ABSTRACT
Slot-filling refers to the task of annotating individual terms in a query with the corresponding intended product characteristics (product type, brand, gender, size, color, etc.). These characteristics can then be used by a search engine to return results that better match the query’s product intent. Traditional methods for slot-filling require the availability of training data with ground truth slot-annotation information. However, generating such labeled data, especially in e-commerce, is expensive and time consuming because the number of slots increases as new products are added. In this paper, we present distant-supervised probabilistic generative models, that require no manual annotation. The proposed approaches leverage the readily available historical query logs and the purchases that these queries led to, and also exploit co-occurrence information among the slots in order to identify intended product characteristics. We evaluate our approaches by considering both how they affect retrieval performance, as well as how well they classify the slots. In terms of retrieval, our approaches achieve better ranking performance (up to 156%) over Okapi BM25. Moreover, our approach that leverages co-occurrence information leads to better performance than the one that does not on both the retrieval and slot classification tasks.

1 INTRODUCTION
Online shopping accounts for an ever growing portion of the total retail sales. As such, developing methods to correctly understand the customers’ search requirements is an important challenge in e-commerce. Various approaches have been developed to address this challenge, which includes query classification [2], boosting intent-defining terms in the query [16, 17], to name a few. Another way to help customers find what they are searching for is to analyze a customer’s product search query in order to identify the different product characteristics that the customer is looking for, i.e., intended product characteristics such as product type, brand, gender, size, color, etc. For example, for the query “nike men black running shoes”, the term nike describes the brand, men describes the gender, black describes the color and the terms running and shoes describe the product type. Once the query’s intended product characteristics are understood, they can be used by a search engine to return results that correspond to the products whose attributes match these intended characteristics, or as a feature to various Learning to Rank methods [9]. Slot-filling refers to the task of annotating individual terms in a query with the corresponding intended product characteristics, where each product characteristic is a key-value pair, e.g., {key : brand, value : Nike Inc.}. Figure 1 illustrates the role of slot-filling in understanding the query’s product intent.

Figure 1: Query understanding with slot-filling.
Slot-filling can be thought of as an instance of entity resolution or entity linking, which is the problem of mapping the textual mentions of entities to their respective entries in a knowledge base. In the case of search queries, these entities are the predefined set of slots (key-value pairs). Slot-filling is traditionally being treated as a word sequence labeling problem, which assigns a tag (slot) to each word in the given input word sequence. Traditional approaches require the availability of tagged sequences as the training data [6, 7, 11–13, 20, 21, 23, 27, 33, 34, 36–38, 40, 41, 43]. However, generating such labeled data is expensive and time-consuming because e-commerce is a dynamic domain, where new products continue to be added to the inventory bringing new product-types, brands, etc., into the picture, which leads to a continuously evolving query terms and slot-values. Therefore, labeling datasets or designing heuristics to generate labeled data for e-commerce is not a one-time thing but has to be done continuously, making the task even more tedious. To overcome this problem, approaches have been developed that work in the absence of labeled data [5, 24, 37, 44]. However, these approaches are either specific to the domain of spoken language understanding or assume that the words are very similar to the source of distant-supervision [22]. The key insight that we exploit is that in most cases, if a particular query term is associated with products that have a specific value for a slot, e.g., {brand : Nike Inc}, then there is a high likelihood that query term will have that particular slot. Hence, the slots for a query are a subset of the characteristics of the engaged products for that query. Since our approaches are distant supervision-based, they do not need any information about this subset or mapping of the slots to query words. Moreover, they also leverage the co-occurrence information of the product characteristics to achieve better performance.

Distant-supervised slot-filling for e-commerce queries

Saurav Manchanda
University of Minnesota
Twin Cities, MN, USA
manch043@umn.edu

Mohit Sharma∗
WalmartLabs
Sunnyvale, CA
sharm165@umn.edu

George Karypis
University of Minnesota
Twin Cities, MN, USA
karypis@umn.edu

Work was done when Mohit was at WalmartLabs.
1https://www.census.gov/programs-surveys/arts.html
To the best of our knowledge, our work is first of its kind that leverages engagement data in search logs for slot-filling task. We evaluated our approaches on their impact on the retrieval and on correctly predicting the slots. In terms of the retrieval task, our approaches achieve up to 156% better NDCG performance than the Okapi BM25 similarity measure. Additionally, our approach leveraging the co-occurrence information among the slots gained ≈ 3.3% improvement over the approach that does not leverage the co-occurrence information. With respect to correctly predicting the slots, our approach leveraging the co-occurrence information gained 8% performance improvement over the approach that does not leverage the co-occurrence information. Therefore, the proposed approaches provide an easy solution for slot-filling, by only using the readily available search query logs.

2 RELATED WORK

The prior research that is most directly related to the work presented in this paper spans the areas of slot filling, entity linking, and credit attribution.

Slot-filling: Slot-filling is a well-researched topic in spoken language understanding, and involves extracting relevant semantic slots from a natural language text. Popular approaches to slot-filling include markov chain methods, conditional random fields and recurrent neural networks. Generative approaches designed for the slot-filling task includes the ones based on hidden markov models and context free grammar composite models like [13, 23, 34]. Conditional models designed for slot-filling based on conditional random fields (CRFs) include [6, 7, 12, 27, 36–38]. In recent times, recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have been applied to the slot-filling task, and examples of such methods include [11, 20, 21, 33, 38, 40, 41, 43]. A common drawback of these approaches is that they require the availability of tagged sequences as the training data. To tackle absence of the labeled training data, some approaches have been developed. These approaches are either specific to the domain of spoken language understanding or make stronger assumptions about the similarity of the words and the slot values [5, 24, 37, 44]. The unsupervised method developed in [42] is closely related to ours but it requires a manually described grammar rules to annotate the queries and is limited to the task of predicting only two slots (product type and brand) as curating grammar for more diverse slots is a non-trivial task. Furthermore, in e-commerce, new products continue to get added in the inventory bringing new product-types, brands, etc. into picture, which leads to continuously evolving vocabulary. Therefore, designing manual rules to align slot-values to the query words for e-commerce queries requires a continuous manual effort which is expensive.

Entity linking: Entity linking, also known as record linkage or entity resolution, involves mapping a named-entity to an appropriate entry in a knowledge base. Some of the earliest work on entity-linking [1, 3] rely upon the lexical similarity between the context of the named-entity and the features derived from Wikipedia page. The most relevant task to the problems addressed in this paper is the optional entity linking task [8, 18], in which the systems can only use the attributes in the knowledge base; this corresponds to the task of updating a knowledge base with no ‘backing’ text, such as Wikipedia text. However, our setup is more strict because of the absence of availability of entity attributes and lack of context as most e-commerce queries are concise.

Credit attribution: A document may be associated with multiple labels but all the labels do not apply with equal specificity to the individual parts of the document. Credit attribution problem refers to identifying the specificity of labels to different parts of the document. Various probabilistic and neural-network based approaches have been developed to address the credit-attribute problem, such as Labeled Latent Dirichlet Allocation (LLDA) [25], Partially Labeled Dirichlet Allocation (PLDA) [26], Multi-Label Topic Model (MLTM) [30], Segmentation with Refinement (SEG-REFINE) [14], and Credit Attribution with Attention (CAWA) [15]. Out of these methods, the ones that are relevant to the methods presented in this paper are the probabilistic graphical approaches (LLDA, PLDA and MLTM), that use topic modelling to associate individual words/sentences in a document with their most appropriate labels. A common problem with these approaches is that they model the documents as a distribution over the topics and capture the document-level word co-occurrence patterns to reveal topics. These approaches, thus suffer from the severe data sparsity in short documents [39]. Therefore, these approaches are not suitable for e-commerce queries which tend to be very short.

