important. We cannot group queries by a bag-of-words approach because it would be hard to express the resulting clusters in this format.

[3]The *Education* domain contained less than 20K queries overall, so we used all of the queries in that domain.

G	N	B	*Single*		*Complete*	
3	0	0	497	(70.7%)	579	(62.6%)
2	1	0	26	(23.3%)	35	(28.6%)
2	0	1	138		230	
1	2	0	2		3	
1	1	1	10	(5.0%)	18	(8.0%)
1	0	2	23		53	
0	2	1	0		2	
0	1	2	3	(1.0%)	1	(0.8%)
0	0	3	4		4	
	Total:		**703**		**925**	

Table 2: Evaluation 1 results in terms of annotator judgments, separated by agglomerative clustering algorithm. G: Good, N: Not sure, B: Bad.

G	N	B	*Before*		*After*	
3	0	0	331	(43.2%)	184	(69.7%)
2	1	0	23	(32.1%)	8	(20.5%)
2	0	1	223		46	
1	2	0	3		2	
1	1	1	32	(22.3%)	5	(9.5%)
1	0	2	136		18	
0	2	1	2		0	
0	1	2	7	(2.3%)	0	(0.4%)
0	0	3	9		1	
	Total:		**766**		**264**	

Table 3: Comparison of the subset of both single- and complete-link clusters changed by EM prior reestimation. The cases in the *After* column are a subset of the cases shown in Table 2

The remaining queries from each domain were treated as unseen queries and annotated using the method described in Section 6. The only exceptions to the input to instance annotation were that we did not run instance annotation on the *Gaming* or *Health* domains, because we found relatively few patterns in these domains. Overall, ∼7.8M unseen queries were processed using the output of both clustering algorithms, resulting in over 15M classification decisions. Of this, ∼125K were classified into a pattern that was discovered. Some other basic statistics on the data and the results of the algorithms are presented in Table 1.

7.2 Evaluation 1: Cluster Pattern Precision

We first evaluated the quality of the cluster patterns that are produced by sequence clustering and intent summarization. For each pattern, we selected between two to three compatible instances arbitrarily, and displayed these to judges along with the pattern. Annotators were given instructions and examples of what good and bad patterns are, then asked to judge whether the pattern describes the instances displayed. We asked annotators to consider the order of the items and whether they fit the concepts in the pattern, as well as the granularity of the concepts, but not the spelling, punctuation, or capitalization. See Appendix A.1 for the complete guidelines. For each pattern, annotators could indicate that the pattern was "good", "bad", or that they were "not sure". Each case was assigned to three different annotators to judge using Mechanical Turk, with a

G	N	B	*Single*		*Complete*	
3	0	0	1282	(65.4%)	1286	(74.5%)
2	1	0	66	(16.3%)	93	(13.2%)
2	0	1	254		134	
1	2	0	3		5	
1	1	1	54	(14.9%)	30	(9.6%)
1	0	2	236		131	
0	3	0	2		0	
0	2	1	6	(3.4%)	1	(2.7%)
0	1	2	10		7	
0	0	3	48		39	
	Total:		**1961**		**1726**	

(a) Original queries

G	N	B	*Single*		*Complete*	
3	0	0	1059	(64.1%)	1012	(68.7%)
2	1	0	63	(20.3%)	76	(17.1%)
2	0	1	272		176	
1	2	0	8		8	
1	1	1	46	(11.0%)	49	(11.7%)
1	0	2	128		116	
0	3	0	1		0	
0	2	1	12	(4.5%)	1	(2.5%)
0	1	2	13		9	
0	0	3	49		27	
	Total:		**1651**		**1474**	

(b) Unseen queries

Table 4: Evaluation 2 results.

resulting free-marginal kappa [22] of 0.657. Table 2 shows the results of this evaluation.

Overall, more than 60% clusters were judged to be correct by all three annotators, and over 90% were correct by at least two annotators. The single-link clusters were more precise than the complete-link ones (Mann-Whitney U $p < 0.001$), but the complete-link algorithm discovered more clusters. Intuitively, complete-link clustering prefers more coherent clusters, which means that the average cluster size will tend to be small, resulting in more non-singleton clusters.

To measure the impact of the EM prior reestimation, we ran the same study on the subset of the clusters where EM reestimation changed the pattern output, before EM reestimation was done. We compared this subset to the analogous subset after EM reestimation (Table 3). Because we filtered out patterns with fewer than two compatible query instances, a different number of patterns qualified before EM reestimation (766 across both clustering algorithms compared to 264). The results show that the hard EM procedure was able to produce substantially more precise patterns (Mann-Whitney U $p < 1^{-14}$), at the cost of having fewer patterns with at least two compatible instances according to Freebase. Notice, however, that this is to be expected, since the point of the prior reestimation is precisely to overcome Freebase annotation errors and omissions.

7.3 Evaluation 2: Query Labeling Precision

In the second evaluation, we asked annotators to judge the correctness of each query instance in a pattern, assuming that the pattern itself is correct. For each case, annotators were given one query (rather than several as in the previous evaluation) and the pattern that describes it, and asked to judge whether the query fits the pattern, does not fit, or whether they are not sure. See Appendix A.2 for the guidelines given to the annotators. We filtered out the patterns that were not judged to be correct by all three annotators from Evaluation 1, as this evaluation is meaningless if the pattern is not correct.

