Towards a simplified ontology for better e-commerce search

Aliasgar Kutiyanawala
Jet.com/Walmart Labs
Hoboken, New Jersey
aliasgar@jet.com

Prateek Verma
Jet.com/Walmart Labs
Hoboken, New Jersey
prateek.verma@jet.com

Zheng (John) Yan
Jet.com/Walmart Labs
Hoboken, New Jersey
john@jet.com

ABSTRACT

Query Understanding is a semantic search method that can classify tokens in a customer's search query to entities such as *Product*, *Brand*, etc. This method can overcome the limitations of bag-of-words methods but requires an ontology. We show that current ontologies are not optimized for search and propose a simplified ontology framework designed specifically for e-commerce search and retrieval. We also present three methods for automatically extracting product classes for the proposed ontology and compare their performance relative to each other.

CCS CONCEPTS

• **Computing methodologies → Information extraction**; **Ontology engineering**;

KEYWORDS

Ontology Creation, Information Retrieval, E-Commerce, Query Understanding

ACM Reference Format:
Aliasgar Kutiyanawala, Prateek Verma, and Zheng (John) Yan. 2018. Towards a simplified ontology for better e-commerce search. In *Proceedings of SIGIR 2018 Workshop on eCommerce (SIGIR eCom 2018).* ACM, New York, NY, USA, Article 4, 7 pages.

1 INTRODUCTION

Search plays a vital part in any e-commerce site and a poor search system leads to customer frustration, which negatively affects both retention and conversion. Most e-commerce sites employ a bag-of-words search method which simply matches tokens in a customer's search query with relevant fields of SKUs (stock keeping unit but used here to describe any item sold by the site). This system is easy to implement specially with solutions like ElasticSearch [10] or Solr [11] but suffers from some significant drawbacks. This system is prone to returning irrelevant results because of its inability to understand what the customer is looking for. Consider an example search query: "`men's black leather wallet`" and let us assume that there are no SKUs that match this query exactly. The bag-of-words system will resort to a partial match and may return men's brown leather wallets (relevant) along with men's black leather belts (irrelevant). This problem is also evident when queries are similar in terms of words but actually relate to very different products. For example: the queries "`camera with lens`" and "`lens for camera`" may produce the same result if prepositions are ignored as stopwords. There are ways to augment the bag-of-words search system with a category pinpointing (or prediction) model, bigrams, etc. to improve the recall. However, this approach requires a deep category tree with leaf nodes as product types and an accurate categorization model, both of which could be an issue given a large catalog.

A better approach to search is to use a query understanding system to understand the customer's search *intent* [13, 24]. One such method is to use a semantic annotation process described in [9, 23] by using a well-defined ontology to classify terms from the customer's search query. Going back to our previous example, if we were to classify tokens in "`men's black leather wallets`" as **`men := Gender, black := Color, leather := Material, wallet := Product`**, it would allow the system to find exactly what the customer is looking for or make relevant substitutions if no such SKU could be found. This task is called *Named Entity Recognition and Classification* (NERC), where entities like Product, Color, Material, etc. are recognized. Nadeau and Sekine [20] provide an excellent overview of this field. We use Bi-directional LSTM-CRF as described by Lample et al. in [14] for performing named entity recognition although other systems like GATE [4, 5] could also be used. The named entity tagger can accurately recognize the customer's intent by recognizing and classifying entities in the query as long as those entities are well-defined. The problem is that most existing product ontologies are designed from a supply-side perspective and not from a search perspective.

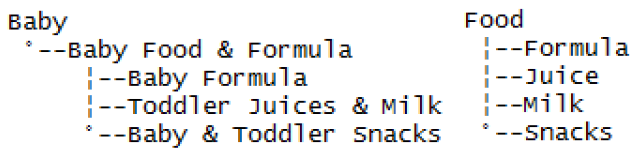
We propose a simplified ontology framework specially designed from a search and retrieval perspective that contains three top-level *concepts* - *Product*, *Brand* and *Attribute* and five *slots* (or *properties*) - *synonyms*, *attributes*, *primary_attributes*, *brands* and *default_product*. We show that these three entity classes along with five slots can provide relevant recall for a customer's search query. We further discuss this ontology in Section 2 and provide insights into why each entity type and slot is necessary and how they help in retrieving relevant results.

Our contributions in this paper are creating a product ontology designed specifically for search and providing three methods to automatically extract *Product concepts* for this ontology. We discuss this ontology in detail in Section 2. We provide an overview of the field of Ontology learning in Section 3 and discuss our methods to extract *Product concepts* in Section 4. Finally, we present our conclusions in Section 5.