3 DEFINITIONS AND NOTATIONS

Let \( V \) be the vocabulary that corresponds to set of distinct terms that appear across the set of queries \( Q \) in a website. The query \( q \) is a sequence of \( |q| \) terms from the vocabulary \( V \), that is, \( q = \langle q_1, \ldots, q_\text{product} \rangle \). Each query \( q \) is associated with \( q_\text{product} \) product characteristics (slots), also referred as product intent in e-commerce. The collection of \( q_k \), \( V_q \) is denoted by \( A \). Each slot is a key-value pair (e.g., \textit{brand: Nike Inc.}), let \( M \) denote the set of all possible slots and \( L \) denote the set of all possible slot-keys (e.g., product-type, brand, color etc.). Let \( y_q \) be the slot associated with the term \( q_q \) in query \( q \) and \( y_q \) denotes the sequence of slots for the terms in the query \( q \), i.e., \( y_q = \langle y_q_1, \ldots, y_q_\text{product} \rangle \). The collection of \( y_q \), \( V_q \) is denoted by \( Y \). Given query \( q \) let \( c_q \) be the set of slots that is extracted from the characteristics of the products that any user issue \( q \) engaged with. The product intent \( a_q \) is a subset of \( c_q \). The set \( c_q \) (and hence \( a_q \)) is constrained to have at most one slot for each unique slot-key, i.e., \( c_q \) cannot contain multiple brands, product-types etc. The collection of \( c_q \), \( V_q \) is denoted by \( C \). Let \( I \) denote the collection of all possible candidate slots. Throughout the paper, the superscript \(- (q, i)\) on a collection symbol denotes that collection excluding the position \( i \) in query \( q \). For example, \( Y^{- (q, i)} \) is the collection of all slot sequences, excluding the one at the position \( i \) of query \( q \). Similarly, the superscript \(- (q)\) on a collection symbol denotes that collection excluding the query \( q \). Star \((\ast)\) in place of a symbol indicates a collection with star \((\ast)\) taking all possible values. For example, \( \phi_i \) denotes the collection of \( \phi_i \) in Table 1 provides a reference for the notation used in the paper.

The discrete uniform distribution, denoted by \( \text{Uniform}(\cdot | u) \), has a finite number of values from the set \( u \); each value equally likely to be observed with probability \( 1/|u| \). The Dirichlet distribution, denoted by \( \text{Dir}(\cdot | \lambda) \), is parameterized by the vector \( \lambda \) of positive reals. A k-dimensional Dirichlet random variable \( x \) can take values in the \((k-1)\) simplex. The categorical distribution, denoted by \( \text{Cat}(\cdot | \lambda) \), is a discrete probability distribution that describes the possible results of
The Dirichlet distribution is the conjugate prior of the categorical distribution, denoted by \( \text{Bern} \), the probability of each category separately specified in the vector \( \omega \).\( \text{Bernoulli} \) parameter for selecting/rejecting a slot from \( \omega \), \( \Omega \) collection of \( \omega \), \( \Omega \) collection of \( \omega \), \( \gamma \) collection of \( \gamma \) product type, \( \delta \) collection of \( \delta \) distribution for the query \( q \), \( \xi \) collection of \( \xi \) probability distribution of the product categories in the query \( \phi \) collection of \( \phi \) probability distribution of the product categories in the query \( \chi \) collection of \( \chi \) probability distribution of the slots in the product category \( \kappa \) collection of \( \kappa \) probability distribution of the slots in the product category \( \psi \) collection of \( \psi \) probability distribution of the query words in the slot \( m \). \( \nu \) collection of \( \nu \) probability distribution of the transition probability from the slot \( m \).

Table 1: Notation used throughout the paper.

| Symbol | Description |
|--------|-------------|
| \( V \) | Vocabulary (set of all the terms). |
| \( q \) | A query. |
| \( Q \) | Collection of all the queries \( q \). |
| \( z_{q} \) | The product category of the query \( q \). |
| \( Z \) | Collection of \( z_{q}, q \). |
| \( a_{q} \) | The slot-set for the query \( q \). |
| \( A \) | Collection of \( a_{q}, q \). |
| \( c_{q} \) | The candidate slot-set for the query \( q \). |
| \( C \) | Collection of \( c_{q}, q \). |
| \( l \) | Collection of all possible candidate slot-sets. |
| \( \omega_{q} \) | A set with \( \omega_{q,i} = 1 \) if the candidate slot \( c_{q,i} \) is present in the slot-set \( a_{q} \), and 0 otherwise. |
| \( \Omega \) | Collection of \( \omega_{q}, q \). |
| \( y_{q} \) | The sequence of slots for the terms in the query \( q \). |
| \( Y \) | Collection of \( y_{q}, q \). |
| \( K \) | Set of all the product categories. |
| \( M \) | Set of all the possible slots (product characteristics). |
| \( L \) | Set of all the possible slot-keys. |
| \( \alpha \) | Dirichlet prior for sampling the product categories. |
| \( \beta \) | Set of \( |K| \) vectors, \( \beta_{k} \) denoting the Dirichlet prior for sampling the slots from the product category \( k \). |
| \( \gamma \) | Bernoulli parameter for selecting/rejecting a slot from \( c_{q} \). |
| \( \delta \) | Set of \( |M| \) vectors, \( \delta_{m} \) denoting the Dirichlet prior for the query words from the slot \( m \). |
| \( \xi \) | Set of \( |M| \) vectors, \( \xi_{m} \) denoting the Dirichlet prior for transitioning from the slot \( m \). |
| \( \phi \) | Probability distribution of the product categories in the query collection. |
| \( \chi \) | Set of \( |K| \) vectors, \( \chi_{k} \) denoting the probability distribution of the slots in the product category \( k \). |
| \( \psi \) | Set of \( |M| \) vectors, \( \psi_{m} \) denoting the probability distribution of the query words in the slot \( m \). |
| \( \nu \) | Set of \( |M| \) vectors, \( \nu_{m} \) denoting the transition probability distribution from the slot \( m \). |

Algorithm 1 Outline of the generative process for slot-filling

1. for each query \( q \in Q \) do
2. Generate \( c_{q} \) from \( I, a_{q} \) from \( c_{q} \), \( y_{q} \) from \( a_{q} \)
3. for each \( i = 1 \) to \( |q| \) do
4. Generate \( y_{q,i} \) from \( y_{q,i} \)

A random variable that can take on one of \( k \) possible categories, with the probability of each category separately specified in the vector \( \chi \). The Dirichlet distribution is the conjugate prior of the categorical distribution. The posterior mean of a categorical distribution with the Dirichlet prior is given by

\[
x_{i} = \frac{\lambda_{i} + n_{i}}{\sum_{j=1}^{k} (\lambda_{j} + n_{j})}, \quad \forall i \in [1, \ldots, k],
\]

where, \( n_{i} \) is the number of times category \( i \) is observed. The Bernoulli distribution, denoted by \( \text{Bern}(\cdot|p) \) is a discrete distribution having two possible outcomes, i.e., selection, that occurs with probability \( p \), and rejection, that occurs with probability \( 1 - p \).

