2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Locally Constructing Product Taxonomies from Scratch Using Representation Learning

Mayank Kejriwal, Ravi Kiran Selvam
Information Sciences Institute
University of Southern California
Marina del Rey, CA
{kejriwal,rselvam}@isi.edu

Chien-Chun Ni, Nicolas Torzec
Verizon Media
{chien-chun.ni,torzecn}@verizonmedia.com

***Abstract*—Given a domain-specific set of concepts, local taxonomy construction (LTC) is the problem of 'locally' inducing the neighborhood of a concept (from the set of target concepts) without being given any example links. The problem, despite having practical importance, has received little research attention due to its difficulty (in contrast with link prediction, a problem that resembles it and has undergone broad study). In this paper, we present a formalism and deep empirical study on the LTC problem. In particular, we show that an innovative application of representation learning approaches from the natural language community could be adapted to tackle the problem, often quite effectively. We also present a detailed information retrieval (IR)-based methodology for evaluating these solutions on three real-world product datasets of varying sizes. To the best of our knowledge, this is the first paper to introduce the LTC problem, especially for e-commerce applications, and offer effective, nearly unsupervised, solutions, for addressing it on real-world data.**

***Index Terms*—Taxonomy Induction, Local Taxonomy Construction, Concept Ranking, Information Retrieval, Representation Learning, E-Commerce**

I. Introduction

Frequently, in many domains, website designers and builders of recommendation systems start from a set of semantic categories or *concepts* that needs to be compiled into a proper taxonomy. For example, as shown in Figure 1, a clothing retailer may start with a catalog of product 'concepts' (such as *Overalls* and *Dresses*), but needs to impose a structure such as on the right to better organize and understand her domain. In its most general form, this problem is known as *taxonomy induction* [1], [2]. For example, in the e-commerce domain, price shopping and comparison websites pull in product categories ('concepts') from multiple websites by the thousands. Some kind of relational ordering between these concepts is necessary, both for developing a deeper understanding about the domain, but also for building practical products such as websites and catalogs that make for an intuitive and satisfying user experience. Such a taxonomy could even serve as a simpler version of, or even the backbone to, a final ontology that is more 'graph-like' and contains other ontological components such as constraints. A knowledge engineer would not have to begin from scratch in constructing the ontology, but could instead start from the taxonomy as a baseline domain model. Manually building such taxonomies is difficult since, in real-world problem settings, there could be thousands of concepts to organize. The total number of possible taxonomies is exponential in this number.

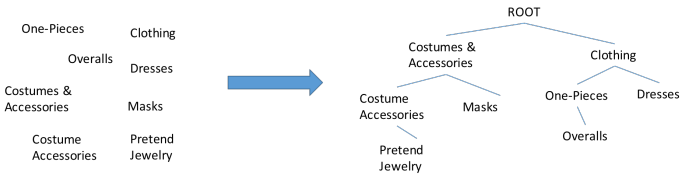

Fig. 1: An illustration of the taxonomy induction problem, using real-data from the Google Product Taxonomy (Section V).

Unlike the traditional link prediction problem in social networks and knowledge graphs [3], the problem of *inducing* such a taxonomy given only a set C of concepts (hereby referred to as the *concept-set*) is a difficult problem because it falls under a class of machine learning problems that have to work without any examples. The best known examples of these problems are clustering-based applications such as community detection. However, taxonomy induction is different, since we have to discover a set of highly localized links for each concept. Another way to understand the difference between clustering and taxonomy induction is that, for the former, the number of clusters is often a small constant number (almost never more than 100, and far fewer than the data points, which can sometimes number in the tens of thousands or even millions) while the number of links that have to be inferred in a taxonomy induction setting is a multiple of the number of concepts.