2 ONTOLOGY

An ontology is a formal explicit description of a domain by identifying *classes* (or *concepts*), *slots* and *slot restrictions* between classes for a particular domain [21]. *Classes* represent the main concepts in a domain and are related to physical objects in that domain, for example: *TV*, *Shirt* or *Screen Size*. *Slots* represents properties of objects and relationships between classes, for example: the slot *attribute* links the classes *TV* and *Screen Size*. *Slot Restrictions* impose restrictions on the values that can be taken by a slot, for example: we can impose the restriction that *Screen Size* is a positive number.

Our goal is to design a product ontology that can be used for search purposes. This ontology must serve a dual purpose - we must be able to classify SKUs onto this ontology and secondly, the classes (and subclasses) in this ontology should serve as named entities for query-side named entity recognition and classification. There are many supply-side ontologies for e-commerce like ecl@ss, Harmonised System, NAICS/NAPCS, RosettaNet, etc. [6] but they tend to focus more on relationships between buyers and sellers. They tend to include slots (or properties) such as *GLN of manufacturer*, *GLN of supplier*, *product article number of supplier*, etc. which are completely unnecessary for search purposes. These ontologies also have product types that are very complex, for example the entire phrase: *"Shirts, underwear, men's and boys', cut and sewn from purchased fabric (except apparel contractors)"* is a product from NAICS. Such product types contain attributes (like men's, boy's, etc.) along with the most basic form of the product (shirt) and hence are not considered *atomic*. The NERC system will have a lot of difficulty in using such *non-atomic* products.



```
Baby                                      Food
 °--Baby Food & Formula                    ¦--Formula
      ¦--Baby Formula                      ¦--Juice
      ¦--Toddler Juices & Milk             ¦--Milk
      °--Baby & Toddler Snacks             °--Snacks
```

Figure 1: Comparison of catalog-side and search-side ontologies.

Work has also been done on ontologies that are focused more on the catalog side [2, 15]. Catalog-side ontologies are closer to search-side ontologies as compared to supply-side ontologies but are still not perfectly aligned with a search perspective. Consider Figure 1, which shows a snippet of *Product* classes from two ontologies - a catalog-side ontology on the left and a search-side ontology on the right. There are three main differences between them. The first difference is that the ontology on the left does not have a *"is a"* relationship between classes and subclasses. For example: *Baby food and formula* is not a *Baby*. The ontology on the left tries to classify items by their intended use case but ontology on the right classifies items according to what they represent. The second difference is that the ontology on the left contains combo products like *Toddler Juices and Milk*, which makes it difficult to know if a SKU classified to this product type is a *Juice* or *Milk*. The third difference is that the ontology on the left contains non-atomic entries like *Baby and Toddler Snacks*, which should just be simplified to *Snacks* as it makes it very easy for the NERC system to identify products in queries like "`snacks for baby`."

Our ontology contains a restriction that requires all classes (and subclasses) to be as *atomic* as possible to improve recall. We define an *atomic* entity as an irreducible unit that describes a concept. It also places a *"is-a"* requirement on all subclasses for a given class. Finally, it tries to avoid combo classes unless they are sold as a set (*dining sets* that must contain both *table* and *chairs*). This requirement keeps the ontology simple and flexible. The following sections describe the classes and slots in our ontology in greater detail.

2.1 Product

A *Product* is defined as the *atomic* phrase that describes what the customer is looking for. Consider an example, "`white chair with ottoman`". Here, the customer is looking to buy a *chair*. It is preferable if the chair is white in color and comes with an ottoman but these requirements are secondary to the primary requirement of it being a *chair*. If such a chair is not available, the customer is more likely to buy a chair in a different color or one that does not come with an ottoman but is less likely to buy a white sofa with ottoman even though it satisfies two requirements out of three. Any specialized product type like *folding chair* must be stripped down to its most basic form *chair*. There are exceptions to this rule, for example, a *bar stool* is a specialized type of stool and ideally we should strip it down to its most basic form *stool* but many customers use the term "`barstool`" (single term without spaces) to describe it. The NERC system has to be able to classify this term to a product and hence we include the term "`barstool`" as a Product in our ontology with the synonym "`bar stool`". The class *barstool* is a sub-class of the class *stool* because every barstool is ultimately a stool. This parent-child relationship also helps during recall because if the customer searches for "`stool`", the search system will include all stools including barstools in the recall. It should be noted that *atomic* does not imply a single-word token because many multi-word tokens like *air conditioner* and *onion rings* are *atomic*. We use a combination of our query and SKU understanding systems along with user data to provide suggestions for parent-child relationships and synonyms (or variations). However, describing this method is beyond the scope of this paper.