4 PROPOSED METHODS

In order to address the problem of finding the most appropriate slot for the terms in a query, when the labeled training data is unavailable, we developed generative probabilistic approaches. Our approaches assume that the product intent of a query correspond to a subset of the product characteristics of the engaged products, and leverage the information of the engaged products from the historical search logs as a source of distant-supervision. The training data for our approaches are the search queries and the product characteristics of the engaged products that form the corresponding candidate slot-sets. For example, for the search query "men running shoes", the characteristics of an engaged product are \{product-type: athletic shoes, brand: Nike Inc., color: red, size: 10, gender: mens\}. These characteristics form the candidate slot-set for the search query. Given a query \( q \) and its candidate slot-set \( c_{q} \), our approaches find the most probable slot-sequence \( y_{q} \), such that \( y_{q,i} \in c_{q} \forall i \). Our approaches achieve this by maximizing the joint probability of the observed query words, observed candidate slots and the unobserved slot-sequence.

Our approaches follow some variation of the common generative process outlined in Algorithm 1. The generative process has four unknowns: generating the candidate slot-set \( c_{q} \) from \( I \), generating the product intent \( a_{q} \) from \( c_{q} \), generating the slot sequence \( y_{q} \) from \( a_{q} \) and generating each query word \( a_{q,i} \) from \( y_{q,i} \). To generate \( c_{q,i} \) from \( y_{q,i} \), our approaches model each slot \( m \) as a categorical distribution over the vocabulary \( V \). The categorical distribution for the slot \( m \) denoted by \( \phi_{m} \), is sampled from a Dirichlet distribution with prior \( \delta_{m} \). Our approaches differ in the way they model the other generative steps.

Our first approach, uniform slot distribution (USD), arguably the simplest approach, assumes that \( c_{q} \) is sampled from the \( I \) under a discrete uniform distribution. USD directly generates \( y_{q} \) from \( c_{q} \), thus bypassing generating \( a_{q} \). USD samples each slot \( y_{q,i} \) in the slot-sequence independently from a uniform discrete distribution.

The rest of our approaches relax the independence assumptions of the USD and leverages different ideas to capture the co-occurrence of the slots. Markovian slot distribution (MSD) leverages the idea that two co-occurring slots \( (m_{1} \& m_{2}) \) should have high transition probabilities, i.e., probability of \( m_{2} \) followed by \( m_{1} \) and/or \( m_{1} \) followed by \( m_{2} \) should be relatively high. Correlated uniform slot distribution (CUSD) extends USD so as to sample \( c_{q} \) from \( I \) such that the slots in \( c_{q} \) are more probable to co-occur. To model this co-occurrence, CUSD leverages the idea that the product intent of a search query is sampled from one of the product categories (set of semantically similar product characteristics). As opposed to CUSD which models all the slots in \( c_{q} \) as belonging to the same product category, correlated uniform slot distribution with subset selection (CUSDSS) assumes that a subset of \( c_{q} \) belong to a product category, and this subset corresponds to the actual product intent of the query \( (a_{q}) \). For example, consider a laundry detergent product with the following slots: \{product-type: laundry detergent, brand: tide, color: yellow\}, but we expect that color is not an important attribute when someone is buying laundry detergent.

In the remainder of this section, we discuss these approaches. Because of space constraints, we only provide final learning and inference equations for each of the approaches, and provide detailed generative process and associated derivation in the attached appendix.
We use Gibbs sampling to perform approximate inference. We estimate the parameters iteratively, by sampling a parameter from its distribution conditioned on the values of the remaining parameters.

The parameters that we need to estimate are $\psi$ and $Y$. However, since Dirichlet is the conjugate prior of the categorical distribution, we can integrate out $\psi$ and use collapsed Gibbs sampling [4] to only estimate $Y$. Once the inference is complete, $\psi$ can then be computed from $Y$. Therefore, we only have to sample the slot for each query word from its conditional distribution given the remaining parameters, the probability distribution for which is given by

$$P(Y_{q,i}|Q,C,Y^{−(q,i)},*) \propto \frac{\delta_{y_{q,i}=v_q} + T^{−(q,i)}(y_{q,i}, v_q)}{\sum_{a^q \in V} \delta_{y_{q,i}=a^q} + T^{−(q,i)}(y_{q,i}, a^q)}$$

where, $T(a,b)$ is the count of word $a$ tagged with the slot $b$. The RHS in Equation (2) is exactly the posterior distribution of $\psi_{y_q}$, excluding the current assignment. Once the learning is complete, $\psi_m, \forall m$ can be estimated as in Equation (1).

4.1.2 Inference: The slot for a query word $v_{q,i}$ is simply the slot $m$ for which $\psi_{m,v_{q,i}}$ is maximum, i.e., $y_{q,i} = \arg\max_{m \in \mathcal{C}} \psi_{m,v_{q,i}}$.

4.2 Markovian Slot Distribution (MSD)

Similar to the USD, the markovian slot distribution (MSD) model samples $c_q$ from $I$ under a discrete uniform distribution. But, instead of sampling each slot independent of the other slots, MSD assumes that the slots in the slot sequence $\psi_y$ follow a first order markov process, i.e.,

$$P(y_{q,i}|Q,C,Y^{−(q,i)},*) \propto \frac{\delta_{y_{q,i}=v_q} + T^{−(q,i)}(y_{q,i}, v_q)}{\sum_{a^q \in V} \delta_{y_{q,i}=a^q} + T^{−(q,i)}(y_{q,i}, a^q)}$$

where, $T(a,b)$ is the count of word $a$ tagged with the slot $b$. The RHS in Equation (2) is exactly the posterior distribution of $\psi_{y_q}$, excluding the current assignment. Once the learning is complete, $\psi_m, \forall m$ can be estimated as in Equation (1).

4.1 Uniform Slot Distribution (USD)

The uniform slot distribution (USD) model assumes that $c_q$ is sampled from $I$ under a uniform distribution. Further, each slot is sampled independently from a uniform distribution i.e.,

$$P(y_q|c_q) = \prod_{i=1}^{\mid Q \mid} P(y_{q,i}|c_q) = \prod_{i=1}^{\mid Q \mid} \frac{1}{\mid c_q \mid}.$$

Algorithm 2 and Figure 2(a) shows the generative process for plate notation for the USD, respectively.

4.1.1 Learning: As mentioned before, our approaches each slot as a categorical distribution over the vocabulary $V$. The probability distribution for the slot $m$ is denoted by $\psi_m$, and is sampled from a Dirichlet distribution with prior $\delta_m$. The overall joint probability distribution, according to the USD model is given by:

$$P(Q,Y,C,\psi,\delta) = \prod_{m \in \mathcal{M}} P(\psi_m|\delta_m) \prod_{q \in Q} \frac{1}{\mid c_q \mid} \prod_{i=1}^{\mid c_q \mid} P(y_{q,i}|\psi_{y_{q,i}})$$

$$= \prod_{m \in \mathcal{M}} \text{Dir}(\psi_m|\delta_m) \prod_{q \in Q} \frac{1}{\mid c_q \mid} \prod_{i=1}^{\mid c_q \mid} \text{Cat}(y_{q,i}|\psi_{y_{q,i}})$$

We use Gibbs sampling to perform approximate inference. We estimate the parameters iteratively, by sampling a parameter from its distribution conditioned on the values of the remaining parameters.
Algorithm 4 Generative process for sampling a product category and intended slots from the product category

