Unsupervised Extraction of Template Structure in Web Search Queries

Sandeep Pandey
Yahoo! Research
Sunnyvale, CA 94089
spandey@yahoo-inc.com

Kunal Punera
RelateIQ
Mountain View, CA 94041
kunal.punera@utexas.edu

ABSTRACT

Web search queries are an encoding of the user's search intent and extracting structured information from them can facilitate central search engine operations like improving the ranking of search results and advertisements. Not surprisingly, this area has attracted a lot of attention in the research community in the last few years. The problem is, however, made challenging by the fact that search queries tend to be extremely succinct; a condensation of user search needs to the bare-minimum set of keywords. In this paper we consider the problem of extracting, with no manual intervention, the hidden structure behind the observed search queries in a domain: the origins of the constituent keywords as well as the manner the individual keywords are assembled together. We formalize important properties of the problem and then give a principled solution based on generative models that satisfies these properties. Using manually labeled data we show that the query templates extracted by our solution are superior to those discovered by strong baseline methods.

The query templates extracted by our approach have potential uses in many search engine tasks; query answering, advertisement matching and targeting, to name a few. In this paper we study one such task, estimating Query-Advertisability, and empirically demonstrate that using extracted template information can improve performance over and above the current state-of-the-art.

Categories and Subject Descriptors:
H.3.3[**Information Search and Retrieval**]: Search process

General Terms: Algorithms, Experimentation

Keywords: query templates, intent analysis, graphical models

1. INTRODUCTION

The World Wide Web has grown in size exponentially for many years, and this has been accompanied by search engines becoming the preferred way that users find and access information online. Hundreds of millions of queries are issued to the major search engines everyday, almost all of them in the form of "keywords". Over years of usage, the keyword-search functionality has become the standard convention expected by users and supported by all major search engines. This has resulted in users becoming adept at reducing their search needs specification to the bare minimum set of keywords needed to help the search engine find relevant results. There have been some attempts amongst search engine practitioners to induce users to provide queries in natural text, most notably by Powerset [2], but by and large all search engines expect users to express their search need via a small set of keywords.

Given these state of affairs, there has been a considerable amount of work on extracting as much information as possible from user queries in order to determine their intent [7, 9, 22, 29]. Once this intent is extracted it can be used to provide relevant results improving the search experience [4], to detect whether a query has a commercial intent [22], to select a useful set of advertisements [8], and even to learn from the user's interaction with the search engine [24]. Challenges in extracting the intent information from queries arise from scalability issues since any potential solution must scale to hundreds of millions of queries a day, as well as instrumentation issues since the only user actions search engines observe are user clicks [16, 24]. However, the main difficulty arises from the brevity, and associated ambiguity, of the keywords in the query. Keywords can sometimes be ambiguous when considered without the surrounding context; as in the oft-cited example of the word "jaguar", which can denote the car as well as the animal. On the other hand, considering all keywords in a query together to maintain the context can result in sparsity issues; for example, determining the characteristics (e.g., advertisement click-through-rate) for tail queries such as "jaguar xj12 95 6.0l engine mount" is largely infeasible given the rarity of the query [22].

In this paper we seek to solve these issues by enriching search queries with information about the hidden structure underlying them. In particular, our goal is to develop methods that can automatically determine that the tail query "jaguar xj 12 95 6.0l engine mount" can be described using the <Brand,Model,Year,Part> pattern. We refer to such patterns as *query templates* and their constituents, such as Brand and Model, as *attributes*. Such an enrichment can help in inferring and catering to the user intent behind the query. For example, we can provide custom search experiences for certain templates, such as returning, on the search results page, the availability and prices for vehicles queried via the <Brand,Model,Year> query template. Moreover, these enrichments can help us in dealing with data-sparsity by learning query characteristics (such as click-through-rates, search difficulty) at the template level instead of the individual query level. In fact, in this paper we demonstrate that using query templates can improve performance over the state-of-the-art method for estimating *Query-Advertisability*.

Many past works have proposed methods to extract information in web search queries. Some of these analyze queries to obtain segmentations [5, 19], while others extract named entities from queries [15, 23, 20]. All these approaches require either direct supervision of the tasks, such as manually labeled seed data, or use ancillary information such web search click-through data, both of which might be expensive or difficult to obtain. Some recent works, such as Agarwal et al. [3] and Sarkas et al. [25] focus on detecting templates in queries, while Szpektor et al. [27] use detected templates for improving query recommendations. However, these works assume that the attribute-set as well as the associated vocabularies are given as inputs in form of database relations or entity hierarchies. Given that search engines field queries on a wide array of domains (with user interests following a long-tail [11]) and the terminology used in these queries is constantly in flux (eg., new movies are released every week), it would be prohibitively expensive to create and maintain these attributes and their vocabularies. Hence, in this work we consider the more realistic setting of extracting templates while constructing attributes and their vocabularies at the same time, without requiring any editorial/manual intervention.

At its very core this problem can be looked at as one of grouping keywords into attributes, and hence text-mining algorithms such as Spherical k-Means and LDA can be used [6, 10]. While these approaches are promising, they are not designed to take into account constraints/properties specific to how users construct queries. For example, for any given domain, users are known to employ only a few configurations of attributes as query templates [3], whereas both these methods allow queries to be generated from arbitrary combinations of attributes. Our proposed approach uses this knowledge to reduce the search space of attribute-word vocabularies and template-attribute configurations. In Section 3 we discuss this and other important properties of the problem and how they are incorporated in our model. While this adds significant complexity to the modeling process, we give a procedure to estimate the parameters of the model. In Section 5 we empirically demonstrate that, without any supervision, manual or external (eg. click-through data), our approach is able to extract meaningful templates and attributes from short keyword queries.

OUR CONTRIBUTIONS.

1) We formulate the problem of simultaneously extracting the templates in queries as well as learning the attributes and their vocabularies. As far as we know this is the first work to tackle this problem without requiring any seed input, external data (eg. click-through data), or manual supervision.

2) We describe various characteristics of the way that keyword search queries are formulated by users. We then give a generative model for queries that takes these properties into account.

3) We give a procedure based on Gibbs sampling to automatically infer the query templates and attributes in our model from observed search queries.

4) In our empirical analysis, we use labeled ground truth to show that our approach finds better query templates and attributes than two state-of-the-art approaches based on LDA and k-Means. To ensure representative and robust results we repeat our experiments on real-world queries from three different domains.

5) In order to showcase the usefulness of the query templates detected by our approach we consider the problem of predicting the *Advertisability* of tail queries. Our experiments demonstrate that using automatically inferred query templates can significantly improve upon the state-of-the-art [22].

ORGANIZATION. In the next section we present our formulation of the problem of extracting templates and attributes from observed web search query data. Our proposed solution for this problem is presented in Section 3. We postpone a detailed discussion of related work to Section 4, where we can better contrast our proposed approach to it. In that section we also describe two existing approaches that can be adapted to extract query templates; these serve as strong baselines in the empirical evaluation of our approach using labeled ground truth in Section 5. In Section 5.5 we evaluate the templates discovered by our model on the task of predicting *query advertisability*. Finally, we summarize the paper in Section 6.

2. EXTRACTING QUERY TEMPLATES

In this section we formulate the problem of extracting query templates from web search queries.

2.1 Motivation for Extracting Query Templates

In this paper our goal is to extract, with a completely unsupervised process, a set of domain-specific query patterns that most search queries in a domain conform to. For example, we would like our approach to discover that many search queries in the *Automobiles* domain can be described using the <Brand, Model, Year> pattern. We refer to such patterns as *query templates* and their constituents as *attributes*. Here the attribute *Brand* denotes a placeholder for brand/manufacturer of the vehicle and can stand-in for words such as Honda, Toyota, Ford, etc. Similarly, attribute *Model* can stand-in for words such as Accord, Civic, etc., and, *Year* denotes the year when the car was manufactured. Extracting attributes from query-logs and enriching queries with the templates they conform to can facilitate many search engine operations. For example, due to sparsity issues search engines struggle with obtaining robust estimates of various properties, such as advertisement click-through rates, of tail queries. With the ability to discover that the query "jaguar xj12 95 6.0l engine mount" corresponds to query template <Brand, Model, Year, Parts>, we can smooth the ad-click-through estimates of tail queries with the aggregated estimates of the corresponding templates. Other applications include building custom search solutions for some query templates and using the extracted templates for improved query recommendations [18, 27].