2.2 Attribute

Attribute is defined as an *atomic* phrase that provides more information about an item. Consider an example "`white wooden folding adirondack chair`". Here, we classify the term *chair* as a *Product* and we can classify the remaining terms (white, wooden, folding and adirondack) as *Attributes*. This gives us a lot of flexibility during recall. Initially, the search system can restrict the recall by filtering out any SKUs that do not match the product type and then boost SKUs by the number of matching attributes. In case of our example, we would restrict the recall to be chairs of all types and then boost those SKUs that match the attributes (white, wooden, folding and adirondack). A SKU that matches all attributes will have a higher score (and placed on top of the recall) than those that match fewer attributes.

Attributes can be subclassed as *Color*, *Material*, *SleeveType*, etc. depending on the category. We found that only a subset of Attributes are relevant for search purposes. An attribute like *Country of Manufacture* may be a valid subclass but it can be argued that it is not very important for search purposes. Since our aim is to create a *simplified* ontology for search, we restrict attribute subclasses to what is actually important for search. This makes the system much more maintainable. The range of most attributes are values from an enumerated set but some attributes like *Screen Size* may have numeric values along with a unit of measurement like *inches*, *cm*, etc. Such numeric values can be normalized using simple rules (*1 inch = 2.54 cm*) so that more relevant SKUs can be recalled for a given query even if they have units from different measurement systems. It is not necessary that numeric values in the query and SKU to match exactly. We compute the difference between corresponding numeric values of the query and SKU and apply a boost that is inversely proportion to the difference. For example, a query: “`45 inch tv`” will match SKUs for *43 inch TVs* (higher boost) as well as *49 inch TVs* (lower boost)

2.3 Brand

A *Brand* is defined as a phrase that provides more information about the manufacturer of the item. *Samsung*, *Calvin Klein*, etc. are examples of brands. Brands are important because they capture information about the preferences of the customer but are not essential in defining the recall. The search system tries to honor the customer’s preference regarding the brand by boosting SKUs that match the brand specified in the query. This scheme ensures that the search result includes SKUs from other brands albeit at a lower position compared to SKUs that match the brand in the query.

We observed that in some cases customers tend to use the brand name as a synonym for a product, for example, “`q-tips`” to denote cotton swabs and “`kleenex`” to denote tissues. This type of behavior is common for a subset of brands that have high brand equity and are taken to represent the product itself. We wanted to respect the customer’s preferences while still providing them with a wide range of similar products from other brands and so we introduced the *default_product* relation, which maps these finite subsets of brands with their default *Product* nodes. This relation then allows the NERC system to map the query “`kleenex`” to **`kleenex := Brand, tissues := Product`** and have the flexibility to present relevant SKUs from other brands at a lower position in the recall. Currently, we do not support a parent-child relationship between brands (for example: *Nike*) and sub-brands (for example: *Nike Air*). and treat each sub-brand as a variation of the original brand.

2.4 Slots

We propose five slots or properties - *synonyms*, *attributes*, *primary_attributes*, *brand* and *default_product* and show how they can be used to recall relevant SKUs for a given query. The *synonyms* slot indicates synonyms of a given class and are typically used to address alternate phrases used to describe the same item. The *synonyms* slot exists for all classes in our ontology.

The *attributes* slot has the *Product* class as its domain and the *Attributes* class as the range. It helps in specifying all relevant attributes for a given SKU. Since we insist on *atomic* products, this slot helps us in distinguishing relevant SKUs from irrelevant SKUs in the recall. Consider the two queries “`Dining Chair`” and “`Outdoor Chair`”, which refer to two very *different* products even though they are both *chairs*. The NERC system is able to extract the attributes *dining* and *outdoor* for those two queries and is able to boost SKUs that match these attributes to the top of the recall. Thus, the customer is presented with relevant SKUs in each case even though the product type of both queries is the same.

Consider a search query “`cotton shirt`”, where the NERC system is able to extract the material *cotton*. As discussed previously, the system will retrieve all shirts and automatically boost cotton shirts so that they appear the top of the recall. Let us assume that there are two SKUs - one shirt made of 100% cotton and the other shirt made out of 95% polyester and only 5% cotton. If there is no notion of *primary_attributes* both SKUs will receive the same attribute boost and will be considered equally relevant. The *primary_attributes* is a special slot that maps a *Product* with a single *Material* or *Color* subclass. In case of the previous example the *primary_attribute* will point to **`cotton := Material`** for the first SKU and **`polyester := Material`** for the second. This slot helps increase relevancy by boosting only SKUs that match the corresponding primary color or material.