1. Generate $\phi = \{\phi_1, \ldots, \phi_{|K|}\} \sim \text{Dir}(\cdot | \alpha)$
2. for each product category $k \in K$ do
3. Generate $\chi_k = (\chi_{k,1}, \ldots, \chi_{k,|M|}) \sim \text{Dir}(\cdot | \beta_k)$
4. for each query $q \in Q$ do
5. Generate $z_q \sim \text{Cat}(\cdot | \phi)$
6. for each $i = 1$ to $|c_q|$ do
7. Generate $c_{q,i} \sim \text{Cat}(\cdot | \chi_{c_q})$

\[
= \prod_{m \in M} \text{Dir}(\psi_m | \delta_m) \prod_{m \in M} \text{Dir}(\upsilon_m | \zeta_m)
\prod_{q \in Q} \frac{1}{|I|} \sum_{i=1}^{I} \frac{\text{Cat}(\upsilon_{q,i} | \psi_{y_{q,i}})}{\text{Cat}(\upsilon_{y_{q,i}} | \psi_{q,i})}
\]

Note the important distinction between the MSD model and the usual hidden markov model, that we are only sampling the states (slots) from the $c_q$ and not $M$. Therefore, $P(y_{q,i} | \upsilon_{y_{q,i}}) = \text{Cat}(y_{q,i} | \upsilon_{y_{q,i}})$. This makes integrating out $\upsilon_{q,i}$ difficult. Hence, we don’t collapse out $\upsilon_{q,i}$, but estimate it in course of our learning. The conditional distribution for the slot of a query word, given the remaining parameters, is given by:

\[
P(y_{q,i} | Q, C, Y_{-\{q\}}, \phi, \delta, \upsilon, \chi_q) \propto P_1 \times P_2,
\]

where $P_1$ corresponds to sampling the slot $y_{q,i}$ and equals:

\[
P_1 = \upsilon_{y_{q,i}} \times \upsilon_{y_{q,i}} \times \upsilon_{y_{q,i}} \times \upsilon_{y_{q,i}}.
\]

$P_2$ corresponds to sampling the query word $\upsilon_{q,i}$ from the slot $y_{q,i}$ and equals

\[
P_2 = \frac{\delta_{\upsilon_{q,i}, \upsilon_{y_{q,i}}} + T^{-1}(q,i) (\upsilon_{y_{q,i}, \upsilon_{y_{q,i}}})}{\sum_{\upsilon' \in V} \delta_{\upsilon_{q,i}, \upsilon'} + T^{-1}(q,i) (\upsilon_{y_{q,i}, \upsilon'})}.
\]

Equation (1) is used to estimate $\phi$, $\upsilon$, and $\chi_q$.

4.2.2 Inference: Once we have $\phi$, $\upsilon$, $\delta$, and $\chi_q$, the slot-sequence for a query can be estimated using the Viterbi decoding algorithm [31].

The primary advantage of exploiting the co-occurrence is to perform selection of $c_q$ from $I$, when $c_q$ is not observed. For this, we can use the transition probabilities of the slot sequence as a proxy of how much the slots are likely to co-occur. To achieve this, we estimate the slot sequence, and the corresponding candidate slot set as below:

\[
y_q, c_q = \text{argmax}(P(y_q | c_q)) \cdot \mu P(\hat{y}_q | q, \hat{c}_q); \hat{c}_q \in I,
\]

where, $P(\hat{y}_q | c_q) = \prod_{i=1}^{I} P(\hat{y}_{q,i+1} | y_{q,i})$, $\mu$ is another hyperparameter establishing how much co-occurrence information we want to incorporate and $P(\hat{y}_q | q, \hat{c}_q)$ corresponds to a posteriori estimate.

4.3 Correlated Uniform Slot Distribution (CUSD)

To model the co-occurrence among the slots, CUSD leverages the idea that slots in the candidate slot-set $c_q$ is sampled from one of the product categories. CUSD extends USD by favoring that $c_q$ such that the slots in $c_q$ are more probable of belonging to the same product category. The number of product categories is a hyperparameter denoted by $K$. Hence, sampling $c_q$ from $I$ corresponds to sampling a product category and then sampling the slot set $c_q$ from this product category. This sampling step is modeled as a mixture of unigrams, i.e., $P(c_q | I) = P(z_q) \prod_{i=1}^{K} P(c_{q,i} | z_q)$.

The generative process for sampling the product category and the candidate slots from the product category is outlined in Algorithm 4. The plate notation for CUSD is shown in Figure 2(c). Note that, when $c_q$ is observed (training queries), the slot annotations generated by USD and CUSD are the same.

4.3.1 Learning: We denote the probability of sampling a product category ($P(z_q)$) as $\phi$, and model it as a categorical distribution sampled from a Dirichlet distribution with prior $\alpha$. Each product category is further modeled as categorical distribution over the slots $M$. The probability distribution for the product category $k$ over the slots is denoted by $\chi_k$, and is sampled from a Dirichlet distribution with prior $\beta_k$. For the training data, since the candidate slot sets are observed, the candidate slot set sampling step can be treated independently of the further sampling steps (same steps as the USD). So, we only look into how to estimate the parameters corresponding to the product categories. The overall joint probability distribution of sampling the candidate slot set is:

\[
P(Z, C, \phi, \upsilon, \phi, \chi_q) = P(\phi | \alpha) \prod_{k \in K} P(\chi_k | \beta_k) \prod_{q \in Q} P(z_q | \phi) \prod_{i=1}^{|c_q|} P(c_{q,i} | \chi_q)
\]

\[
= \text{Dir}(\phi | \alpha) \prod_{k \in K} \text{Dir}(\chi_k | \beta_k) \prod_{q \in Q} \text{Cat}(z_q | \phi) \prod_{i=1}^{|c_q|} \text{Cat}(c_{q,i} | \chi_q)
\]

We can integrate out $\phi$, and $\chi_q$, giving the update equations for the product category of a query as:

\[
P(z_q | C, \chi_q | \beta_q) = P_1 \times P_2,
\]

where $P_1$ corresponds to sampling a product category and equals

\[
P_1 = \alpha z_q + \sum_{U \in \mathcal{U}} (q) z_q.
\]

where, $U(z_q)$ is the count of the queries sampled from the product category $z_q$. $P_2$ corresponds to sampling each of the slot in $c_q$ from the product category $z_q$ and equals

\[
P_2 = \frac{\prod_{m \in c_q} (\beta_{z_q,m} + R^{-1}(q) (z_q, m))}{\prod_{m \in M} (\beta_{z_q,m} + R^{-1}(q) (z_q, m')) + 1},
\]

where $R(z_q, m)$ is the count of the slot $m$ sampled from $z_q$.

4.3.2 Inference: The product category for a query $q$ given its candidate slot set $c_q$ can be found using a max (a posteriori) estimate, i.e., $z_q = \text{argmax}_{k} \phi_k \prod_{i=1}^{|c_q|} \chi_{k,c_{q,i}}$. The advantage of CUSD over the USD is to perform selection of $c_q$ from $I$, when $c_q$ is not observed. For this, we can use the probability of the candidate slot-set $c_q$ of being sampled from the product category $z_q$ as a proxy of how much the candidate slots are likely to co-occur. To achieve this, we estimate the slot sequence, and the corresponding candidate slot set as below:

\[
y_q, c_q = \text{argmax}(P(\hat{c}_q, z_q)) \cdot \mu P(\hat{y}_q | q, \hat{c}_q); \hat{c}_q \in I.
\]
Algorithm 5 Generative process for CUSDSS