In this paper, we address a simpler, but still important, version of the global taxonomy induction problem called *local taxonomy construction* (LTC). We state this problem as follows

IEEE/ACM ASONAM 2020, December 7-10, 2020
978-1-7281-1056-1/20/$31.00 © 2020 IEEE

(with more details provided in Section III): given a set C of *target* concepts, and other generic 'background' resources available on the Web (such as a domain-specific text corpus or an example taxonomy from a different website or data source), *independently* infer the neighbors of every concept $c \in C$. The independence assumption is important as it gives the problem its local flavor. An important assumption, not stated above, is that the concepts are represented using their 'labels', which is one of the few sources of information that we can use (in conjunction with generic background resources) to guide us in recovering local structure. As we argue in Section III, framing the problem in this way allows us to make it tractable both in terms of solving it and evaluating it. It is also a 'stepping stone' to solving the global version of the problem.

Despite the simple formalism of the problem, solutions for it are complex, and would clearly depend on the background resources available (or that can be feasibly acquired online). A naive approach, for example, might forge a link (usually, but not necessarily, understood as having 'super-type' semantics[1]) between two concepts if one is a sub-word of the other. Compared to a ground-truth taxonomy, this approach only (unsurprisingly) does slightly better than random, since so many concepts do not share (or misleadingly share) sufficient sub-string similarity, making it difficult to capture semantic relations between them.

We hypothesize that a more promising way (barring availability of user experiments or human-generated training data) to approach the problem is via representation learning, popularly known as 'embeddings' [4], [5]. These approaches have been used to exceed state-of-the-art performance in multiple applications, including information extraction, entity resolution and sentiment analysis [6], [7], [8]. However, no data exists on how they would perform if applied to this problem, where no examples are available.

This paper presents a principled and detailed empirical study of how word embedding models could be used in a variety of potential taxonomy induction scenarios (with availability of different classes of background resources). Our goal is not to present a radically new algorithm but to explore how established representation learning algorithms, including so-called *retrofitting* algorithms (which attempt to adapt a pre-trained embedding to a specific domain using an existing semantic lexicon such as WordNet) could be applied to this problem [9], [10]. We also detail how different approaches could be evaluated using a novel information retrieval-based methodology. In our study, we use real-world e-commerce taxonomies and embedding resources that are widely used in the community. Based on 20GB+ worth of experimental data collected and analyzed, we show that representation learning offers a promising first solution (and could even be used feasibly in practice on new concept-sets) to the local taxonomy construction problem on practical e-commerce tasks.

[1]Also referred to as hypernymy in natural language versions of this problem.

II. Related Work

The research presented in this paper is related to several existing lines of research that we briefly cover below.

Representation Learning and NLP. With the advent of neural networks and deep learning in the last decade, representation learning (also known as 'embeddings') have become very prominent in the Natural Language Processing (NLP) community [4], [11], [5], [12]. More recently, transformer-based models such as BERT have become popular [11], though much more expensive to train on a corpus (and requiring far more data) than previous algorithms such as bag-of-tricks [12] and Glove [5]. More intriguingly, the ability of the former word embedding algorithms to do analogical reasoning without any apparent supervision (e.g., the famous 'King is to Queen as Man is to Woman' example), and the ability of recent algorithms to achieve near-human performance on commonsense question answering raises the question of whether such models are capable of successfully capturing the complex semantics in natural language.

Embeddings for Structured Data. The success of word embeddings, and similar representation learning models (e.g., based on autoencoders and decoders [13]) in the computer vision community has led to similarly successful proposals being floated both in the knowledge discovery and semantic web communities. In the former, network embedding algorithms such as DeepWalk have become popular and are based loosely on word2vec [14]. In the Semantic Web, RDF2Vec has become a popular choice and shown to be useful in several Semantic Web problem areas [15]. More generally, 'knowledge graph embeddings' such as TransE and HolE have become popular choices for embedding multi-relational graphs such as DBpedia and Freebase [3]. However, such algorithms invariably assume that (i) the node-set (what we call the concept-set in this paper) of the graph is typically known in advance; (ii) there exist a set of 'positive' links between many of these nodes, which can be used to train a model and 'complete' the graph by inferring more links. The problem in this paper is very different: we only consider one relation[2], rather than multiple relations, and the problem is a 'from-scratch' problem (no example edges are available during test time for an algorithm to exploit).