Here we note that some past works [3, 5, 15, 19, 23, 20, 25, 27] have proposed methods to extract similar information from web search queries. However, these approaches require either direct supervision of the tasks, such as manually labeled seed data, or use ancillary information such web search click-through data (more details in Section 4). The manually labeled seed data takes the form of attributes and their vocabularies in [3, 25] and entity classes or hierarchies in [5, 27], all of which are expensive to create and maintain in a dynamically changing environment like web search. While the ancillary information needed in the form of entities [23] and text of documents [19] clicked in response to queries, can be difficult to obtain. Therefore, in this work we restrict ourselves to the setting of extracting templates while constructing attributes and their vocabularies at the same time, without requiring any editorial/manual intervention.

2.2 Desiderata for Template Model for Queries

From our inspection of search query logs we observed that different users with the same search intent issue variations of a query to the search engine. However, at a fundamental level they follow a common process for generating these queries from a common underlying schema, as shown in Figure 1. For example, for the search intent "find information about the 6.0l engine mount of a 1995 jaguar xj12" some users might formulate the query "jaguar xj 12 95 6.0l engine mount" while others might use "jaguar xj 95 engine mount". Both these queries can be thought of as being generated from the query template <Brand, Model, Year, Parts> with the latter query containing fewer terms from the attributes Model and Parts. Similar observations hold for other domains of web search such as Entertainment, Travel, etc. In each case while we do not know the underlying schema – the templates, attributes, and vocabularies are hidden – and do not observe the underlying generative process, we do see the generated query load. Our approach in this paper is to mathematically construct a realistic generative process for the queries so that we can reconstruct (infer) the hidden template-structure used to generate them (as shown in Figure 1).

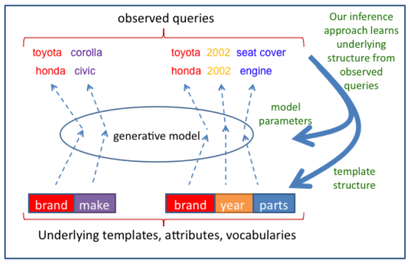


Figure 1: Search queries are assumed to be generated from a common hidden underlying structure. Our goal in this paper is to devise an algorithm to use the observed queries to reconstruct the parameters of the generative process as well as the underlying template structure

The simplest generative process would be *single-attribute template* model whereby each template has a single unique attribute, and each attribute is associated with a set of words (a word distribution). When constructing a query a user picks a template, and then picks words from the associated attribute. This, however, is an unrealistic process since we have seen in the examples above that most queries have words from different attributes of a domain, such as Brand, Year etc. Hence, we consider *multi-attribute template* models whereby each template is associated with multiple attributes. Here, each attribute has its own word distribution from which the words are chosen to instantiate the query.

Besides having multi-attribute templates we list some additional properties that we desire in our model. These properties bring the generative model closer to reality. Moreover, they are also critical for the model performance since queries are short and sparse, and so searching a more constrained/restrictive but realistic model space is likely to be more robust and perform better.

PROPERTY I: While most queries are different, we believe that a large number of queries in a given domain can be captured by only a few templates. In other words, our model must not allow arbitrary distributions of attributes as templates; instead we constrain the number of distinct templates that can be formed in the model.

PROPERTY II: Once a template for generating a query is picked, we force each attribute in the template to generate at least one word. The intuition behind this is that since queries are short, with 2-3 words on average, if an attribute is not contributing any words towards a query, it is not required in the template.

PROPERTY III: Each attribute has a specific word distribution as well as a distinct tendency for the number of words it contributes in a query. For example, the Year attribute in the Automobiles domain is likely to contribute 1 word (e.g., 1995), while the Parts attribute typically generates more words (e.g., 6.0l engine mount).

Following this discussion we can formulate our problem as follows:

PROBLEM DEFINITION: Given a set of queries, extract the underlying schema (templates, attribute, and their vocabularies) and learn the parameters of the generative process in a completely unsupervised manner while respecting the properties mentioned above.

Accounting for these properties significantly increases the technical complexity of our model over the state-of-the-art baseline methods such as LDA [6] and Spherical k-Means [10] that are not designed to enforce these properties. However, in Section 3.2 we show that it is still feasible, mathematically and computationally, to perform inference in our model. Moreover, we empirically show in Section 5 that these properties help our approach significantly improve performance over the baselines.

3. PROPOSED APPROACH

In this section we formally describe our proposed generative model and give an algorithm to perform inference on it. We also describe how our modeling process satisfies the properties mentioned in Section 2.2.

3.1 Generative Model

Our generative model works as follows. We start with a pool of template configurations,[1] called *candidate pool*, whereby each configuration consists of a set of attributes. For example, say we have 3 attributes, Model, Year and Brand. Then the candidate pool of template configurations can be any subset of {<Model>, <Brand>, <Year>, <Model, Brand>, <Model, Year>, <Brand, Year>, <Model, Brand, Year>}. Note that the empty template configuration is not allowed in the pool. In a real-world setting, the candidate pool can also be constructed by a domain expert.

Given the candidate pool we let the model choose T template configurations to construct vector $\vec{\theta}$ where θ_t denotes the template configuration at index t.[2] Then each query q picks a template index t_q which leads to its template configuration θ_{t_q}. This way the queries can pick from only those template configurations which are present in $\vec{\theta}$ (while the candidate pool has many more configurations), helping our model enforce PROPERTY I mentioned above. One way to think of this is that the candidate pool denotes the set of template configurations which are appropriate for the domain. Then the model chooses T template configurations from the candidate pool, $\vec{\theta}$, to best explain the generation of queries. By varying the value of T we can control the trade-off between data likelihood and over-fitting.

Lastly, given a template configuration for a query, from each attribute in the configuration we generate a few words and then arrange them to create the query. More specifically, each attribute has an associated Poisson distribution to determine the number of words it contributes (PROPERTY III), and a Multinomial distribution over the vocabulary to decide which words it contributes. In our running example, in our learnt model we expect the Poisson parameter for the attribute Year to be smaller than that for the attribute Parts. Moreover, we expect that the Multinomial distribution associated with attribute Brand has a much larger probability of generating the word "honda" than, say, the attribute Year. The priors for all distributions are chosen to be their conjugates; Dirichlet for Multinomial and Gamma for Poisson. We enforce PROPERTY II within the inference process described in Section 3.2.

Formally, the parameters of the model are:

$$\begin{aligned}
\mu &\sim \mathit{Dirichlet}(\alpha) \ \{\text{multinomial distribution over all the template configurations in the candidate pool}\}\\
\theta_t &\sim \mathit{Multinomial}(\mu) \ \{\text{configuration for template index } t\}\\
\gamma &\sim \mathit{Dirichlet}(\delta) \ \{\text{vector of size equal to the number of template indices, say } T\text{, that queries are allowed to chose from}\\
t_q &\sim \mathit{Multinomial}(\gamma) \ \{\text{template index for query } q\}\\
\phi_a &\sim \mathit{Dirichlet}(\beta) \ \{\text{multinomial word distribution for attribute } a\}\\
\eta_a &\sim \mathit{Gamma}(g_1, g_2) \ \{\text{Poisson parameter for the number of words that attribute } a \text{ contributes towards a query when the attribute is present in the template for the query}\}
\end{aligned}$$

[1]We use the phrases template configurations, attribute configurations, and query templates interchangeably

[2]For simplicity we construct $\vec{\theta}$ by choosing configurations with replacement, i.e., a configuration can make into $\vec{\theta}$ at two different indices. In other words, for $t \neq t'$, θ_t may be equal to $\theta_{t'}$.