The *Brands* slot has the *Product* class as the domain and the *Brands* class as the range. It defines the manufacturer for a given SKU. As mentioned previously, the *default_product* slot helps in assigning a product to a small set of brands like *Kleenex* that are used synonymously with products. Both slots help increase relevancy by boosting all SKUs that match the extracted brand from the query but without sacrificing the ability to show SKUs from other brands at lower positions on the search page.

Our current implementation of ranking SKUs is rather simple - providing fixed boosts when products, brand and attributes from the query match products, brands and attributes in the SKU. In future, we will use these matches in conjunction with a ranking model to further improve relevancy.

3 ONTOLOGY LEARNING

The task of building an ontology is a time consuming and expensive task and usually involves a domain expert. Techniques that support ontology engineering and reduce the cost of building and maintaining ontology are required to ensure that this task is scalable. Ontology learning can be thought of as data driven methods that support building ontologies by deriving classes and meaningful relations between them. Petucci et al [22] formulate the problem of ontology learning from natural language as a transductive reasoning task that learns to convert natural language to a logic based specification. It breaks down the problem into two tasks - sentence transduction phase and sentence tagging phase. It uses RNN for sentence tagging and RNN Encoder-Decoder model for sentence transduction.

Figure 2 shows the concept of *ontology learning layer cake*, which was introduced by Cimiano et. al [3] and further discussed in [17]. The layers focus on ontology learning and show dependencies between various tasks in the ontology learning system. The layers are designed such that results of lower layers serve as inputs to higher layers.

The *term extraction layer* is the lowest layer in the cake. It aims to learn the relevant terminology of the domain. A naive approach is to just use term frequencies assuming that relevant concepts are also most frequent. However, other sophisticated methods like TF-IDF [28] or C-value/NC-value measure proposed in [7] can also be used.

The next layer is the *synonym extraction layer*, which deals with extracting synonyms for the terms identified in the previous layer. Synonyms can be extracted using a distributional representation of words, which claim that similar words share similar contexts [27]. Semantic relatedness using wordnet or Wikipedia categories can be used as well [8].

The third layer is the *concept formation layer*, which provides a definition of concepts, their extension and the lexical signs which are used to refer to them. The fourth layer is the *concept hierarchy layer*, which deals with inducing, extending and refining the ontology hierarchy. This task can be accomplished by methods like matching lexico-syntactic patterns as demonstrated by Hearst in [12], clustering different objects based on their feature vectors and using phrase analysis i.e., making use of internal structure of noun phrases to discover taxonomic relations [25].

The fifth and sixth layers deal with *Relations*, which is the task of learning relation labels (or identifiers) as well as their corresponding domain and range. Some common methods include finding co-occurrence between words as proposed by Madche [16].

The last two layers are *Axiom Schemata* and *General Axioms*, which are related to rules and axioms. These two layers deal with transformation of natural language definitions into OWL Description Logic axioms, and building a domain specific ontology by pruning an existing general ontology using the given corpus [1].

Since we do not deal with axioms, the last two layers of the ontology learning cake are not relevant to our task. The first three layers require most manual effort and are most time consuming for our task. This paper describes three methods that can automatically derive terms and *Product* concepts, which correspond to the first and the third layer in the ontology learning layer cake. Ontology creation cannot be fully automated and our methods produce candidates for manual review, greatly decreasing the time required for ontology development. These methods do not address the problem of synonym resolution but other methods that use click logs on top of an existing ontology can help with the second layer. Unfortunately, describing this method is beyond the scope of this paper.

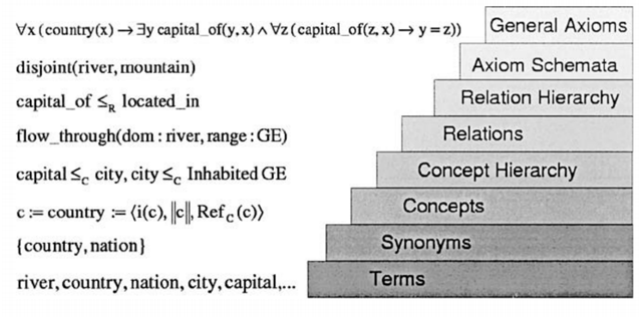

$\forall x\,(\text{country}(x) \rightarrow \exists y\ \text{capital\_of}(y,x) \wedge \forall z\,(\text{capital\_of}(z,x) \rightarrow y=z))$
General Axioms
disjoint(river, mountain)
Axiom Schemata
$\text{capital\_of} \leq_R \text{located\_in}$
Relation Hierarchy
flow_through(dom : river, range : GE)
Relations
$\text{capital} \leq_C \text{city}, \text{city} \leq_C \text{Inhabited GE}$
Concept Hierarchy
$c := \text{country} := \langle i(c), \|c\|, \text{Ref}_C(c)\rangle$
Concepts
{country, nation}
Synonyms
river, country, nation, city, capital,...
Terms

Figure 2: Ontology Learning Layer Cake.