Retrofitting Pre-Trained Embeddings. While more recent embedding algorithms such as BERT rely on the notion of 'fine-tuning' to derive a 'refined' set of task-specific embeddings from a set of pre-trained embeddings[3] [11], the problem of refinement and computational costs of re-training were realized earlier in the decade in the NLP community. Some authors showed, for example, that performance on various downstream tasks tended to improve when pre-trained embeddings were 'retrofitted' to an existing *semantic lexicon* such as

[2]One could theoretically consider the induction of a multi-relational taxonomy from scratch; we leave it as a promising avenue for future research.

[3]Trained on a large, generic corpus of text, which usually includes, at the very minimum, Wikipedia and/or a news corpus) that took many hundreds of hours of computational time to train even in the industrial labs where these algorithms were developed.

WordNet [9], [10]. In this paper, we show that retrofitting can be a valuable baseline for learning domain-specific models for local taxonomy construction. To the best of our knowledge, such methods have mostly been applied to improving the embeddings for an NLP problem, and not on a taxonomy construction problem as is this paper's subject.

Taxonomy and Ontology Induction. Taxonomy (and more generally, ontology) induction has been studied for some time now, but the underlying data on which the taxonomy is induced tends to be textual or context-rich (and in most cases, involves a body of generic nouns, such as WordNet extensions [1]). Examples include the work by [1], [16], [2] and [17]. Most importantly, these works rely significantly on help from extra resources *during test time* for semantic understanding, which is understandable since many of them preceded the advances in representation learning that occurred in the last decade. The work by [1] starts from the WordNet 2.1 base, which means some links are already available. In [16], the authors present *OntoLearn Reloaded*, which relies on a specific set of text documents and Web documents for context, and is not constrained by a set of specific concepts over which a taxonomy must be induced. In contrast, the problem considered in this paper starts from the concepts, which the domain expert has decided must be included in the taxonomy. A text corpus is not guaranteed, though we do consider the possibility in one set of experiments in this paper. We show, in fact, that the best approaches do not need a domain-specific corpus for training, if a sufficiently robust embedding has been trained on a generic corpus. More recently, [2] and [17] used hypernym subsequences and reinforcement learning respectively. The former considers a problem definition that is most related to this work, but the domains used in the evaluation are still very closely related to generic domains (e.g., food) rather than complex, specific domains such as e-commerce that we consider in the evaluations in this work. For near-generic domains such as food, a resource like WordNet already contains many links, which makes the problem less challenging than a typical e-commerce application .

More broadly, there is no current work that has proposed to use representation learning for addressing this problem. For other problem domains such as information extraction and entity resolution [6], [7], similar work has shown that reasonable representation learning models can often outperform the state-of-the-art, sometimes on problems (e.g., relation extraction) that had proven too difficult to tackle before, especially in minimally supervised settings (such as involving weak or distant supervision) [6]. We attempt to prove the same for local taxonomy construction through a carefully designed empirical study.

III. Local Taxonomy Construction

The *general* problem of taxonomy induction may be stated as one of inducing a taxonomy T given a concept-set C, potentially given some background or 'training' resource (or a set of resources) B. T, in this context, may be thought of as a 'tree' over C: $T = \{(c_i, c_j) | c_i, c_j \in C \times C, c_i \neq c_j\}$. Taxonomies could be defined in more general ways, but in practice, product taxonomies often are modeled along the lines of a tree as they are meant to guide product categorization and website navigation[4]. Formally, to ensure that a concept can have at most one parent, we assume the constraint that if $(c_i, c_j) \in T$ and $(c_k, c_j) \in T$, then it must necessarily be the case that $c_i = c_k$. We call c_i the *parent* (equivalently, super-type) of c_j. This is an abstract relation with reasonably well-defined semantics (though usually of a domain-specific nature): in the natural language community, it tends to go by the name *hypernymy* while in the Semantic Web, the RDFS rdfs:subClassOf predicate is the best fit. Note that, since there may be 'upper-level' nodes $\{c_1, \ldots, c_m\}$ that do not have parents, we assume (without loss of generality), that there is a single artificial *root* node c_R that is introduced after taxonomy induction and that serves as the parent of every concept in $\{c_1, \ldots, c_m\}$.