We performed this evaluation not only on the original 10K samples as in Evaluation 1, but also on the classified unseen queries from Section 6, to check the precision of instance annotation. To ensure that the sample sizes and distribution between domains are comparable, we sampled from each domain the same number of unseen queries as were put into a cluster in the original data.

G	N	B	*Incompatible*	
3	0	0	379	(57.3%)
2	1	0	29	(19.2%)
2	0	1	98	
1	2	0	1	
1	1	1	25	(19.0%)
1	0	2	100	
0	3	0	2	
0	2	1	5	(4.5%)
0	1	2	6	
0	0	3	17	
	Total:		**662**	

Table 5: Evaluation of query labeling on subset not compatible with the intent pattern according to Freebase.

The results of the second evaluation are presented in Table 4. The inter-annotator agreement by kappa was 0.692. In contrast to Evaluation 1, the precision of the instances in complete-link clusters was greater (Mann-Whitney U $p < 1^{-10}$). Complete-link clustering is known to produce more coherent clusters than single-link clustering in general, which likely explains this result.

Since query labeling correctness and pattern correctness are highly interconnected, we would expect the original queries to be very precise, since we filtered out the patterns not judged to be correct by all three annotators in Evaluation 1. This is seen in the results, as the precision of the original queries is slightly higher. However, instance annotation results are only slightly lower, despite not being used in the original clustering algorithm. The difference in precision between the original and the unseen cases was statistically significant (Mann-Whitney U $p < 0.01$). Note, however, that the large sample size means that even very small differences in magnitude would be significant.

One interesting point to note is that our algorithms are tolerant of Freebase omissions. For example, the pattern "who is [celebrity] married to" contains many celebrities that are not listed in this concept in Freebase, such as "alanis morissette" and "julie berman". Table 5 shows the results on the subset of instances that do not fit the pattern according to Freebase. As expected, the precision on this subset is lower than the overall precision, but even so, the majority of such instances do actually fit the discovered pattern

Domain	*Single*	*Complete*
cars	5.4%	5.9%
celebrities	1.2%	1.7%
real estate	0.31%	0.45%
sports	0.84%	9.4%
travel	20.2%	21.2 %
jobs	1.2%	1.2 %
movies	2.3%	9.2%
Average:	**4.5%**	**7.0%**

Table 6: Traffic-weighted domain coverage.

according to annotators. Thus, our algorithm can be used to discover new intent-specific slot-fillers that cannot be directly predicted from the knowledge base.

7.4 Evaluation 3: Domain Coverage

Another measure of performance is the domain coverage of the patterns that are extracted; *i.e.*, the proportion of the traffic in a particular domain that is described by one of the patterns that are extracted. We approximate this by calculating the proportion of unseen queries in each domain that are classified into a cluster pattern, weighted by the traffic of the queries. Table 6 shows the coverage results by domain and clustering method. Complete-link clustering results in higher domain coverage than single-link. There is substantial variation in coverage between domains, ranging from less than 0.4% in the *Real estate* domain to more than 20% in the *Travel* domain. This variation is likely due to the Freebase coverage of the entities in that domain. For example, the *Real estate* domain contains many queries that are specific addresses in real estate listings, and many of the local street names are not recognized in Freebase as such. On the other hand, the *Travel* domain contains countries and cities that are correctly recognized.

8. CONCLUSION

We have presented an unsupervised method to find clusters of queries with similar intent, induce patterns to describe these clusters, and classify new queries into the cluster that are discovered, all using only domain-independent resources. Judgments by human annotators indicate that the patterns and the instance annotations are done precisely, and the domain coverage of the patterns that our algorithms found reach up to 20% of traffic in certain domains. Our work can facilitate structured search development by reducing the amount of manual labour needed to extend structured search into a new domain.

There are many avenues of future research. Besides refining the sequence clustering and intent summarization algorithms, it remains to merge patterns with the same intent. For example, word order sometimes (but not always) does not matter, as in "[movie] showtimes" and "showtimes [movie]", and there may be minor variations in the lexical items, such as "[movie] review" and "[movie] reviews" which convey the same intent. Another future direction is to automate the process of matching the slots discovered by our method to a structured database. A modified version of our algorithms could also be applied to match patterns to URLs and website, which often contain structure that can be exploited. Doing so could further automate instant answer generation for many more patterns and domains.

9. ACKNOWLEDGMENTS

The authors would like to thank Alex Acero and Ye-Yi Wang for useful discussions.

References

[1] G. Agarwal, G. Kabra, and K. Chang. Towards rich query interpretation: Walking back and forth for mining query templates. In *Proceedings of the 19th international conference on World wide web*, pages 1–10. ACM, 2010.

[2] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Current Trends in Database Technology-EDBT 2004 Workshops*, pages 395–397. Springer, 2005.

[3] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Improving search engines by query clustering. *Journal of the American Society for Information Science and Technology*, 58(12):1793–1804, 2007.