4 AUTOMATICALLY EXTRACTING PRODUCT ENTITIES

We describe three methods (Token Graph Method, Augmented Graph Method and LSTM-CRF method) that can be used to automatically extract atomic Product entities from a customer's search query. Two of these methods may also be extended to extract relevant attributes and brands from the search query as well as from product titles. We then compare the performance of these three methods relative to each other.

We assume that there exists a bipartite graph $G : q \mapsto S$ that maps a customer's search query q to a set of clicked SKUs S. This graph may be further augmented by including SKUs that were added to cart or bought. Search queries and SKUs are represented by nodes in the graph and an edge between a query and a SKU indicates that a customer searched for the query and clicked on the corresponding SKUs. The weight of the edge indicates the strength of the relationship between the query and the SKU and is modeled using number of clicks between the query and the SKU aggregated over a certain length of time. There are no edges between queries or between SKUs. Very broad queries like "`cheap`" or "`clothing`" either do not contain any products or contain very generic product terms and add noise to the data. We use entropy of a query across different categories to determine if it is broad and remove it from the graph. We also remove queries that are just brands from the graph and query-SKU pairs that have edge weights less than some threshold (T). Finally, we apply a stemmer to perform stemming for terms in the query. Let G' denote this cleaned bipartite graph. The task can be formulated as follows: Given a cleaned bipartite click graph G', compute a sorted list of *Product* sub-classes that are *atomic* and *relevant* for that category. We present three methods to create the sorted list of *Product* classes and compare them.

4.1 Token Graph Method

This method is a very simple unsupervised method for extracting relevant products from a customer's search query and can be applied to any category without any previous data. Let $C = \{q_0, q_1, \ldots, q_n, s_0, s_1, \ldots s_m\}$ be a connected component in the bipartite graph G' mentioned previously. Let $Q = \{q_0, q_1, \ldots, q_n\}$ be a set of queries in this connected component and we can assume that all of them are related to each other because they share some of the same clicked SKUs. Let us assume that we can detect prepositions in the query and have removed them and all words after it from the query. Each token in the query is either a brand, product, attribute or other (part number, stopword, etc.) and we can create a new graph G_{token} where each token is a node and there are edges between adjacent tokens. Figure 3 shows the token graph G_{token} for the query set *{women dress, white dress, DKNY sleeveless dress white}*. Most often, the product token is the last term in the query before any prepositions and thus it is the node that maximizes the ratio $\frac{N_i}{N_o+N_i}$, where N_o is the number of outgoing edges and N_i is the number of incoming edges for the node corresponding to the token. If the search query contains just a single token, we set $n_i = n_o = 1$. We can further improve precision by requiring that $N_i \geq T$, where T is some threshold. There are obvious exceptions to the rule, for example the search query: "`DKNY sleeveless dress white`" where the product *dress* does not appear in the end of the query. However, we assume that such cases are rare and assume that aggregating this process over all related queries takes care of the occasional exception. We can generate a potential product from each connected component and aggregating over all connected components gives us a potential list of products.

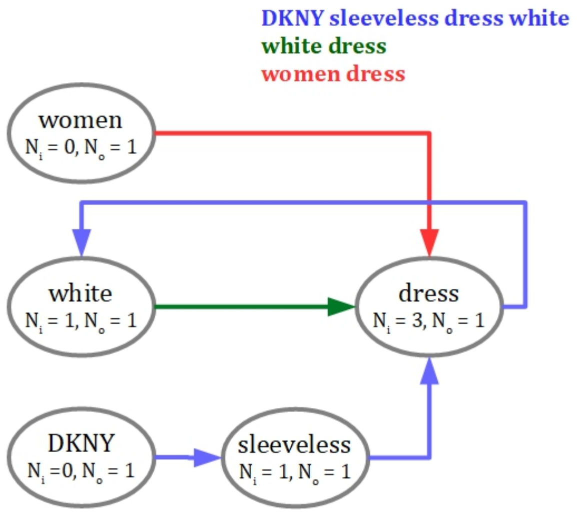


Figure 3: Token Graph Method.

4.2 Augmented Graph Method

The graph method in the previous section works pretty well but makes a very strong assumption that the product always appears towards the end of the search query. It is also very aggressive in removing the preposition and all tokens after it. For example, it will convert the query "`seven for all mankind skinny jeans`" to "`seven`", which is obviously wrong. Finally, it is oblivious to the parts-of-speech of the terms.