Inducing a taxonomy given just C is a difficult problem that has not been addressed much in the literature, though the previous section covered some relevant work. In practice, inducing the full taxonomy is not necessary. Instead, we tackle the more manageable problem of determining the neighbors of a concept without being given any training data (existing neighbors). While this problem is more difficult, it is somewhat immune from the problem of *error cascading* that would ensue if we were trying to recover a single global taxonomy by combining local 'fragments' in some principled way.

With the terminology above, *local taxonomy construction* (LTC) can be stated in similar terms as full taxonomy induction (henceforth called taxonomy induction). Given a concept $c_i \in C$, an LTC approach would aim to determine the neighbors (including the super-type) of c_i in the underlying *unknown* taxonomy T (though in evaluations we only use known taxonomies as ground-truths). Similar to taxonomy induction, LTC can also draw upon a background resource B.

The formulation above raises two important questions. First, what is the nature of B? We present several choices in the next section, based on representation learning, including a domain-specific text corpus, a pre-trained embedding model and a taxonomy over the same domain but on a different concept-set. Second, regardless of an actual approach used, how do we *evaluate* T? It is reasonable to assume that, on average, a 'good' LTC system would aim to retrieve true neighbors (in the underlying taxonomy) while minimizing false positives and negatives. In practice, to make a solution more robust, we would frame it in terms of *Information Retrieval (IR)*, used by all major search engines to evaluate the efficacy of their ranking algorithms [18]. We describe the IR-based evaluation protocol and its rationale in more detail in Section V-C.

Before presenting some viable approaches for LTC, we present two reasons why it is expected to be a more challenging problem than (for example) more traditional 'edge-

[4]However, in principle, future work could consider generalizing the problem to one of 'graph induction.'

discovery' problems such as link prediction, even when given as background resource, a 'training' taxonomy T'. First, while the training taxonomy is valuable and it is also domain-specific (e.g., also from the product domain, perhaps scraped from a different website), it usually has little or no overlap between C and its own concept-set, denoted as $C_{T'}$ (equivalently, $C_{T'}$ is just the set of nodes in the graph-theoretic definition of T'). Otherwise, a simple solution would have been to just induce a sub-graph on T' (over the overlapping nodes between $C_{T'}$ and the 'target' concept-set C over which we are trying to do local taxonomy construction) and then apply a link prediction algorithm to deduce the remaining edges. We do not discount the possibility of some overlap, but in empirical practice, the chance of overlap is minor and it is rarer still for an *edge* in the ground-truth taxonomy to overlap with an edge in a background taxonomy[5].

The second challenge arises because concepts are assumed to be represented by their *labels*, which may be single words, but could also be multi-word phrases. A good method should be able to generalize to all such cases. Another important aspect to note is that, while C and $C_{T'}$ are not necessarily large, they are still too large[6] to manually parse and build a taxonomy or 'ontology' (using completely manual techniques).

Another challenge, specific to using embedding-based solutions for LTC, is that for a structured (i.e. link prediction) problem such as this, the typical approach is not to use 'natural language' embeddings such as word2vec or BERT [4], [11], but knowledge graph (KG) embeddings like RDF2Vec and TransE [15], [3]. However, such embeddings assume the existence of a multi-relational KG where each entity has plenty of context (such as many relations and paths to other entities) to draw on, when inferring other 'missing' edges. The goal of these algorithms is to do prediction or completion on this semi-complete and partially noisy KG. This is also true for 'network embeddings' such as node2vec, DeepWalk and LINE [14], [19], [20]. A disadvantage of network embeddings that they only rely on structure and not labels. In our case, we begin with no 'structure' (pre-existing edges or links) and we have to only rely on labels. At the same time, there is structure in the training taxonomies that do not have label overlap, but contain domain-specific information that could be useful.