Given these parameters search queries are generated by the following process:

1. Sample the prior over template indices: $\gamma \sim Dirichlet(\delta)$.
2. Sample the prior over allowed template configurations: $\mu \sim Dirichlet(\alpha)$.
3. Sample template configurations for each template index from 1 to T: $\theta_t \sim Multinomial(\mu)$.
4. For each attribute a:
   (a) Sample the word distribution prior: $\phi_a \sim Dirichlet(\beta)$.
   (b) Sample the Poisson parameter: $\eta_a \sim Gamma(g_1, g_2)$.
5. To generate words for query q:
   (a) Sample template index $t_q \sim Multinomial(\gamma)$, which leads to template configuration θ_{t_q}
   (b) For each attribute $a \in \theta_{t_q}$:
      i. Sample $n_{i,a} \sim Poisson(\eta_a)$.
      ii. Place attribute a at $n_{i,a}$ number of positions in vector $\vec{z}_q$.
   (c) For each position j in query q, sample word $w_{q,j}$ from attribute $z_{q,j}$,
   i.e., $w_{q,j} | z_{q,j}, \phi_{z_{q,j}} \sim Multinomial(\phi_{z_{q,j}})$.

Other probabilistic models could be modified to extract templates (such as LDA [6]). In Section 4.1.1 we describe these alternative models in detail and show how our model differs from them. In Section 5 we empirically compare them against each other.

3.2 Model Learning

Here we describe how we perform inference on our model. Due to paucity of space we skip many intermediate steps of algebra; see the longer version [21] for detailed derivation of the expressions.

Recall that $\vec{\theta}$ denotes the set of T template configurations (i.e., $\vec{\theta} = \{\theta_t\}_{t=1}^{T}$) and $\vec{\eta}$ denotes the Poisson parameters for all attributes $\mathcal{A}$, i.e., $\eta = \{\eta_a\}_{a=1}^{|\mathcal{A}|}$. Also, let $\vec{w}, \vec{z}, \vec{t}$ denote the sequence of words, attributes and templates over all queries. Given the hyper-parameters, we first derive the collapsed representation of the joint distribution over the known and latent variables. The joint distribution is then used for inference in Section 3.2.1.

$$\begin{aligned} &P(\vec{w}, \vec{z}, \vec{t}, \vec{\eta}, \vec{\theta}, \phi, \gamma, \mu \mid \alpha, \beta, \delta, g_1, g_2) \\ &\quad= \big(P(\vec{w}|\vec{z}, \phi)\ P(\phi|\beta)\big) \cdot \big(P(\vec{t}|\gamma)\ P(\gamma|\delta)\big) \\ &\qquad \cdot \big(P(\vec{z}|\vec{\theta}, \vec{t}, \vec{\eta})\ P(\vec{\eta}|g_1, g_2)\big) \cdot \big(P(\vec{\theta}|\mu)\ P(\mu|\alpha)\big) \end{aligned}$$

Integration over the latent variables $\vec{\eta}, \phi, \gamma, \mu$ gives:

$$\begin{aligned} &P(\vec{w}, \vec{z}, \vec{t}, \vec{\theta} \mid \alpha, \beta, \delta, g_1, g_2) \\ &= \int \big(P(\vec{w}|\vec{z}, \phi)\ P(\phi|\beta)\big)\, d\phi \cdot \int \big(P(\vec{t}|\gamma)\ P(\gamma|\delta)\big)\, d\gamma \\ &\cdot \int \big(P(\vec{z}|\vec{\theta}, \vec{t}, \vec{\eta})\ P(\vec{\eta}|g_1, g_2)\big)\, d\vec{\eta} \cdot \int \big(P(\vec{\theta}|\mu)\ P(\mu|\alpha)\big)\, d\mu \\ &= \text{TERM I} \cdot \text{TERM II} \cdot \text{TERM III} \cdot \text{TERM IV} \end{aligned}$$

We compute each of these terms in turn.

TERM I

$$\int \big(P(\vec{w}|\vec{z}, \phi)\ P(\phi|\beta)\big)\, d\phi = \prod_{a=1}^{|\mathcal{A}|} \frac{\Delta(\vec{n_a} + \vec{\beta})}{\Delta(\vec{\beta})}$$

where $n_a(w)$ denotes the number of times word w is assigned to attribute a, $\vec{n_a} = \{n_a^{(w)}\}_{w=1}^{V}$, and $\Delta(\vec{\beta}) = \frac{\prod_{k=1}^{dim\ \vec{\beta}} \Gamma(\beta_k)}{\Gamma(\sum_{k=1}^{dim\ \vec{\beta}} \beta_k)}$.

TERM II

$$\int \big(P(\vec{t}|\gamma)\ P(\gamma|\delta)\big)\, d\gamma \quad = \quad \frac{\Delta(\vec{n_t} + \vec{\delta})}{\Delta(\vec{\delta})}$$

where $\vec{n_t} = \{n_t\}_{t=1}^{T}$ denotes the vector of counts of queries assigned to each template index, and T denotes the total number of template indices that queries are allowed to choose from.

TERM III

$$\int \big(P(\vec{z}|\vec{\theta}, \vec{t}, \vec{\eta})\ P(\vec{\eta}|g_1, g_2)\big)\, d\vec{\eta}$$

$$= \int \prod_{q}^{|Q|} \bigg(\frac{1}{n_q!} \prod_{a \in \theta^{(t_q)}} \big(Poisson(n_{q,a}|\eta_a)\ n_{q,a}!\big)\bigg) Gamma(\eta_a|g_1, g_2)\, d\vec{\eta}$$

where n_q denotes the length of query q and $n_{q,a}$ denotes the number of words from attribute a in the query. Since $Gamma$ is conjugate prior for $Poisson$, the above integration can be simplified to:

$$= \quad \Big(\prod_{q}^{|Q|} \frac{1}{n_q!}\Big) \prod_{a \in \mathcal{A}} \bigg(\frac{{g_2}^{g_1} (g_1 + \sum_Q n_{q,a} - 1)!}{(g_2 + N_a)^{(g_1 + \sum_Q n_{q,a})} (g_1 - 1)!}\bigg)$$

where N_a denotes the number of queries with attribute a in their templates.

TERM IV

$$\int \big(P(\vec{\theta}|\mu)\ P(\mu|\alpha)\big)\, d\mu \quad = \quad \frac{\Delta(\vec{n_\theta} + \vec{\alpha})}{\Delta(\vec{\alpha})}$$

where the i^{th} element of n_θ is the number of template indices, from 1 to T, pointing to the template configuration θ^i.

3.2.1 Inference

Our goal is to infer the attribute assignment of words and templates assignment for the queries. Mathematically speaking, we want to infer the distribution $P(\vec{z}, \vec{\theta}, \vec{t}|\vec{w})$, which can be written as:

$$P(\vec{z}, \vec{\theta}, \vec{t}|\vec{w}) \quad = \quad \frac{P(\vec{z}, \vec{\theta}, \vec{t}, \vec{w})}{P(\vec{w})} = \frac{P(\vec{z}, \vec{\theta}, \vec{t}, \vec{w})}{\sum_{\vec{z}, \vec{t}, \vec{\theta}} P(\vec{z}, \vec{\theta}, \vec{t}, \vec{w})}$$

where we omit the hyper-parameters for convenience. Clearly, the denominator in the above expression is a summation over a large number of combinations and is difficult to compute. Hence, we use Gibbs sampling to perform this inference [13, 14].[3] Under the Gibbs sampling procedure, the full conditional distributions ($P(z_i|\vec{z}_{\neg i}, \vec{\theta}, \vec{t}, \vec{w})$, $P(\theta_i|\vec{z}, \vec{\theta}_{\neg i}, \vec{t}, \vec{w})$, $P(t_i|\vec{z}, \vec{\theta}, \vec{t}_{\neg i}, \vec{w})$) are used to simulate $P(\vec{z}, \vec{\theta}, \vec{t}|\vec{w})$. To derive these conditionals, we use the joint distribution $P(\vec{z}, \vec{\theta}, \vec{t}, \vec{w})$ (computed earlier):

$$P(\vec{z}, \vec{\theta}, \vec{t}, \vec{w}|\alpha, \beta, \delta, g_1, g_2) = P(\vec{w}|\vec{z}, \beta)\ p(\vec{\theta}|\alpha)\ p(\vec{t}|\delta)\ p(\vec{z}|\vec{\theta}, \vec{t}, g_1, g_2)$$

Next we give derivation of the conditional distributions for different latent variables, i.e., z_i, θ_i, t_i. As mentioned above, these conditionals are then used to perform the Gibbs sampling for inferring the query templates and attributes. The overview of our complete approach is given in Section 3.3.