1. Generate $\phi = (\phi_1, \ldots, \phi_{|K|}) \sim \text{Dir}(\cdot|\alpha)$
2. for each product category $k \in K$
3. Generate $\gamma_k = (X_{k,1}, \ldots, X_{k,M}) \sim \text{Dir}(\cdot|\beta_k)$
4. for each slot $m \in M$
5. Generate $\phi_m = (\phi_{m,1}, \ldots, \phi_{m,V}) \sim \text{Dir}(\cdot|\delta_m)$
6. for each query $q \in Q$
7. Generate $z_q \sim \text{Cat}(\cdot|\phi)$
8. for each slot $i = 1$ to $|c_q|$
9. Generate $\omega_{q,i} \in [0,1] \sim \text{Bern}(\cdot|y)$
10. for each $i = 1$ to $|q|$
11. if $\omega_{q,i} = 1$
12. Generate $c_{q,i} \sim \text{Cat}(\cdot|\gamma_k)$; Add $c_{q,i}$ to $a_q$
13. else
14. Generate $\neg c_{q,i} \sim \text{Cat}(\cdot|\gamma_k)$
15. for each $i = 1$ to $|q|$
16. Generate $y_{q,i} \sim \text{Uniform}(\cdot|\mu); a_{q,i} \sim \text{Cat}(\cdot|\psi_{y_{q,i}})$

where, $P(q_{z,q}, \bar{q}) = \phi_{z,q} \sum_{i=1}^{|c_q|} \gamma_{z,q,c_{q,i}} \mu$ is another hyperparameter establishing how much co-occurrence information we want to incorporate and $P(y_{q,i} | q, \bar{q})$ corresponds to a posteriori estimate according to USD.

4.4 Correlated Uniform Slot Distribution with Subset Selection (CUSDSS)

CUSD leverages the co-occurrence among the slots by favoring $c_q$ such that the slots in $c_q$ are more probable of belonging to the same product category. However, not all the slots in $c_q$ are related to each other. For example, the query ‘tide detergent liquid’ has the following product intent: [product-type: laundry detergent, brand: tide], but the engaged product with this query can have more slots than the query’s intent, for example [color: yellow], as the catalog records all the possible slots. We expect that the slot color: yellow is not an important attribute when someone is buying detergent. In this case, we expect that the slot color: yellow is not related to the other slots in $c_q$. Therefore, a better way to capture the co-occurrence among the slots is the favor that $a_q$ (product intent of the query $q$) such that the slots in $a_q$ are more probable of belonging to the same product category. We present correlated uniform slot distribution with subset selection (CUSDSS), which leverages this idea. CUSDSS extends CUSD by selecting or rejecting each slot in the set $c_q$ using a Bernoulli coin toss for each slot with parameter $y$. The selected slots make up the set $\omega_q$. For each query $q$, we define the set $\omega_q$ of size $c_q$, with $\omega_{q,i} = 1$ if $c_{q,i} \in a_q$ and 0 otherwise. The collection of all these $\omega_q, q$ is denoted by $\Omega$.

The generative process for CUSDSS is outlined in Algorithm 5 and the plate notation is shown in Figure 2(d).

4.4.1 Learning: Unlike CUSD, we are not sampling the slot-sequence $y_q$ from the observed $c_q$ but from the unobserved $a_q$. Therefore, sampling $a_q$ is not independent of the subsequent sampling steps.

The joint probability distribution according to the CUSDSS is:

$$P(\mathcal{Q}, \mathcal{Y}, Z, C, A, \chi, \psi, \Omega, \omega_{\alpha}, \beta_{\beta}, \gamma, \delta_{\delta}, \xi_{\xi}) = P(\phi | \alpha) \int \prod_{k \in K} P(\gamma_k | \beta_k) \int \prod_{m \in M} P(\phi_m | \delta_m) \int \prod_{m \in M} P(\nu_m | \zeta_m)$$

$$\prod_{q \in \mathcal{Q}} \left( P(\omega_q | y_q) P(q | \phi) \prod_{i=1}^{|q|} P(c_{q,i} | \gamma_k)^{\omega_{q,i}} P(\neg c_{q,i} | \gamma_k)^{1-\omega_{q,i}} \right)$$

$$\prod_{i=1}^{|q|} \left[ \sum_{a_{q,i} \in \mathcal{A}} \text{Cat}(a_{q,i} | \nu_{y_{q,i}}) P(y_{q,i} | \psi_{y_{q,i}}) \right]$$

An important unknown in the above distribution is how to model $P(\neg \gamma_k | \gamma_k)$. We model this by introducing a dummy slot $\neg \gamma$ for every slot $\gamma$. We call this extended slot set as $M$. Hence, each product category $k$ is now a distribution over $|M| = 2 \times |M|$ slots.

$$P(\mathcal{Q}, \mathcal{Y}, Z, C, A, \chi, \psi, \Omega, \omega_{\alpha}, \beta_{\beta}, \gamma, \delta_{\delta}, \xi_{\xi}) = \text{Dir}(\phi | \alpha) \int \text{Dir}(\gamma_k | \beta_k) \int \text{Dir}(\phi_m | \delta_m) \int \text{Dir}(\nu_m | \zeta_m)$$

$$\prod_{q \in \mathcal{Q}} \left( \text{Cat}(\gamma_k | \phi) \int \nu_{y_{q,i}} (1-y) ^{1-\omega_{q,i}} \right)$$

$$\prod_{i=1}^{|q|} \text{Cat}(c_{q,i} | \gamma_k)^{\omega_{q,i}} \text{Cat}(\neg c_{q,i} | \gamma_k)^{1-\omega_{q,i}}$$

$$\prod_{i=1}^{|q|} \left[ \sum_{a_{q,i} \in \mathcal{A}} \text{Cat}(a_{q,i} | \nu_{y_{q,i}}) \text{Cat}(a_{q,i} | \psi_{y_{q,i}}) \right]$$

We can integrate out the $\chi, \psi, \phi, \gamma$. We then perform block Gibbs sampling updating $a_q, z_q$ and $y_{q,i}$ in one step, with the update equation as:

$$P(\mathcal{Y}, z_q, a_q, \omega_{\alpha}, \beta_{\beta}, \gamma, \delta_{\delta}, \xi_{\xi}) \propto P_1 \times P_2 \times P_3 \times P_4 \times P_5$$

where, $P_1$ corresponds to sampling the product category $z_q$ and equals,

$$P_1 = a_{z_q} + \frac{\neg z_q}{U} (z_q)$$

$P_2$ corresponds to sampling slots in $a_q$ from the product category $z_q$ and equals,

$$P_2 = \prod_{m \in \mathcal{A}} (\beta_{z_q,m} + \frac{\neg z_q}{U} (z_q, m)) \prod_{m \in \mathcal{A}} (\beta_{z_q,m} + \frac{z_q}{U} (z_q, m') + 1)$$

$P_3$ corresponds to the subset selection and equals,

$$P_3 = \prod_{i=1}^{|q|} \frac{1}{1+y_{q,i}} y_{q,i} \in a_q - \omega_q, y_{q,i} \notin a_q - \omega_q$$

$P_4$ corresponds to sampling each slot and equals,

$$P_4 = 1 - \frac{1}{|\omega_q|}$$

$P_5$ corresponds to the probability of sampling the query word $a_{q,i}$ given the slot $y_{q,i}$ and equals,

$$P_5 = \frac{\delta_{z_q,a_{q,i}} + \frac{\neg z_q}{U} (a_{q,i}, y_{q,i})}{\sum_{a' \in \mathcal{A}} (\delta_{z_q,a'} + \frac{\neg z_q}{U} (a', y_{q,i}))}$$
Note that, for subset selection, $\gamma$ doesn’t have to necessarily be less than 0.5. This is because of the label-bias problem [10], which leads to preferring small sized $a_\theta$. This can also be seen through Equation (4), which causes $P_4$ to have higher values for smaller $|a_\theta|$. Therefore, $\gamma$ can be more than 0.5 to select the required subset.