[4] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 407–416. ACM, 2000.

[5] S. Chuang and L. Chien. Enriching web taxonomies through subject categorization of query terms from search engine logs. *Decision Support Systems*, 35(1): 113–127, 2003.

[6] P. Clerkin, P. Cunningham, and C. Hayes. Ontology discovery for the semantic web using hierarchical clustering. *Semantic Web Mining*, page 27, 2001.

[7] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–274. ACM, 2009.

[8] V. Guralnik and G. Karypis. A scalable algorithm for clustering sequential data. In *Proceedings of the IEEE International Conference on Data Mining*, pages 179–186. Published by the IEEE Computer Society, 2001.

[9] Z. Harris. Distributional structure. *Word*, 1954.

[10] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th Conference on Computational Linguistics*, pages 539–545, 1992.

[11] A. Jain, M. Murty, and P. Flynn. Data clustering: a review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[12] L. Karoui, M. Aufaure, and N. Bennacer. Ontology discovery from web pages: Application to tourism. In *In the Workshop of Knowledge Discovery and Ontologies*. Citeseer, 2004.

[13] X. Li. Understanding the semantic structure of noun phrase queries. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1337–1345. Association for Computational Linguistics, 2010.

[14] X. Li, Y.-Y. Wang, and A. Acero. Extracting structured information from user queries with semi-supervised conditional random fields. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 572–579. ACM, 2009.

[15] D. Lin and P. Pantel. Concept discovery from text. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–7, 2002.

[16] C. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, chapter 17. Cambridge University Press, 2008.

[17] P. Pantel and D. Ravichandran. Automatically labeling semantic classes. In *Proceedings of HLT/NAACL*, volume 4, pages 321–328, 2004.

[18] M. Pasca, D. Lin, J. Bigham, A. Lifchits, and A. Jain. Organizing and searching the world wide web of facts-step one: the one-million fact extraction challenge. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, pages 1400–1405, 2006.

[19] M. Pennacchiotti and P. Pantel. Ontologizing semantic relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 793–800. Association for Computational Linguistics, 2006.

[20] M. Pennacchiotti and P. Pantel. Entity extraction via ensemble semantics. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 238–247. Association for Computational Linguistics, 2009.

[21] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition*, chapter 4. Prentice Hall, 1993.

[22] J. Randolph. Free-marginal multirater kappa (multirater κfree): An alternative to fleiss fixed-marginal multirater kappa. In *Joensuu Learning and Instruction Symposium*, 2005.

[23] E. Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1044–1049, 1996.

[24] E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. Clustering query refinements by user intent. In *Proceedings of the 19th International Conference on World Wide Web*. ACM, 2010.

[25] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *Proceedings of the 2010 International Conference on Management of Data*, pages 771–782. ACM, 2010.

[26] R. Snow, D. Jurafsky, and A. Ng. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 801–808. Association for Computational Linguistics, 2006.

[27] I. Szpektor, A. Gionis, and Y. Maarek. Improving recommendation for long-tail queries via templates. In *Proceedings of the 20th international conference on World wide web*, pages 47–56. ACM, 2011.

[28] J. Wen, J. Nie, and H. Zhang. Clustering user queries of a search engine. In *Proceedings of the 10th International Conference on World Wide Web*, pages 162–168. ACM, 2001.

[29] D. Yu, S. Wang, and L. Deng. Sequential labeling using deep-structured conditional random fields. *IEEE Journal of Selected Topics in Signal Processing*, 4(6): 965–973, 2010.

APPENDIX

A. ANNOTATOR GUIDELINES

A.1 Evaluation 1 Guidelines

Do the phrases fit the pattern?
— A pattern contains category names in square brackets (e.g., [movie]) and/or words or abbreviations (e.g. vs).
— Ignore misspellings or capitalization mistakes and treat words as if they were spelled correctly.
— A good pattern fits the words in the phrases in the correct order.
— A good pattern has categories that are neither too general nor too specific.

Example.

1. kung fu panda 2 showtimes
2. pirates of the caribbean showtimes
GOOD: [movie] showtimes
BAD: [martial arts] 2 showtimes (doesnt́ generalize correctly)
BAD: [location] [artist] (irrelevant patterns)
BAD: [art] showtimes (too general)
BAD: [movies that i have watched] showtimes (too specific)

A.2 Evaluation 2 Guidelines

Does the phrase fit the pattern?
— A pattern contains category names in square brackets (e.g. [movie]) and/or words or abbreviations (e.g. vs). It will be italicized.
— Order matters; a phrase that fits the pattern should fit it in the correct order.
— Think about whether the entire phrase fits in context; not just whether each individual part fits.
— Ignore misspellings or capitalization mistakes and treat words as if they were spelled correctly.
— Ignore any blank cases.

Examples.

1. "kung fu panda 2 showtimes" fits [movie] showtimes, because kung fu panda 2 is a movie and the word showtimes matches and comes after the movie title.
2. "houses for sale in chicago" does not fit houses for sale in [musical], because in this context, Chicago refers to the city, not the musical.
3. "honda 2012" does not fit [year] [model], because the order of the elements is reversed.