Given these challenges and observations, in the next section, we argue that representation learning approaches from the natural language community can be a good way to address the challenges above, if properly used and adapted to the problem of LTC. We present some reasonable approaches by adapting publicly available resources in an innovative way. We implement and investigate the approaches empirically in Section V.

[5]For example, it may be that even if both taxonomies contain the concepts 'baby clothes and toys' and 'diapers', one taxonomy has an edge between them, but the other does not, meaning that the intersection would actually lead to noise when doing local taxonomy construction on the second taxonomy. The reason why this might happen is because there might be a better super-type for 'diapers' (e.g., 'baby essentials').

[6]A general rule of thumb is several hundreds or even thousands of concepts, but not tens of thousands, of concepts.

IV. Representation Learning Approaches for LTC

One of the questions raised in the previous section was the nature of the background resource, B. In this study, B is related to, or used by, a representation learning algorithm R. We assume that R is a word embedding algorithm, since graph embeddings are not applicable here without significant algorithmic innovation (which is not the purpose of this paper), due to 'nodes' in the concept-set C not having any links to begin with. The core idea is to obtain a *model* from B, either directly (such as in the case of the pre-trained embedding described below) or indirectly (by training our own embedding on a domain-specific corpus, or retrofitting an existing embedding to a related or 'training' taxonomy, as was mentioned earlier and is also described below). During test-time, when the concept-set C is revealed or becomes available, this model, which we also generically denote as B in a slight abuse of notation, is used to map each concept c in C to a vector $\vec{c}$, which is dense and real-valued in modern representation learning frameworks, usually comprising a few hundred dimensions. In vector space, we now have a number of options to induce a tree. Below, we first describe the different forms that B can take, followed by how to *use* the vectors obtained by a model for taxonomy induction.

Pre-trained Embedding Model A *pre-trained embedding model* P, on which further details are provided in Section V-B, is perhaps the best example of B. Such models are trained on a large (usually, but not always, openly available) corpus, such as Wikipedia and Google Books. The 'corpora' undergo significant preprocessing, including chunking and removal of wayward characters and punctuation, and are generally input to a word embedding algorithm such as Glove or word2vec [5], [4] as multi-sets of sequences of words. The output is a vector $\vec{w}$ for word w in the sequence. In recent years, pre-trained models have been used with great success in a variety of tasks (see e.g., [8]).

Domain-specific Embedding Model. If the domain is sufficiently different, the words in the corpus used for pre-training could carry a very different meaning (at least in the statistical sense that is relevant for word embeddings), both in theory and practice, compared to ordinary parlance. Some words simply do not show up often enough for meaningful representations to be learned (especially in legal, commercial and biomedical domains). It has become commonplace to 'pre-train' the embedding on a domain-specific corpus in order for it to be useful in those domains. Examples include BioBERT and Law2Vec [21], [22]. However, many domains, such as e-commerce, to which taxonomy induction tends to be applied in practice share considerable semantic overlap with Wikipedia or the Google news corpus. In part, this is expected because the users and customers in these domains are ordinary people, who may get confused if a word suddenly takes on a different meaning. Nevertheless, there are confounds, since pre-trained models may not make some important distinctions e.g., between the fruit 'apple' and the company 'Apple'.

For this reason, it is also worthwhile considering (as another

potential approach) an embedding trained from scratch on a domain-specific corpus D. In this formulation, the background resource is not the pre-trained embedding model P but the corpus D, which has been acquired from a domain-specific Web resource. The embedding trained on D can then be used to infer links in the same way as P (described subsequently in the *Using the Vectors for Taxonomy Induction* sub-section).