4.4.2 Inference: We perform the inference iteratively using block Gibbs sampling in the same manner as learning. Similar to CUSD, we use the probability of the product intent $a_\theta$ of being sampled from $z_q$ as a proxy of how much the slots in the product intent are likely to co-occur. Therefore, in this case, we estimate the slot sequence, and the corresponding candidate slot set as below:

$$
y_q, c_q = \underset{y_q, c_q}{\text{argmax}}(P(\hat{c}_q, \hat{y}_q))P(\hat{y}_q|q, \hat{c}_q)\hat{c}_q \in I,
$$

where $P(\hat{c}_q, \hat{y}_q) = \phi_{z_q} \prod_{i=1}^{|c_q|} x_{\hat{c}_q, z_q}(1 - x_{\hat{c}_q, z_q})^{(1 - \phi_{\hat{c}_q})}$ and $\mu$ controls how much co-occurrence information we want to incorporate.

5 EXPERIMENTAL METHODOLOGY

5.1 User engagement and annotated dataset

We used two datasets to estimate and evaluate our models. We call the first dataset engagement, and it contains pairs of the form $(q, c_q)$, i.e., queries along with their candidate slot-sets. We call the second dataset annotated and it contains ground-truth annotated queries. We first describe our process of generating the engagement dataset and then describe the annotated dataset.

We used the historical search logs of an e-commerce retailer to prepare our engagement dataset. This data contains 48,785 distinct queries, and the size of vocabulary is 1,936. We assume that $c_q$ corresponds to the product characteristics of the products that were engaged for $q$. The product characteristics belong to the following six types (slot-keys) and their number of distinct values is in parenthesis: product-type (983), brand (807), gender (7), color (20), age (86) and size (243). We created the required dataset by sampling query-item pairs from the historical logs spanning 13 months (July 17 to July 18), such that the number of orders of that item for that query over a month is at least five. Note that each query can be engaged with multiple items, all of which were included to create $c_q$. We limited our dataset to the queries whose words are frequent, i.e., each query word occurs at least 50 times. Furthermore, we use the items only if their product characteristics are present in at least 50 items. These filtering steps avoid data sparsity, thus, help estimating models accurately.

The annotated dataset contains a manually annotated representative sample of the queries. The instructions for slot-key annotation along with a few examples illustrating the same were given to annotators. Two annotators tagged each query, and when the annotators disagree over their annotations, they were asked to resolve conflicts through reasoning and discussions. Furthermore, we discarded the queries for which annotators failed to resolve conflicts. In addition to the slot-keys product-type, brand, gender, color, age and size, the set of slot-keys contain an additional tag miscellaneous corresponding to the words which do not fit into either of the other tags. The final annotated dataset contains a total of 3,430 annotated queries.

For reproducibility, the ideal way is to perform training and evaluation on publicly available datasets. However, to the best of our knowledge, there exists no public dataset containing e-commerce queries and their engagement with products having attributes (slot-sets). Hence, we limit our evaluation to the above mentioned proprietary dataset.

5.2 Evaluation methodology

We evaluate our methods on three tasks, as described below:

$T_1$: This task evaluates how well the predicted slots for a query improve the retrieval performance. We calculate the similarity between a query and a product by simply counting the number of predicted slots for the query that match with the characteristics of the product.

In a typical e-commerce ranking framework, the similarity of a query with a product is composed of many features, such as the textual similarity between the query and the product description. We compare our proposed methods against Okapi Best Matching 25 (BM25) [28, 29], which is a text similarity measure extensively used by search engines. We also evaluate the retrieval performance of our methods after incorporating textual similarity of the query with the product description while calculating the similarity between a query and a product. To incorporate BM25 into our ranking function, we simply normalize the BM25 scores between a query and all the products between zero and one, and add the resultant score to the slot-filling score, which is used to rank the products for a query.

To the best of our knowledge, our work is the first attempt at using engagement data for distant-supervised slot-filling in e-commerce queries. This limits availability of baselines to perform a fair evaluation. Moreover, the most closely related unsupervised method [42] uses described grammar to annotate the queries. The grammar developed by the authors work for only two slots (product-type and brand) as developing such grammar for more diverse slots is a non-trivial task. Thus, we limit our baselines to the BM25-based approaches.

$T_2$: This task compares the predicted slots with the manually annotated queries. The candidate slot-set is observed, thus, this task evaluates how well our methods can tag each word in the query with the corresponding slot, given a set of slots to choose from.

$T_3$: This task is similar to the task $T_2$, but the candidate slot-set $c_q$ is not observed for the task $T_3$. Thus, in addition to predicting the tagging performance, this task also evaluates how well our methods are able to correctly choose the candidate slot-set $c_q$. When $c_q$ is not observed, it has to be sampled from the collection $I$, which can be very large (of the order $10^{12}$ for the presented experimental setup). This makes calculating Equations (3) and (5) computationally expensive. Hence, we heuristically decrease the number of candidate slot-sets. For each query-word $a_{q,t}$, we find the top-$t$ most probable slots for each possible slot-key that may have generated $a_{q,t}$, i.e., top-$t$ indices corresponding to $\psi_{a_{q,t}}$, for each possible slot-key. The collection of the candidate slot-sets for a query then becomes equal to the all possible candidate slot-sets based on top-$t$ slots for each query word. In this paper, we use $t = 1$ for simplicity.

Note that a query word can have only one slot-key but multiple slot values leading to annotation ambiguity. To avoid this ambiguity, we relax our evaluation to predict the correct slot-keys for the tasks $T_2$ and $T_3$. However, when the candidate slot-set is observed, predicting the correct slot-key indeed corresponds to predicting the correct slot, as there is only one possible slot-value for a slot-key.
We select the hyperparameters for each metric. We used grid search on validation set to select our hyper-parameters.

5.3 Performance Assessment metrics

For T1, we calculate the Mean Reciprocal Rank (MRR) [32] and Normalized Discounted Cumulative Gain (NDCG) [35] which are popular measures of the ranking quality in information retrieval. To calculate the MRR, we assume that the relevant products for a query are the most purchased products for that query. To calculate the NDCG, we assume that the relevance of a product for a query is the number of times that product was purchased after issuing that query. We calculate NDCG only till rank 10 (NDCG@10), as users usually pay attention to the top few results and break ties as proposed in [19].

For T2 and T3, our first metric, accuracy, calculates the overall percentage of the query words whose slot-key is correctly predicted. Our second metric calculates the percentage of the words whose slot-key is predicted correctly in a query. We report average of this percentage over all the queries and call this metric q-accuracy. Metric q-accuracy treats all queries equally but accuracy is biased towards longer queries. Compared to accuracy, q-accuracy being a transaction-level metric gives a better picture of how our methods impact the traffic. The metrics accuracy and q-accuracy are biased towards the frequently occurring slots-keys. Therefore, we also report macro-averaged precision (avg-prec), recall (avg-rec) and F1-score (avg-F1).

5.4 Parameter selection

We used grid search on validation set to select our hyper-parameters (priors for all probability distributions, number of product categories and the parameter γ). For simplicity, we present our results for uniform Dirichlet priors, i.e., αi = α, βi,j = β, δi,j = δ, ξi,j = ξ, Vi,j. For estimating all our approaches, we ran 1,000 training iterations. We ran 100 sampling iterations for inference of CUSDSS. We select the hyperparameters for each metric (accuracy, q-accuracy, avg-prec, avg-rec, avg-F1) independently, corresponding to the best performance of the metric on the validation set. Note that we perform validation only for the task T3, i.e., when cq is not observed, but we report the results for the task T2 on the same selected hyperparameters. For T1, we use the same hyperparameters corresponding to the q-accuracy metric for T3.