Typically, product words are nouns (television, shirt, etc.) and we can take advantage of parts-of-speech tags to improve the accuracy of the system. One option is to use global parts-of-speech tags from wordnet [19] or some other similar repository. However, a word like `pack` may be used as a noun (`battery pack`) or a verb (`pack your stuff`) depending upon the category. Another problem with using a service like wordnet is that it may not contain some brand words like `Samsung`. A better approach is to use a service like Google's SyntaxNet [26] to generate parts-of-speech tags on the fly and this helps us retain local information as well as get parts-of-speech tags for brands like `Samsung`. We realized that most queries are not grammatically correct and so the generated parts-of-speech tags may not be very accurate. To get around this problem, we ran SyntaxNet on the descriptions of all SKUs in G' to generate a mapping between terms and their parts-of-speech tags. Let $v_i^P = [NOUN, VERB, ADVERB, ADJ, PREP, NUM, \ldots]$ denote a vector $\in \mathcal{R}^7$ that represents the parts-of-speech for some term t_i. Here, $NOUN$ indicates the fraction of the time the part of speech tag for that term was a noun, $VERB$ indicates the fraction of the time the part of speech tag for that term was a verb and so on. We can use this map to generate parts-of-speech vectors for each term in the search query. We found that it was better to aggregate the parts of speech tags for terms across the entire category because the quality of descriptions greatly varies across SKUs. Thus, the parts of speech vector for each term is constant across the entire category.

We want to capture the local graph information discussed in the previous section. This can be done by creating the local graph and computing the number of incoming and outgoing edges for each term in the query. Let $v_i^G = [n_i, n_o, \frac{n_i}{n_i+n_o}]$ denote a vector that captures local graph information for the i^{th} term. Here, n_i indicates the number of incoming edges for the node denoting the term in the local graph and n_o indicates the number of outgoing edges for the same node.

Let $v_i^N = N - i$ denote a scalar describing the position of the i^{th} term in the search query, where N is the number of terms in that query. This vector helps the model prefer later words in the query as products.

Finally, let $v_i = (v_i^P, v_i^G, v_i^N)$ denote a concatenated vector that captures all relevant information for the i^{th} term in the query and let $V = (v_0, v_1, \ldots, v_n)$ denote the vector for the entire search term. We will use this vector as an input to the model to predict the product terms from the search query. We use a convolution neural network (CNN) that consists of three convolution layers with filter sizes of $n_1 = 7$ for the first layer, $n_2 = 5$ for the second layer and $n_3 = 3$ for the third layer. The number of filters are set to 256 in each case. There is no max-pooling layer because we want to keep the filter information for each stride. The output of the last filter is then passed to fully-connected layers with a time-distributed-dense layer as the very last layer for making tag predictions.

The intuition behind this model is that the convolution layers are able to capture local information using the parts-of-speech tags of surrounding terms and the number of incoming and outgoing edges for the terms in the vicinity. It is then able to make a decision by combining all three vectors to predict if a term in the query is a product or not. The model is trained using queries across six categories (Electronics, Women's clothing, Men's clothing, Kid's clothing, Furniture, and Home) and the tested using queries from the Baby category. Each query can give zero or more product candidates and we aggregate candidates from all queries to come up with a list of potential products.

4.3 NER Model using Bidirectional LSTM-CRF

This model is very different from the two described earlier. It does not look at the local term graph but makes a decision using a word2vec [18] vector for each term in the query. The word2vec vectors are of dimension $D = 300$ and are generated using data from Wikipedia and from SKU titles from the Jet.com catalog. The training data consists of queries where each term has been tagged in IOB format with either a O (other), B-PRODUCT (beginning of product) or I-PRODUCT (intermediate of product). For example, the query phrase `metal bar stool for kitchen` would be tagged as: *metal O bar B-PRODUCT stool I-PPRODUCT for O kitchen O*. We use bi-directional LSTM-CRF model described in [14] to train the NER model. The training data was tagged automatically using existing the existing query and SKU understanding service along with user engagement data to filter out potentially bad results.

$$\begin{aligned} h_t &= o_t \odot \tanh(c_t) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\ c_t &= (1 - i_t) \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \end{aligned} \tag{1}$$

Let $S = (x_1, x_2, ..., x_n)$ represent a sentence containing n words where x_t represents the word at position t and each word is represented by a d-dimensional vector. We compute the left-context $\overrightarrow{h_t}$ using a forward LSTM and also a right-context $\overleftarrow{h_t}$ using a backward LSTM, which reads the same sequence in reverse order. The contexts $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ are computed as shown in equation 1, where σ is the element-wise sigmoid function, $\odot$ is the element-wise product, W is the weight matrix and b is the bias. The left and right contexts are then concatenated to represent a word representation $h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$, which is used by the conditional random field (CRF) for NER tagging.