[3] In the general formulation of a Gibbs sampler, the latent variables, say $\vec{x}$, are estimated by computing: $P(\vec{x}|\vec{w}) = P(x_i|\vec{x}_{\neg i}, \vec{w}) = \frac{P(\vec{x}, \vec{w})}{P(\vec{x}_{\neg i}, \vec{w})} = \frac{P(\vec{x}, \vec{w})}{\int_X P(\vec{x}, \vec{w}) dx_i}$.

3.2.2 Computing the Conditional Distributions

Here we simply give the expressions for the conditional distributions and give the intuition behind them; please refer to the longer version [21] for complete derivations.

CONDITIONAL FOR z.
Ideally, we would like to compute $(P(z_i|\vec{z}_{\neg i}, \vec{\theta}, \vec{t}, \vec{w})$ where z_i is the i^{th} element of attribute sequence z. But in our case z_i's are not sampled independently as this can result in one of the attributes contributing zero words to the query (we do not allow this under PROPERTY II mentioned above). Hence, the sampling is done on per-query basis, say $\vec{z}_q$, which consists of the attributes for every word in the query. Hence, it is convenient to compute the conditional in terms of $(P(\vec{z}_q|\vec{z}_{\neg q}, \vec{\theta}, \vec{t}, \vec{w})$. One can show that (derivation in the long version [21]):

$$P(\vec{z}_q = \vec{v}_q|\vec{z}_{\neg q}, \vec{\theta}, \vec{t}, w) = \frac{1}{n_q} \prod_{a \in \vec{v}_q} \frac{(g_2+N_a-1)^{(g_1+\sum_Q n_{i,a}-n_{q,a})}}{(g_2+N_a)^{(g_1+\sum_Q n_{i,a}-n_{q,a}+v_{q,a})}} \frac{(g_1+\sum_Q n_{i,a}-1-n_{q,a}+v_{q,a})!}{(g_1+\sum_Q n_{i,a}-1-n_{q,a})!}$$

where $\vec{v}_q$ denotes a new attribute sequence for query q, N_a denotes the number of queries with attribute a in their templates, n_i is the length of query i, $n_{i,a}$ is the number of terms in query i from attribute a, and $n_{q,a}$ and $v_{q,a}$ denote the number of terms from attribute a in the old query configuration q and the new configuration v, respectively.

CONDITIONAL FOR t.
Recall that $\vec{t}$ denotes the vector consisting of the template indices of all queries. We compute the conditional by deriving the probability of updating the template index of query j.

$$P(t_j = k|\vec{t}_{\neg j}, \vec{z}, \vec{w}, \vec{\theta}) \propto \frac{P(\vec{z}|\vec{\theta}, \vec{t})}{P(\vec{z}_{\neg j}|\vec{\theta}, \vec{t}_{\neg j})} \frac{P(\vec{t})}{P(\vec{t}_{\neg j})}$$

We computed $\frac{P(\vec{z}|\vec{\theta}, \vec{t})}{P(\vec{z}_{\neg j}|\vec{\theta}, \vec{t}_{\neg j})}$ earlier. Next we look at $\frac{P(\vec{t})}{P(\vec{t}_{\neg j})}$.

$$\frac{P(t_j = k, \vec{t}_{\neg j})}{P(\vec{t}_{\neg j})} = \frac{n_{t,\neg j}^{(k)} + \delta_k}{\sum_k n_{t,\neg j}^{(k)} + \delta_k}$$

where $n_t^{(k)}$ and $n_{t,\neg j}^{(k)}$ denote the number of queries assigned template index k, with and without query j. Hence, $n_t^{(k)} = n_{t,\neg j}^{(k)} + 1$.

CONDITIONAL FOR θ.
Next we compute the conditional for θ, i.e., $P(\theta_j|\vec{\theta}_{\neg j}, \vec{z}, \vec{w}, \vec{t})$. Recall that $\vec{\theta}$ denotes the vector of T template configurations selected by the model, from which each query is assigned a configuration. Note that by updating the index j of $\vec{\theta}$ to any configuration c (i.e., set $\theta_j = c$), we indirectly update the template of each query that is pointing to θ_j (i.e., queries which have $t_q = j$). Hence,

$$P(\theta_j = c|\vec{\theta}_{\neg j}, \vec{z}, \vec{w}, \vec{t}) \propto \frac{P(\vec{z}|\vec{\theta}, \vec{t})}{P(\vec{z}_{\neg j}|\theta_{\neg j}, \vec{t})} \frac{P(\vec{\theta})}{P(\vec{\theta}_{\neg j})}$$

where $\vec{z}_{\neg j}$ denotes the attribute sequence, excluding all the queries whose template is pointing to θ_j configuration. We can compute $\frac{P(\vec{z}|\vec{\theta}, \vec{t})}{P(\vec{z}_{\neg j}|\vec{\theta}_{\neg j}, \vec{t})}$ as we computed it earlier. Next we look at $\frac{P(\vec{\theta})}{P(\vec{\theta}_{\neg j})}$

$$\frac{P(\theta_j = c, \theta_{\neg j})}{P(\theta_{\neg j})} = \frac{n_{\theta,\neg j}^{(c)} + \alpha_c}{\sum_{c'} n_{\theta,\neg j}^{(c')} + \alpha_{c'}}$$

where $n_\theta^{(c)}$ and $n_{\theta,\neg j}^{(c)}$ denote the number of elements in vector $\vec{\theta}$ that have attribute configuration c, with and without including element j. Hence, $n_\theta^{(c)} = n_{\theta,\neg j}^{(c)} + 1$.

3.3 Overview of the Inference Algorithm

Above we have described the model and the update equations (i.e., conditionals). Here we summarize how the conditionals are used to perform the inference. The procedure begins with a random initialization of the queryword-attribute assignment $\vec{z}$, the query-template assignment $\vec{t}$, and the set of T template configurations $\vec{\theta}$. Then we iterate over queries and the template set using the conditionals derived in Section 3.2.2 to update the $\vec{z}$, $\vec{t}$, and $\vec{\theta}$ vectors. At each iteration we compute the likelihood of the observed query data given the current learnt model parameters (ϕ, μ, γ, $\vec{\eta}$). The procedure ends with an assignment of each query to a template, and each word in the query to an attribute.

4. ALTERNATIVE APPROACHES AND A SURVEY OF RELATED WORK

Some existing methods can be adapted to tackle the problem of discovering templates; in this section we describe two such approaches, LDA, and k-Means, and highlight the ways in which our proposed model differs from them. We end this section with a survey of some past works that are broadly related to our problem setting.

4.1 Alternative Approaches

4.1.1 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation is a popular model for unsupervised discovery of document topics [6, 14]. Before showing how it can be applied for template extraction, we briefly describe the generative model here. In the LDA model, each topic has an associated ϕ distribution over the vocabulary. To construct document d, first a multinomial distribution over the topics, denoted by θ^d is sampled from a Dirichlet prior. The i^{th} word in the document is picked by choosing a topic from this multinomial distribution, and then sampling a word from the ϕ distribution associated with the topic.