Retrofitted Embedding. Finally, we consider another taxonomy, called the 'training' taxonomy T', as another background resource. The training taxonomy is not a 'true' training dataset because the concept-set on which T' was constructed is different from C. Hence, link prediction or knowledge graph embedding algorithms cannot be applied, due to the node disjointness problem (this bears resemblance to the cold-start problem observed sometimes in recommender systems [23]). Nevertheless, we assume that, by 'retrofitting' the pre-trained embeddings to another taxonomy from the domain, we may obtain a better embedding without re-training the whole embedding on a domain-specific corpus such as D. To this effect, we use the retrofitting algorithm implemented in [9] and traditionally used for retrofitting word embeddings to WordNet. The intuition behind that algorithm is relatively simple. For any two words in the pre-trained model P that are connected by an edge in the training taxonomy, the retrofitting algorithm's objective function 'pushes' the two vectors closer together. The algorithm is allowed to run for several iterations. In this way, the final word embeddings output by the algorithm capture not only the 'usual' semantics of the words captured in ordinary corpora, but the domain-specific semantics embodied in the training taxonomy. For example, if 'diapers' and 'baby clothes' are neighbors (or even a short distance away) in the training taxonomy, the retrofitting will bring their embeddings closer together than would otherwise be entailed in an off-the-shelf pre-trained model.

An important detail to mention here is how we treat *unknown words* and *multi-word* phrases (such as 'baby clothes') in either the training or test taxonomy. When dealing with unknown words and multi-word phrases in the test taxonomy, the approach adopted is similar for all embedding methods. If the concept occurs directly as a vocabulary item in the embeddings, we use that embedding. Otherwise, we average the individual word embeddings (of words in the phrase), ignoring unknown words.

Using the Vectors for Taxonomy Induction Once the vectors have been obtained, we have to use them for the LTC task. We propose an *Information Retrieval (IR) approach*, based on established methodology, for doing so. Technical details on parameter settings that will be relevant for the empirical study are described in the next section.

The IR approach treats each concept $c \in C$ as a *query* q similar to keyword and other queries issued to search engines and other similar systems in the IR literature. Specifically, given a query-concept $q \in C$, where C is the concept-set over which we must induce the local taxonomy around q, we can *rank* all members of $C - q$ in descending order of each member's *cosine similarity*[7] (of its embedding) and the query embedding $\vec{q}$. In Section V, we describe how IR metrics can be used to evaluate the ranked list in response to a query. This is a decentralized way of constructing the tree, since we evaluate the method by independently computing IR metrics for each query-concept, followed by averaging[8]. It is well-suited to the LTC problem, though whether it can be similarly extended to the global taxonomy induction problem is for future work to address.

V. Experiments

A. Data

We consider three taxonomies for evaluating our approach: *Google Product Taxonomy (GPT)*, *PriceGrabber* and *Walmart*. Key statistics are provided in Table I. The GPT is a list of thousands of 'product categories' designed by Google to uniformly categorize products in a shopping feed. It is publicly available at the following link[9] and has undergone some updates in recent years. We use the latest version for the experiments. PriceGrabber[10] is a 'smart shopping' website that helps customers find savings and discounts on a broad category of products. The PriceGrabber taxonomy can be downloaded through an API provided to affiliate partners (merchants or publishers of product links via reviews or other useful information for potential purchasers). Walmart refers to the website[11] of the well-known retailer of the same name; we obtained the taxonomy (which is technically a graph with a taxonomy-like structure) by crawling product pages starting from the sitemap[12]. We extract 'product paths' (e.g., 'Home/Appliances/Freezers') from these crawled pages, and re-construct the ground-truth taxonomy from the paths.

All three of these datasets are used heavily in the real world and offer alternate perspectives of a given domain (which may be broad, such as e-commerce and online shopping, and not easily definable using formal language). The three taxonomies share some similarities such as overlap in some product categories, but differ significantly both in size and the purposes to which they have been applied in practice. This allows us to assess the effectiveness and robustness of both the representation learning methods and the super-type classifiers (described in the next section) in a controlled setting.