6 RESULTS AND DISCUSSION

Task T1: Table 2 shows the performance of the different approaches for the retrieval task. CUSDSS performs the best of all the models on both metrics, whereas CUSD performs better than USD and MSD when it is combined with BM25. This validates our hypothesis that modeling co-occurrence and subset-selection leads to better performance on slot-filling, which in turn leads to better retrieval. The performance improvement of CUSDSS over USD, using only the predicted slots in the ranking function, is ≈ 5.6% and ≈ 4.2% on the MRR and NDCG metrics, respectively. Upon incorporating the BM25 with the predicted slots in the ranking function, the performance improvement is ≈ 2.3% and ≈ 3.3% on the MRR and NDCG metrics, respectively.

On both the MRR and NDCG metrics, using the predicted slots to rank the products leads to better performance than using BM25. Additionally, using BM25 in conjunction with the predicted slots lead to even better performance. For example, the performance improvement of CUSDSS over BM25, using only the slots in the ranking function, is ≈ 108% and ≈ 156% on the MRR and NDCG metrics, respectively. The performance further improves upon incorporating BM25 with the predicted slots in the ranking function. This is particularly appreciable because the proposed approaches do not require any labeled data, instead they leverage the readily available search query logs to boost the retrieval performance. We also see that the USD and MSD performs almost equally. This is expected as all the queries nike mens running shoes, mens nike running shoes, running mens shoes nike etc. are possible, favoring uniform transition probabilities.

Task T2 and T3: Tables 3 and 4 show the performance for the slot classification when the candidate slot-set is observed and unobserved, respectively. We report average of 100 runs with random initializations. For all the methods, we see that q-accuracy is more than accuracy, which shows that all the methods perform better on shorter queries. Figure 3 shows how the q-accuracy varies with the number of words in the queries. For all the methods, q-accuracy
Table 3: Slot-filling results when $c_q$ is observed (Task T2).

| Model   | accuracy | $q$-accuracy | avg-prec | avg-rec | avg-F1  |
|---------|----------|--------------|----------|---------|---------|
| USD/CUSD | 0.678    | 0.708        | 0.426    | 0.587   | 0.453   |
| MSD     | 0.679    | 0.709        | 0.428    | 0.589   | 0.455   |
| CUSDSS   | 0.657    | 0.686        | 0.571    | 0.493   | 0.465   |

1 The performance of CUSDSS is statistically significant over USD and MSD on the avg-prec and avg-F1 metrics ($p$-value ≤ 0.01 using t-test).

Table 4: Slot-filling results when $c_q$ is unobserved (Task T3).

| Model   | accuracy | $q$-accuracy | avg-prec | avg-rec | avg-F1  |
|---------|----------|--------------|----------|---------|---------|
| USD     | 0.558    | 0.583        | 0.506    | 0.645   | 0.472   |
| MSD     | 0.558    | 0.583        | 0.506    | 0.645   | 0.472   |
| CUSD    | 0.565    | 0.588        | 0.510    | 0.650   | 0.476   |
| CUSDSS  | 0.566    | 0.586        | 0.526    | 0.622   | 0.511   |

1 The performance of CUSD is statistically significant over USD and MSD on the accuracy, $q$-accuracy and avg-rec metrics ($p$-value ≤ 0.01 using t-test).

2 The performance of CUSDSS is statistically significant over USD and MSD on the accuracy, $q$-accuracy, avg-prec and avg-F1 metrics and over CUSD on the avg-prec and avg-F1 metrics ($p$-value ≤ 0.01 using t-test).

Figure 3: Variation of the $q$-accuracy with query length.

decreases with the length of query. Similar to the ranking task T1, we see that USD and MSD performs equally good for both the cases. CUSD performs better than USD and MSD on all the metrics, which shows that selecting $c_q$ with co-occurring slots leads to better performance. Moreover, co-occurrence information is exploited when the query has multiple query words. As shown in Figure 3, CUSD performs considerably better than USD and MSD on the $q$-accuracy metric as the number of words in the queries increases.

Interestingly, CUSDSS performs better than the USD and MSD on the $q$-accuracy and accuracy metrics when the candidate slot-set is observed, but does not perform as good when the candidate slot-set is unobserved. In addition, CUSDSS performs best on the avg-prec and avg-F1 metrics, but not so good on the avg-rec metrics. To investigate this peculiar behavior of CUSDSS, we compared the individual precision, recall and F1 scores for the slot-keys of CUSDSS with the other approaches. Tables 5 and 6 shows these statistics for each of the slot when the candidate slot-set is observed and unobserved, respectively. Subset selection leads to CUSDSS labeling more query words with frequent slots like product-type and brand. This not only decreases the recall of the other less frequent slots (gender, color, age, size) but also decreases the precision of the frequent slots. However, this also tends to make CUSDSS predict only those less frequent slots for which it has high confidence, leading to an increase on the avg-prec metric. Consequently, CUSDSS also performs the best on the avg-F1 metric (3% over the USD when $c_q$ is observed, and 8% improvement when $c_q$ is not observed). When the candidate slot-set is unobserved, there are more possible ways to make an error in slot-prediction, than the case when the candidate slot-set is observed. For the unobserved case, as the prediction is biased towards the many slots, CUSDSS performs better on the micro-averaged metrics, i.e., accuracy and $q$-accuracy. However, for observed candidate slot, CUSDSS does not benefit as the scope of error is small.

Additionally, even though the overall performance of CUSDSS on the $q$-accuracy metric is not at par with the USD, the performance is considerably better for the queries with multiple words, and the difference in performance increases with the number of words in the query, as depicted in Figure 3. When there is only one word in the query, CUSDSS favors frequent slots, leading to degraded performance as there is no co-occurrence information to be leveraged.

To study qualitatively how subset selection helps CUSDSS make better predictions than CUSD. Table 7 shows the most probable slots for some product categories estimated by CUSDSS and CUSD. We manually label the estimated product categories with a relevant semantic name. We observe that the product categories estimated by CUSD contain more slots corresponding to the slot-keys age, gender, size and color, while the product categories estimated by CUSDSS contain more slots corresponding to the slot-keys product-type and brand. For example, the product category houseware for CUSD has age: adult as the most probable slot, while CUSDSS has brand: mainstays as the most probable slot. This follows from the fact that CUSD models the complete candidate slot-sets $c_q$ as belonging to the same product category, which leads to the frequent occurring slots in $c_q$ having high probabilities in the product categories. In contrast, CUSDSS models the actual product-intent $a$ as belonging to the same product category, which leads to the actual intended slots having high probabilities in the product categories.