In our scenario each query can be thought of as a document. By applying LDA we can find the attributes (i.e., the topics) and their associated vocabularies that have been used to generate the queries. These attributes can then be used to construct the query templates.

Here we note that there are some fundamental differences between the LDA model and our proposed generative model. First, LDA allows each query to pick an arbitrary topic distribution, while we constrain the query to conform to one of the finite templates (i.e., attribute configurations) as is required by PROPERTY I of the problem definition in Section 2.2. Second, even if the topic distribution for a query has a high probability for a topic, LDA does not require that topic to contribute a word in the query. In contrast, our model forces each attributes in the template to contribute at least one word to the query enforcing PROPERTY II of the problem definition. As discussed in detail in Section 2.2 these properties bring the generation closer to reality, and reduce the model search for our approach to a more constrained and realistic space; this is particularly helpful since queries are short and sparse.

Incorporating these properties in our model adds significant technical complexity. For example, for enforcing PROPERTY I we maintain a finite pool of allowed configurations ($\vec{\theta}$). All other latent variables (e.g., $\vec{z}$, $\vec{t}$) depend on this $\vec{\theta}$ vector, and as a result, when $\vec{\theta}$ is updated in the inference process, all other latent variables have to be updated accordingly. Similar, enforcing PROPERTY II means that we cannot sample the attributes for words (z's) independently. Instead, the sampling is done on a per-query basis. In our experiments we justify this added complexity by comparing LDA with our proposed model and showing that our approach results in much better template extraction performance.

4.1.2 k-Means

The problem of extracting attributes can be framed as a problem of grouping the query words, and hence any text-clustering approach can be used. In this section we describe how Spherical k-Means [10] can be applied. We represent each word w_u as a vocabulary-sized vector, where each element v of the vector contains the number of times words w_u and w_v occur together in queries. By running Spherical k-Means on these vectors, we put together those words into a cluster which have similar co-occurrence behavior as other words, e.g., words such as accord and toyota should have similar co-occurrence behavior with respect to brand words and years, and should fall into the same cluster. Hence, these clusters act like attributes for our scenario; using them we construct query templates.

4.2 Survey of Related Work

Many past works have tackled problems related to modeling query keywords. While a full survey is not possible due to lack of space, we discuss some key works that can help us put our work into context.

The problem of assigning an attribute to each word in a query has been explored in [3, 25]. Agarwal et al. [3] proposed an approach based on random-walk on the tri-partite graph of queries, sites, and templates. Sarkas et al. [25] assumed that "structured" data is given in the form of tables. Then queries are annotated by mapping a query to a table and the attributes of this table. Both these works, however, assume that the attributes and their vocabularies are given as input to the algorithms. In contrast our work finds both the query templates as well as the attributes and their vocabularies in a completely unsupervised manner.

The problem of named entity recognition in web queries (eg. finding movie names) is related to our problem with entities playing the role of attributes. In [15], Guo et al. give a nice semi-supervised approach to extract entities from queries while ensuring that the model topics and pre-defined classes align. In [30] the approach learns a topic-model using click-through data under some supervision from humans and uses the model to resolve ambiguities among named entities. In another related work [20] a weakly-supervised extraction framework is given for extracting named entities from web search queries starting from a seed set. Another set of work seek to obtain useful segmentations of web queries via learning from labeled examples [5] or using click-through data [19]. Our work differs from these in the focus – we focus assigning all words to attributes, not just named entities – and based on the fact that our proposed approach works with just the query-set and does not need manual intervention or data click-through interactions.

Finally, there are recent works that seek to exploit query templates for accomplishing search related tasks. In [27] Szpektor et al. used entity hierarchies to mine query templates, which were used to improve query recommendation algorithms by defining better relationships between queries. Similarly, Jain et al. [18] use many heuristics to define relationships between queries, and query templates could be used as one such signal. Hence, our work is complementary to these works that seek to improve web mining algorithms via query templates and could act as input into them.

5. EXPERIMENTS

In this section we analyze the performance of our approach (QTGEN) on real-world search engine queries. We also provide an empirical comparison with the state-of-the-art alternatives described in Section 4.1. We begin with a description of the experimental methodology and then proceed to describing the results. We end the section with an application of our approach to the task of predicting query-advertisability.

Domain	Attribute	Vocabulary	Vocab-size
Automobiles	Brand	Honda, Toyota ...	56
	Model	Civic, Camry ...	188
	Year	2010, 2009 ...	73
	Parts	engine, tires ...	148
	Specs	mpg, 250hp ...	30
	Vehicle_type	car, sedan ...	44
Travel	Tasks	purchase, flights ...	42
	Brand	hilton, southwest ...	27
	Location	hawaii, SFO ...	132
	Qualifiers	cheap, discount ...	27
Movies	Tasks	tickets, reviews ...	54
	Names	avatar, brad pitt ...	60
	Genres	horror, bollywood ...	25
	Qualifiers	free, online ...	20

Table 1: Ground Truth: attributes and vocabularies created manually. The learned attributes output by QTGEN **and baselines are evaluated in terms of their match to this ground truth.**

5.1 Experimental Setup

We first describe the construction of the query dataset and ground truth. We then describe the implementation details of our approach, QTGEN, and of the competing baselines.

QUERY DATA-SET AND GROUND TRUTH.

In order to obtain robust results and reduce the effect of any one topic we perform our empirical evaluation using queries from multiple different domains: *Automobiles*, *Travel*, and *Movies*. This is the standard methodology in this area. Topical classification of queries is a well-studied problem and we use a state-of-the-art multi-label classifier [28] to classify a randomly sampled set of 100 million Yahoo search queries executed in September of 2010 into the above domains. These queries were suitably anonymized and care was taken to remove all personally identifiable information was removed. From these domain-specific queries we construct datasets for our two evaluation tasks. First, we construct three domain-specific query-sets of roughly 1000 queries each; the size was picked so that we could manually construct the ground truth. Second, for the large-scale automatic evaluation on the query-advertisability task we construct three datasets with 43793, 83387, and 15050 tail queries from the Automobiles, Travel, and Movies domains, respectively. For this task we also obtain the sponsored search impressions and clicks from the search logs.

In order to construct the ground truth, templates underlying the queries in these query-sets were manually extracted. This resulted in the construction of 6 attributes for Automobiles domain and 4 each for the Travel and Movies domains, each of which was populated with the words likely to be generated from them; some words were labeled as being generated from multiple attributes. Some details about the ground truth are given in Table 1. Note that the ground truth construction was completed before the outputs of any of the approaches under evaluation were seen by the editors.

OUR APPROACH AND BASELINES.

QTGEN: This is an implementation of the approach outlined in Section 3. Each run of QTGEN is parameterized by the following settings: number of attributes (k), number of iterations (N), and values of model parameters β, g_1, and g_2. Please see the description of our model for details of these parameters. In our empirical analysis we perform many experiments by varying values for these parameters, but unless mentioned otherwise, the values are set to $k = 5$, $N = 100$, $\beta = 0.1$, $g_1 = 4$, and $g_2 = 0.2$; these parameter values were tuned using a 10% validation set.

LDA: For this baseline approach we used the Mallet [1] implementation of Latent Dirichlet Allocation, which has been described in detail in Section 4.1.1. The parameters values are set to $k = 5$, $\alpha = 50$ and $\beta = 0.01$.

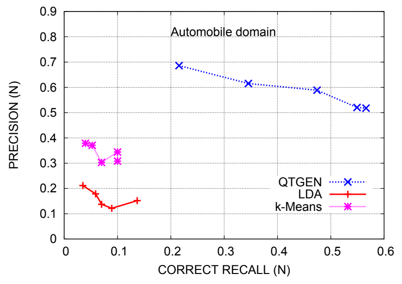


Figure 2: PRECISION vs. CORRECTRECALL curves of QTGEN and baselines on the Automobiles domain. Each mark on the curves (from left to right) is produced by evaluating the top-N learnt attributes.