Lexical features of queries can be quite different across categories. So for this method to generalize well, it was important to select the training dataset such that the labeled queries belonged to different categories. We chose queries from six categories (Electronics, Women's clothing, Men's clothing, Kid's clothing, Furniture, and Home) for training data and extracted candidate products using queries from the Baby category.

4.4 Model comparison

The token graph method described in section 4.1 is an unsupervised model and so does not require any training data. The other two models are trained using labeled queries from six categories and all three models are tested using the same test set, which are queries from the Baby category. We exclude all broad queries and all queries that are just brands to keep it consistent with the training data. We believe that this is a fair test as it allows us evaluate the model's performance on a previously unseen category - a task that is essential for automatically creating ontologies.

Each model produces potential product candidates from queries and these candidates are sorted in decreasing order of frequency. We evaluate the top 500 candidates from each model and manually verify if each potential product was actually a product or not. We consider a term to be a product only if it is *atomic* and sellable on the site. For example, *diaper* is a product but *baby* (we don't sell babies) and *diaper cover* (not atomic) are not. Table 1 shows the top ten candidates (from the top 500 candidates) from each model along with our manually annotated results denoting if the given entry is a product (P) or not (N).

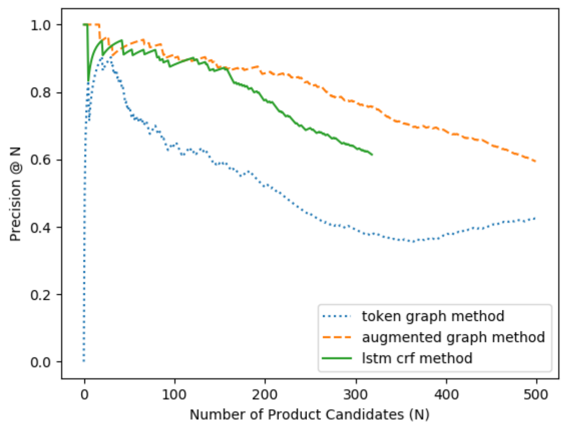


Figure 4: Precision @ N graphs for the three models (top 500 candidates).

Figure 4 shows a *precision @ n* graph for all the three models over their top 500 candidates. The LSTM-CRF model produced just over 300 candidates and so its graph is truncated. Both the augmented graph method and the LSTM-CRF method have a higher precision initially and are able to correctly identify products from the query logs. Figure 5 shows a zoomed in view of the first 100 candidates and it can be observed that the augmented graph model is able to predict products more accurately than the LSTM-CRF method. As expected the naive graph method performs the worst in terms of accuracy but can return more products than the LSTM-CRF method.

The naive graph method may seem like the worst method but it has one very significant advantage over the other two methods - it is completely unsupervised. This allows it to be used when there is no training data from other categories. We recommend that this method should be used initially and it can pave the way for the other two supervised methods for other categories.

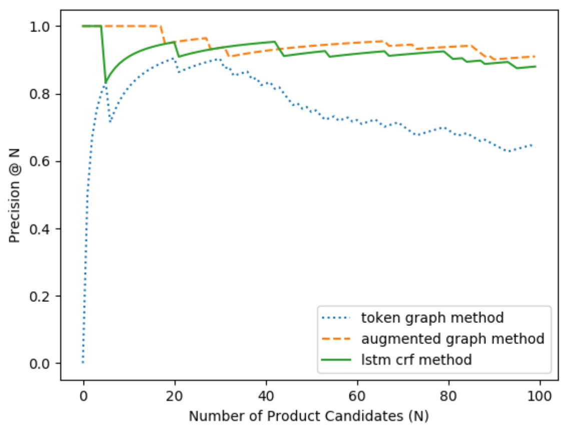


Figure 5: Precision @ N graphs for the three models (top 100 candidates).

Table 1: Top 10 Potential Products

Num	Graph Method	Augmented Graph Model	NER Model
1	all (N)	diaper (P)	diaper (P)
2	sippycup (P)	wipe (P)	wipe (P)
3	cup (P)	formula (P)	bottle (P)
4	bib (P)	carseat (P)	bag (P)
5	playard (P)	bottle (P)	cover (P)
6	insert (P)	stroller (P)	ups (N)
7	ct (N)	bag (P)	pants (P)
8	highchair (P)	gate (P)	seat (P)
9	case (P)	cereal (P)	pad (P)
10	stroller (P)	highchair (P)	bib (P)

P denotes a *product* and N *not a product*

5 CONCLUSION

In this work we proposed a search-side ontology that can be used for Named Entity Recognition and Classification of Queries. We show that this ontology is better suited for search as compared to supply-side or catalog-side ontologies. We propose three methods to generate *Product* classes for this ontology. We also compare the three methods and show that the Augmented Graph Method which uses local token information along with parts-of-speech tags performs better than the naive Graph Method and the bidirectional LSTM-CRF method in generating *Product* classes.