7 CONCLUSION

In this paper, we used the historical query reformulation logs of e-commerce retailers to develop distant-supervised approaches to perform slot-filling for the e-commerce queries. Our approaches solely depend on the user engagement data and require no manual labeling effort. In terms of retrieval, our approaches achieve better ranking performance (up to 156%) over Okapi BM25. In addition, when used along with Okapi-BM25, our approaches improve the ranking performance by 250%. Our work makes a step towards leveraging distant-supervision for slot-filling, and envision that the proposed method will serve as a motivation for other applications that rely on the labeled training data, which is expensive and time-consuming. To our best knowledge, this paper is the first attempt in only relying upon the engagement data to perform slot-filling for the e-commerce queries. Code accompanying with this paper will be available at Github.
### Table 5: Individual precision (P), recall (R) and F1-score (F1) when \( c_q \) is observed.

| Model     | product-type | brand | gender | color | age | size | miscellaneous |
|-----------|--------------|-------|--------|-------|-----|------|---------------|
| USD/CUSD  | 0.93         | 0.74  | 0.82   | 0.35  | 0.89| 0.51 | 0.63          |
| MSD       | 0.93         | 0.74  | 0.82   | 0.35  | 0.89| 0.51 | 0.63          |
| CUSDSS    | 0.90         | 0.74  | 0.81   | 0.26  | 0.88| 0.40 | 0.88          |

### Table 6: Individual precision (P), recall (R) and F1-score (F1) when \( c_q \) is unobserved.

| Model     | product-type | brand | gender | color | age | size | miscellaneous |
|-----------|--------------|-------|--------|-------|-----|------|---------------|
| USD       | 0.91         | 0.59  | 0.71   | 0.20  | 0.85| 0.35 | 0.81          |
| MSD       | 0.91         | 0.59  | 0.71   | 0.20  | 0.85| 0.35 | 0.81          |
| CUSDSS    | 0.90         | 0.63  | 0.72   | 0.19  | 0.84| 0.32 | 0.83          |

### Table 7: Examples of product categories estimated by CUSD and CUSDSS

| Product category | Most probable slots estimated by CUSD (ranked by probability) | Most probable slots estimated by CUSDSS (ranked by probability) |
|------------------|---------------------------------------------------------------|---------------------------------------------------------------|
| Houseware        | age: adult, genus: unisex, brand: mainstays, size: 1, color: white, color: black, price: better homes & gardens, price: brown, pt: storage chests & boxes, brand: sterilite, color: silver, color: gray | brand: mainstays, pt: storage chests & boxes, pt: vacuum cleaners, brand: sterilite, price: better homes & gardens, price: brown, brand: the pioneer woman, pt: food storage jars & containers, brand: holiday time |
| Toys             | pt: dolls, color: multicolor, price: child, gender: unisex, brand: disney princess, price: doll playsets, gender: girls, age: adult, brand: shopkins, brand: disney, price: stuffed animals & plush toys, | pt: dolls, price: action figures, price: play vehicles, price: disney princess, price: stuffed animals & plush toys, brand: disney, price: interlocking block building sets, brand: lego, brand: paw patrol |
| Dental hygiene   | gender: unisex, price: adult, color: multicolor, pt: toothpastes, price: crest, price: colgate, brand: oral-b, size: 1, color: white, price: adult, color: child, price: 4, pt: mouthwash, price: powered toothbrushes, price: teen, size: 6 | pt: toothpastes, price: crest, price: colgate, brand: oral-b, price: baby foods, price: brand, price: gerber, price: mouthwash, price: meal replacement drinks, price: manual toothbrushes, price: powered toothbrushes, brand: ensure |

### REFERENCES

[1] Razvan Bunescu and Marius Paşca. 2006. Using encyclopedic knowledge for named entity disambiguation. In 11th conference of the European Chapter of the Association for Computational Linguistics.

[2] Huanhuan Cao, Derek Hao Hu, Dou Shen, Daxiong Jiang, Jian-Tao Sun, Enhong Chen, and Qiang Yang. 2009. Context-aware query classification. In Proceedings of the 32nd annual ACM SIGIR conference on Research and development in information retrieval. 3–10.

[3] Silviu Cucerzan and Eric Brill. 2005. Extracting semantically related queries by exploiting user session information. Technical Report. Technical report, Microsoft Research.

[4] Thomas L. Griffiths and Mark Steyvers. 2004. Finding scientific topics. Proceedings of the National academy of Sciences 101, suppl 1 (2004), 5228–5235.

[5] Matthews S. Henderson. 2015. Discriminative methods for statistical spoken dialogue systems. Ph.D. Dissertation. University of Cambridge.

[6] Minwoo Jeong and Gary Gurnee Lee. 2007. Structures for spoken language understanding: A two-step approach. In 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07. Vol. 4. IEEE, IV–141.

[7] Minwoo Jeong and Gary Gurnee Lee. 2008. Practical use of non-local features for statistical spoken language understanding. Computer Speech & Language 22, 2 (2008), 148–170.

[8] Heng Ji, Ralph Grishman, Hoo Trang Dang, Kira Griffitt, and Joe Ellis. 2010. Overview of the TAC 2010 knowledge base population track. In Third Text Analysis Conference (TAC 2010). Vol. 3. 3–5.

[9] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On application of learning to rank for e-commerce search. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 475–484.

[10] John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[11] Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. arXiv preprint arXiv:1609.01454 (2016).

[12] Jingjing Liu, Scott Cypher, Panupong Pasupat, Ian McGraw, and James Glass. 2012. A conversational movie search system based on conditional random fields. In Thirteenth Annual Conference of the International Speech Communication Association.

[13] Klaus Macherey, Franz Josef Och, and Hermann Ney. 2001. Natural language understanding using statistical machine translation. In Seventh European Conference on Speech Communication and Technology.

[14] Saurav Manchanda and George Karypis. 2020. CAWA: An Attention-Network for Credit Attribution. In AAAI. 8472–8479.

[15] Saurav Manchanda, Mohit Sharma, and George Karypis. 2019. Intent term selection and refinement in e-commerce queries. arXiv preprint arXiv:1908.0856 (2019).

[16] Saurav Manchanda, Mohit Sharma, and George Karypis. 2019. Intent term weighting in e-commerce queries. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management. 2345–2348.

[17] Paul McNamee and Hou Trang Dang. 2009. Overview of the TAC 2009 knowledge base population track. In Text Analysis Conference (TAC), Vol. 17. National Institute of Standards and Technology (NIST) Gaithersburg, Maryland . . . , 111–113.

[18] Frank McSherry and Marc Najork. 2008. Computing information retrieval performance measures efficiently in the presence of tied scores. In European conference on information retrieval. Springer, 414–421.

[19] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tür, Xiaodong He, Larry Heck, Golkan Tur, Dong Yu, et al. 2015. Using Recurrent Neural Networks for Slot Filling in Spoken Language Understanding. IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING 23, 3 (2015).

[20] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In Interspeech. 3771–3775.
[22] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AISEP. Volume 2-Volume 2. Association for Computational Linguistics, 1003–1011.

[23] Roberto Pieraccini, Evelyne Tourkermann, Zakhar Gorelov, J-L Gauvain, Esther Levin, C-H Lee, and Jay G Wilpon. 1992. A speech understanding system based on statistical representation of semantics. In [Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 1. IEEE, 193–196.

[24] Stephen Della Pietra, Mark Epstein, Salim Roukos, and Todd Ward. 1997. Fertility algorithms for spoken language understanding. In Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 168–173.

[25] Daniel Ramage, David Hall, Ramen Nallapati, and Christopher D Manning. 2009. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing; Volume 1-Volume 1. Association for Computational Linguistics, 248–256.

[26] Daniel Ramage, Christopher D Manning, and Susan Dumais. 2011. Partially labeled topic models for interpretable text mining. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 457–465.

[27] Christian Raymond and Giuseppe Riccardi. 2007. Generative and discriminative algorithms for spoken language understanding. In Eighth Annual Conference of the International Speech Communication Association.

[28] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poission model for probabilistic weighted retrieval. In SIGIR '94. Springer, 232–241.

[29] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at TREC-3. Nist Special Publication Sp (1995).

[30] Hossein Soleimani and David J Miller. 2017. Semisupervised, multilabel, multi-instance learning for structured data. Neural computation 29, 4 (2017), 1053–1102.

[31] Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE transactions on Information Theory 13, 2 (1967), 260–269.