K-MEANS: This baseline approach is described in detail in Section 4.1.2. In this implementation the initialization is done via farthest first approach. For our experiments, k was set to 5 (set using a 10% validation dataset) and distance measure between vectors was computed using cosine similarity [10].

5.2 Evaluation on Manually Labeled Ground Truth

Above we described the process through which the templates that generate queries were manually extracted for three domains. In this section we evaluate the performance of QTGEN, LDA, and K-MEANS in successfully retrieving the attributes that form these templates. Before proceeding to the results we describe the metrics we use to evaluate the outputs of the various approaches.

INTERPRETING THE OUTPUT OF ALGORITHMS.

Traditional clustering evaluation measures such as pairs-based ones (e.g., Adjusted RAND [17]) and entropy-based ones (e.g., normalized mutual information [26]) are not suited for the tasks we consider in this paper. In designing an evaluation scheme relevant to our problem setting, we first attempt to understand how the extracted templates/attributes are likely to be used. Most applications of interest need the attributes to be returned in some order, and since none of the approaches under evaluation ranks the attributes, we assume each approach returns attributes in the decreasing order of the number of queries that generate words from it. Once the order is set, we want that each learned attribute contain words from only one attribute from ground truth. Moreover, we want that for each attribute in the ground truth, all its words be covered by one learnt attribute.

To fulfill these requirements each ground truth attribute must be mapped to a unique learnt attribute. In a real-life application, this mapping would be constructed by a human expert when she studies the attributes output by the system. In order to evaluate numerous runs of our approach and baselines objectively, we construct this mapping automatically by evaluating all possible mapping in terms of the total AUC [12] between the learnt and the ground truth attributes, and picking the one with the maximum score.

Once the mapping is fixed, we go over the learnt attributes in the order established and evaluate them in terms of net PRECISION and CORRECTRECALL. We say that a word is *correctly placed* if the learnt attribute and the ground truth attribute it belongs to are mapped to each other. Hence, PRECISION(N) is the fraction of words in the first N learnt attributes (in the algorithm's ordering) that are correctly placed. Similarly, CORRECTRECALL(N), is the fraction of words in ground truth attributes mapped to the first N learnt attributes that are correctly placed. Intuitively, it can be seen that PRECISION measures the accuracy of the system while CORRECTRECALL measures the correct coverage.

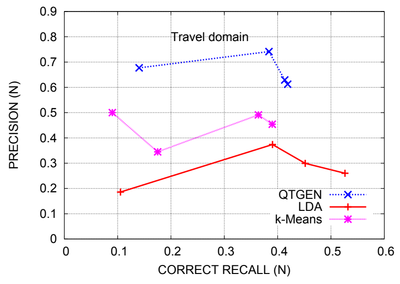


Figure 3: PRECISION vs. CORRECTRECALL curves of QTGEN and baselines on the Travel domain. Each mark on the curves (from left to right) is produced by evaluating the top-N learnt attributes.

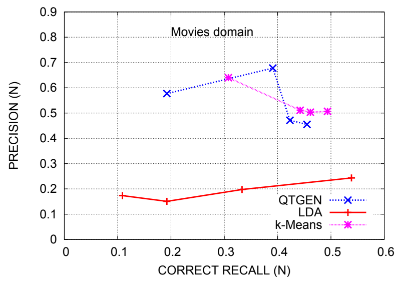


Figure 4: PRECISION vs. CORRECTRECALL curves of QTGEN and baselines on the Movies domain. Each mark on the curves (from left to right) is produced by evaluating the top-N learnt attributes.

PRECISION VS. CORRECTRECALL.

We plot PRECISION and CORRECTRECALL of all approaches on the three domains in Figures 2, 3, and 4. The PRECISION is plotted on the y-axis while the CORRECTRECALL is on the x-axis. The markers on each curve indicate the performance values at different N, with markers on the left indicating values for lower N. As we can see, in general, at higher values of N, PRECISION tends to decrease while CORRECTRECALL monotonically increases.

First we observe that attributes found by QTGEN match the ground truth very precisely in their word compositions. For example, in the Automobiles domain, the top-3 learnt attributes have > 60% of the words that were assigned to them and were also manually labeled as belonging to these attributes. Similar results are also seen for the other two domains. Being this precise makes the learnt attributes easier for downstream human users to interpret as well as easier for automated algorithms to use. We also observe that for all domains and operation points, the attributes found by QTGEN dominate those found by LDA and K-MEANS in terms of CORRECTRECALL. The runners-up in terms of PRECISION is clearly the K-MEANS approach; in fact it equals QTGEN in terms of PRECISION for the Movies domain.

The second observation about the results concerns the CORRECTRECALL of the learnt attributes. As we can see, upon learning 5 attributes our approach consistently finds around 50% of the words in the matched ground truth attributes. On the Automobiles dataset, QTGEN vastly outperforms the baseline approaches, while on the other two domains the algorithms are roughly comparable in terms of CORRECTRECALL. In fact, LDA, and to a lesser degree K-MEANS, consistently operate at a lower PRECISION and slightly higher CORRECTRECALL profile. In many applications producing output with high CORRECTRECALL is very important, however, we feel that the the low PRECISION of attributes learnt by LDA and K-MEANS will make them too obscure to be used automatically by algorithms or manually by human editors.

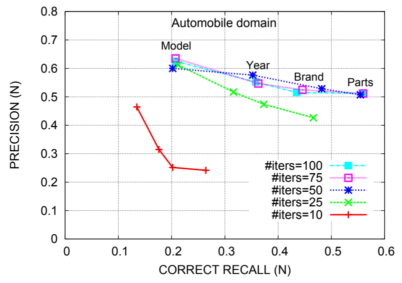


Figure 5: PRECISION vs. CORRECTRECALL curves of QTGEN after different number of iterations on the Automobiles domain.

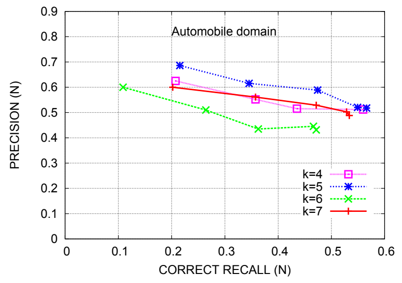


Figure 6: PRECISION vs. CORRECTRECALL curves of QTGEN using different number of attributes on the Automobiles domain.

Finally, we note that the results indicate that learnt attributes that are heavily used also score high in terms of PRECISION and CORRECTRECALL. This is completely true in the case of the Automobiles domains and to a slightly lesser degree for the other two domains. This can be seen in the curves of QTGEN in Figures 2, 3, and 4 which are constructed by considering attributes in the decreasing order of their usage. As we can see for the performance curve of QTGEN on Automobiles domain (Figure 2), the PRECISION drops monotonically, indicating that the most precise attributes are the most heavily used, and the gaps between successive marks on the x-axis become smaller, indicating the same trend for the CORRECTRECALL of learnt attributes. It is also clear from the figures that the attributes learnt by LDA and K-MEANS display this desirable property to a much lesser degree.

To summarize, we have seen that QTGEN outperforms the baselines in terms of obtaining interpretable attributes that cover a large fraction of the query-terms correctly. The parameter setting used to report this performance was tuned over a 10% held-out validation dataset. Next, we show the effect of variation of these parameters on the performance of QTGEN in terms of PRECISION and CORRECTRECALL.

5.3 Effect of Parameter Values

EFFECT OF NUMBER OF ITERATIONS.

We start by studying the effect of the number of model iterations. In Figure 5 we have plotted the scores of attributes identified by QTGEN after a fixed number of iterations. These performance numbers are plotted by averaging the result of multiple runs with random initializations, but we do not show the confidence intervals to reduce clutter. Note that the results of this experiment on all three domains were very similar and hence we show them only for the Automobiles domain. From the results displayed in Figure 5 we make two observations.