ACKNOWLEDGMENTS

The authors would like to thank Brent Scardapane and Dennis Jordan for their help with the ontology creation process.

REFERENCES

[1] Paul Buitelaar and Bogdan Sacaleanu. 2001. Ranking and selecting synsets by domain relevance. In *Proceedings of WordNet and Other Lexical Resources: Applications, Extensions and Customizations, NAACL 2001 Workshop*. Citeseer, 119–124.
[2] Bruno Charron, Yu Hirate, David Purcell, and Martin Rezk. 2016. Extracting semantic information for e-commerce. In *International Semantic Web Conference*. Springer, 273–290.
[3] Philipp Cimiano. 2006. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer-Verlag, Berlin, Heidelberg.
[4] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*.
[5] Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. 2013. Getting more out of biomedical documents with GATE's full lifecycle open source text analytics. *PLoS computational biology* 9, 2 (2013), e1002854.
[6] Ying Ding, Dieter Fensel, Michel Klein, Borys Omelayenko, and Ellen Schulten. 2004. The role of ontologies in ecommerce. In *Handbook on ontologies*. Springer, 593–615.
[7] Katerina T Frantzi and Sophia Ananiadou. 1999. The C-value/NC-value domain-independent method for multi-word term extraction. *Journal of Natural Language Processing* 6, 3 (1999), 145–179.
[8] Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis.. In *IJcAI*, Vol. 7. 1606–1611.
[9] Rafael Glater, Rodrygo LT Santos, and Nivio Ziviani. 2017. Intent-Aware Semantic Query Annotation. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 485–494.
[10] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. " O'Reilly Media, Inc.".
[11] Trey Grainger, Timothy Potter, and Yonik Seeley. 2014. *Solr in action*. Manning Cherry Hill.
[12] Marti A Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, 539–545.
[13] Jian Hu, Gang Wang, Fred Lochovsky, Jian-tao Sun, and Zheng Chen. 2009. Understanding user's query intent with wikipedia. In *Proceedings of the 18th international conference on World wide web*. ACM, 471–480.
[14] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. *CoRR* abs/1603.01360 (2016). arXiv:1603.01360 http://arxiv.org/abs/1603.01360
[15] Taehee Lee, Ig-hoon Lee, Suekyung Lee, Sang-goo Lee, Dongkyu Kim, Jonghoon Chun, Hyunja Lee, and Junho Shim. 2006. Building an operational product ontology system. *Electronic Commerce Research and Applications* 5, 1 (2006), 16–28.
[16] Alexander Maedche and Steffen Staab. 2000. Discovering conceptual relations from text. In *Ecai*, Vol. 321. 27.
[17] Alexander Maedche and Steffen Staab. 2004. Ontology learning. In *Handbook on ontologies*. Springer, 173–190.
[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.
[19] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.
[20] David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30, 1 (2007), 3–26.
[21] Natalya F Noy, Deborah L McGuinness, et al. 2001. Ontology development 101: A guide to creating your first ontology.
[22] Giulio Petrucci, Chiara Ghidini, and Marco Rospocher. 2016. Ontology learning in the deep. In *European Knowledge Acquisition Workshop*. Springer, 480–495.
[23] Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov. 2004. KIM–a semantic platform for information extraction and retrieval. *Natural language engineering* 10, 3-4 (2004), 375–392.
[24] Daniel E Rose and Danny Levinson. 2004. Understanding user goals in web search. In *Proceedings of the 13th international conference on World Wide Web*. ACM, 13–19.
[25] David Sánchez and Antonio Moreno. 2005. Web-scale taxonomy learning. In *Proceedings of Workshop on Extending and Learning Lexical Ontologies using Machine Learning (ICML 2005)*. 53–60.
[26] Announcing SyntaxNet. 2016. The Worlds Most Accurate Parser Goes Open Source.
[27] Gerhard Wohlgenannt and Filip Minic. 2016. Using word2vec to Build a Simple Ontology Learning System.. In *International Semantic Web Conference (Posters & Demos)*.
[28] Ziqi Zhang, José Iria, Christopher Brewster, and Fabio Ciravegna. 2008. A comparative evaluation of term recognition algorithms. (2008).