First, it can be easily seen that while the performance of QTGEN does improve with increasing iterations, after very few iterations (>50) the performance differences become indistinguishable. The only statistically significant differences in performance are between

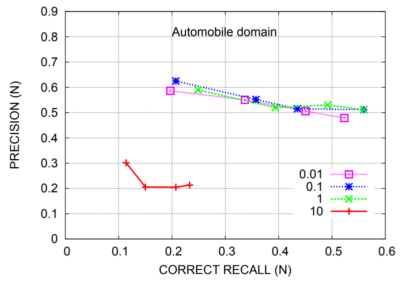


Figure 7: PRECISION vs. CORRECTRECALL curves of QTGEN run with different values of ϕ on the Automobiles domain.

the $\#iters = 10$ and $\#iters = 25$ curves and the rest. This fast convergence of QTGEN can be attributed to the fact that it imposes a lot of structure derived from domain knowledge on the model. For instance, in our model queries are required to be generated from one of a few attribute configurations and each attribute is required to generate at least one word. These restrictions reduce the number of choices that must be evaluated and hence the convergence is achieved in fewer iterations.

Second, we observe that QTGEN converges faster to a precise characterization of attributes that are used more frequently than it does for other less frequently used attributes. In Figure 5 we have annotated each mark of a converged attribute with the ground truth attribute that it maps to. As we can see, QTGEN converges to the final characterization of the attribute that maps to *Model* first. This is because a model-name is specified in nearly every query in our data. Next most commonly used attribute is *Year*, which, in addition, also has a small vocabulary and is hence found in few iterations as well. Finally, *Brand* and *Parts* that participate in fewer queries are found.

EFFECT OF NUMBER OF ATTRIBUTES.

In Figure 6 we plot the performance of QTGEN when it learns templates by allowing for different number of attributes. As we can see the performance for most settings are very similar showing that with any given number of attributes QTGEN is able to learn attributes that closely map to ground truth attributes. However, the main difference is in the number of ground truth attributes that are covered by the learnt attributes. This difference is primarily represented in the CORRECTRECALL values that QTGEN is able to achieve with the lower settings of number of attributes.

Often, in unsupervised learning scenarios, one of the hardest parameters for a domain expert to set is the true number of clusters / attributes in the data. These results show that QTGEN has the ability to find appropriate attributes as long as the number of desired attributes is set to a reasonable value.

EFFECT OF MODEL PARAMETERS.

In these experiments we report on the results of tuning the Dirichlet parameter ϕ that acts as a prior to the attribute-word multinomial distributions and the Gamma distribution parameters g_1 and g_2 that are used to generate the Poisson distributions for each attribute. The ϕ parameters controls the extent to which the modeling procedure relies on the observed data as opposed to the priors. The smaller the value of ϕ, the more the multinomial distribution of words associated with each attribute is tuned to the data. In Figure 7 we plot the performance of QTGEN when run with different settings of ϕ. It is clear from the results that for a wide range of parameters values the results stay stable. Only in the case when the value of ϕ is very high do the results deteriorate since the system now relies on the prior too much and ignores the evidence of the data.

The Gamma distribution parameters g_1 and g_2 control the Poisson parameters which in turn control the propensity of each attribute to generate words in the query. We performed experiments with

Attribute	Words most frequently used in queries
Brand	honda, toyota, chevy, ford, jeep, nissan, dodge, mustang, ranger
Model	truck, camaro, corvette, civic, bmw, accord, silverado, yukon, ram
Year	2010, 2007, 2008, 2006, 2004, 2005, engine, 2011, 2009, custom
Parts	parts, accessories, couple, rims, reviews, problems, tires, manual, seat
Specs	owners, belt, floor, door, coupe

(a)

Query Template	Frequency	Ad-clicks
Brand Model Year	33.3%	11%
Brand Model Year Parts	16.6%	8.7%
Brand Model Year Parts Specs	13.6%	5.3%
Model Year Parts	8%	2.9%
Brand Model Parts	7%	4.2%

(b)

Table 2: Structure extracted by QTGEN **for the Automobiles domain. Table (a) shows the attributes found along with the most popular words in them. The attribute name is the ground truth attribute it matched with in our evaluation. Table (b) shows the top-5 frequently used query templates found by** QTGEN **in the Automobiles domain. The frequency indicates the fraction of the query traffic that is generated from this template. The Ad-clicks is fraction of all ad clicks on Automobiles domain queries that are on queries generated from this template.**

settings that forced attributes to generate very few words per query and others that allowed attributes to generate more. Our results were remarkably similar across these settings and we do not show them here due to paucity of space.

Our results in this section show that QTGEN is extremely robust to changes in parameters values and performs well for all values in a reasonable range.

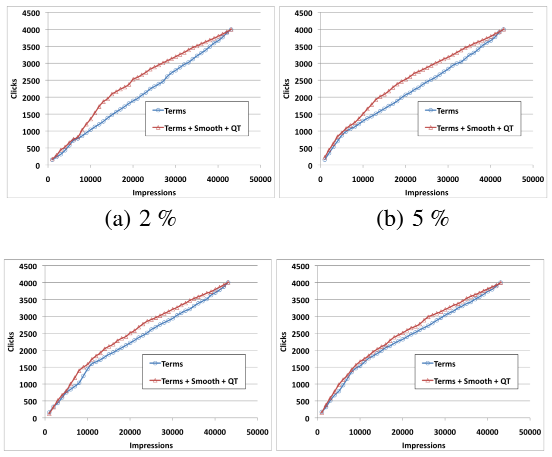
5.4 Anecdotal Evidence and Discussion

Here we present a qualitative evaluation of the attributes and templates found by QTGEN and discuss potential applications.

In Table 2 we have shown the template structure uncovered by QTGEN. Table 2(a) shows the attributes found along with the most popular words in their vocabularies. The attribute name in the table is the ground truth attribute that matched it. As is clear most of the words are grouped together into coherent attributes. Moreover, these attributes and their vocabularies closely match those in the manually constructed ground truth in Table 1.

In Table 2(b) we have listed the five most popular query templates, and the fraction of queries in our dataset covered by them, as detected by QTGEN. The results make intuitive sense. The most popular query template found by QTGEN is "Vehicle" queries (that covers 33% of queries in our set) and describes the vehicle of interest using the three most important attributes, Brand, Model, and Year. These users are most probably researching an automobile for a purchase or lease. Beyond this template the users are querying for more detailed information on specific vehicles: to do this they first establish the identity of the vehicle using some combination of Brand, Model, and Year, and then express their information need via attributes Part and Spec. These "Parts" queries all together are about as frequent as the "Vehicle" queries in our dataset.

In the final column of Table 2 we list the fraction of all sponsored search advertisement clicks (in the Automobiles domain in our dataset) due to queries that are generated from these templates. As we can see these query templates demonstrate a large difference in their tendency to attract ad-clicks from users. For example, the "Parts" queries contribute a larger fraction of ad-clicks while occurring a smaller fraction of times than the "Vehicle" queries. Hence, knowledge gathered by QTGEN on the template from which the query has been generated should be useful for inferring its advertisability [22]. Other potential applications of extracting the attributes and query templates can be in generating special case search experiences, such as returning on the search results page the price and availability of the automotive part plugged into the query template.



(a) 2 % (b) 5 % (c) 10 % (d) 50 %

Figure 8: Clicks vs Impressions curves for models trained using varying amounts of data in the Travel domain.

	Approaches	
% of Training data	TERM + SMOOTH	TERM + SMOOTH + QT
2%	**6.26%**	**12.04 %**
5%	**6.57%**	**10.41%**
10%	**4.29%**	**6.20%**
50%	**2.81%**	**4.18%**
100%	-0.83%	1.35%

Table 3: The %-improvement in the AUC of predicting the query-advertisability for different approaches. The two approaches using SMOOTH **and** QT **are explained in this section. The improvements are measured over the** TERM **baseline. The numbers in bold represent improvements that are statistically significant at** $\alpha = 0.01$

5.5 Case Study: Query Advertisability

The results from Table 2 show that the presence of certain extracted templates and attributes is positively correlated with the click-through-rate (CTR) on sponsored search advertisements. Motivated by these observations, in this section we conduct experiments to validate the usefulness of templates extracted by QTGEN for the problem of predicting *Query-Advertisability* of tail queries [22].

In brief, the search query frequencies are skewed as a power-law and a large fraction of queries are unique (or occur very few times). This makes predicting the CTR of sponsored search advertisements on these queries very challenging. One way is to predict the *Advertisability* for these tail queries in an attempt to help the search engine decide whether to show advertisements for them. In [22] we proposed an approach for this task that was shown to outperform state-of-the-art baselines. Due to paucity of space we refer the reader to the original paper [22] for details of the proposed approach. Here we only describe how we enhance this approach using query templates learnt by QTGEN.

EXPERIMENTAL SETUP.

The approach proposed in [22] is based on learning word-specific scores that are then combined to predict the query-advertisability; we call this approach TERM. We then learn the query templates for the training set of these tail queries and hence the assignment of each term to an attribute. We then augment the query with these learnt attributes and learn the attribute-specific scores. Our two enhancements of the TERM baseline are as follows:

1) TERM + SMOOTH: Here each term's score is smoothed using the attribute it is assigned to by adding (a fraction of) the attribute

impression and click counts to the term's. This is likely to help for rare terms and in cases where very little data is available to reliably estimate term-specific scores. The weight to be given to the attribute impressions is tuned on a validation set and varies with number of training data points. Finally, only the smoothed term-specific scores are then combined into the query-advertisability score.

2) TERM + SMOOTH + QT: In this baseline we perform smoothing as above. In addition, however, we combine term-specific scores as well as attribute-specific scores to learn the query-advertisability score. The methodology for combining these scores is exactly the same as in [22].

As before, in order to remove the effects of particular topics and obtain robust results, the experiments were performed on 43793, 83387, and 15050 tail queries from the Automobiles, Travel, and Movies domains, respectively, randomly sampled from the Yahoo! query logs. Most of these queries occurred once with less than 5% occurring twice. Each of the approaches orders the queries in terms of their predicted advertisability scores and we report the AUC [12] of the clicks-vs-impressions plots (see Figure 8). Queries in each domain were evaluated separately since the presence of exclusive templates makes the predicted scores a little difficult to compare. The AUC values reported in Table 3 are weighted average across the domains, with tests in each domain averaged over 100 runs with randomly selected training points. 40% of queries in each domain were put into the training set, 10% for tuning the SMOOTH parameter, and rest for testing. Care was taken to ensure that the queries used in the test data had occurred later in time than the ones used for training and validation. To simulate low-data situations we sub-sampled the training data to 2%, 5%, 10%, and 50%.

RESULTS AND DISCUSSION.

The averaged AUC results are in Table 3 and results from the Travel domain are plotted in Figure 8. The main points to note are that we get huge percentage improvements from the addition of templates found by QTGEN in situations when very little training data is present. This is because attribute-specific scores are estimated over a larger number of terms making them more robust than term-specific scores, and using them to smooth the latter improves accuracy significantly. Moreover, note that even the presence of certain attributes (SMOOTH + QT) gives the algorithm an additional accuracy boost over just using SMOOTH; this was as expected from the observations in Table 2. While the improvements are overwhealming for smaller training set sizes, the effect remains until at least 50% of the training data is available.

To conclude, the main goal of this experiment was for us to verify that QTGEN finds good query template assignments by showing that these assigned attributes help in estimating query-advertisability. As the plots in Figure 8(a) and 8(b) show, the very strong baseline [22] is essentially random in very low-data situations. However, the accuracy significantly improves with the addition of query templates. Moreover, even in settings with more data the benefit of using query templates persists.

6. SUMMARY

In this paper we focused on the goal of automatically enriching short keyword search queries by finding domain-specific query templates in a completely unsupervised manner. More specifically, we gave a generative model based approach that finds query templates as well as query attributes and their vocabularies, without any human intervention. We empirically demonstrated the performance of our approach by comparing it against two state-of-the-art approaches on real datasets. Finally, we showed an application of query templates to computational advertising. In particular, we significantly improved the performance of an approach for predicting query advertisability by supplementing query keywords with their attributes and template information.

7. REFERENCES

[1] Mallet. `http://mallet.cs.umass.edu/`.

[2] Powerset. `wikipedia.org/wiki/Powerset_(company)`.

[3] G. Agarwal, G. Kabra, and K. C.-C. Chang. Towards rich query interpretation: walking back and forth for mining query templates. In *19th WWW*, 2010.

[4] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman, Boston, MA, 1999.

[5] S. Bergsma and Q. I. Wang. Learning noun phrase query segmentation. In *EMNLP-CoNLL*, 2007.

[6] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3, 2003.

[7] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36, 2002.

[8] A. Broder, M. Ciaramita, M. Fontoura, E. Gabrilovich, V. Josifovski, D. Metzler, V. Murdock, and V. Plachouras. To swing or not to swing: Learning when (not) to advertise. In *17th CIKM*, 2008.

[9] A. Z. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *30th SIGIR*, 2007.

[10] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1), 2001.

[11] S. Goel, A. Broder, E. Gabrilovich, and B. Pang. Anatomy of the long tail: ordinary people with extraordinary tastes. In *3rd*, 2010.

[12] M. Gonen. Receiver operating characteristic (ROC) curves. *SAS Users Group International (SUGI)*, 31, 2006.

[13] T. Griffiths. Gibbs sampling in the generative model of latent dirichlet allocation. Technical report, Stanford University, 2002.

[14] T. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101, 2004.

[15] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. In *32nd SIGIR*, 2009.

[16] Q. Guo and E. Agichtein. Exploring mouse movements for inferring query intent. In *31st SIGIR*, 2008.

[17] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2, 1985.

[18] A. Jain, U. Ozertem, and E. Velipasaoglu. Synthesizing high utility suggestions for rare web search queries. In *34th SIGIR*, 2011.

[19] Y. Li, B.-J. P. Hsu, C. Zhai, and K. Wang. Unsupervised query segmentation using clickthrough for information retrieval. In *34th SIGIR*, 2011.

[20] M. Paşca. Weakly-supervised discovery of named entities using web search queries. In *16th CIKM*, 2007.

[21] S. Pandey and K. Punera. Unsupervised extraction of template structure in web search queries. `www.ideal.ece.utexas.edu/~kunal/papers/querytemplates_long2011.pdf`.

[22] S. Pandey, K. Punera, M. Fontoura, and V. Josifovski. Estimating advertisability of tail queries for sponsored search. In *SIGIR*, 2010.

[23] P. Pantel and A. Fuxman. Jigs and lures: Associating web queries with structured entities. In *ACL*, 2011.

[24] K. Punera and S. Merugu. The anatomy of a click: modeling user behavior on web information systems. In *19th CIKM*, 2010.

[25] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *SIGMOD*, 2010.

[26] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining multiple partitions. *JMLR*, 3, 2002.

[27] I. Szpektor, A. Gionis, and Y. Maarek. Improving recommendation for long-tail queries via templates. In *20th WWW*, 2011.

[28] L. Tang, S. Rajan, and V. K. Narayanan. Large scale multi-label classification via metalabeler. In *18th WWW*, 2009.

[29] X. Wang, D. Chakrabarti, and K. Punera. Mining broad latent query aspects from search sessions. In *15th KDD*, 2009.

[30] G. Xu, S.-H. Yang, and H. Li. Named entity mining from click-through data using weakly supervised latent dirichlet allocation. In *15th KDD*, 2009.