B. Methods and Parameters

In Section III, we presented three methods that relied predominantly on representation learning, each with a slightly different rationale. We evaluate all three methods in this section to determine which one is appropriate for deriving a taxonomy

[7]Given two vectors, $\vec{a}$ and $\vec{b}$, the cosine similarity is $\frac{||\vec{a}.\vec{b}||}{||\vec{a}||||\vec{b}||}$

[8]Since every concept in C will be treated as a 'query' exactly once, the averaging will occur over $|C|$ computations of an IR metric such as the NDCG (Section V-C).

[9]https://support.google.com/merchants/answer/6324436?hl=en

[10]http://www.pricegrabber.com/

[11]https://www.walmart.com/

[12]The sitemap is itself a hierarchy and can be accessed at: https://www.walmart.com/robots.txt

TABLE I: Statistics on taxonomies used for the evaluation.

Name	Num. Concepts	Num. Edges	Avg. Num. Children/Node
GPT	5,582	5,561	6.36
PriceGrabber	948	947	12.62
Walmart	10,166	11,040	14.17

given only a concept set. Note that the retrofitting method relies on an example ('training') taxonomy. The methods and their parameterization are briefly enumerated below. Recall that each method can be thought of as outputting a ranked list of concepts, given a query concept. Earlier, Section IV described how we deal with phrases.

Method 1 (*Pre.*): Pre-trained Embeddings. We downloaded the English pre-trained fastText embeddings at the following link[13]. We believe that the 'bag-of-tricks' approach in fastText is particularly useful in our scenario (even compared to more transformer-based language models such as BERT) because its faster training (and re-training) speed allow us to perform uniform comparison between embeddings. Also, acquiring word vectors is relatively simple, and the model is robust to misspellings and uncommon words (but which are common in specific domains). The vocabulary size of the pre-trained embeddings is 1 million words and the model was trained on the combined corpus of Wikipedia 2017, UMBC WebBase corpus and the statmt.org news dataset (16B tokens).

Method 2 (*Dom.*): Domain-specific Corpus-trained Embeddings. We used the fastText model to obtain a domain-specific corpus-trained embedding model. We obtained the corpus as follows. First, we downloaded the *Product*-specific subset of schema.org data available on Web Data Commons[14].

Although not very well known outside the Web community, schema.org has seen steady growth as structured markup embedded within HTML webpages [24]. According to the schema.org website, over 10 million sites use Schema.org to markup their web pages and email messages. Many applications from Google, Microsoft, Pinterest, Yandex and others already use these vocabularies to faciliate a rich suite of applications. The WDC schema.org project, which relies on the webpages in the Common Crawl, is able to automatically extract schema.org data from webpages due to its unique syntax and make it available as a dataset in Resource Description Framework (RDF). Since text attributes are an important part of this data, we were able to use them to construct our domain-specific corpus, as described below.

Since the *Product*-specific component of the overall schema.org data on WDC is a massive, multi-lingual corpus (more than 100 GB+ in compressed form), we downloaded the first 20 chunks of the data (about 25 GB compressed), and extracted all the English language text literals from the product entities (by checking for @en and @en-US language tags). If a text literal is fully numeric, or only contains a URL (which sometimes appear within the quotes of text literals) we discard the literal. We also did standard[15] pre-processing by removing unicode symbols, newlines and tabs. The final uncompressed text corpus, after cleaning and preprocessing was 5 GB, a big enough corpus to train the fastText model. We used the skipgram model (with vector dimensionality set to 300, min and max context size set to 1 and 5 respectively, and all other parameters set to their default values) for training the model. While the original corpus had 744 million words, many of these only occurred once and were spelling variations of a more frequent word. Since fastText is capable of handling such variations within the model, we took the top 1 million frequently occurring word embeddings and discarded the rest for the experiments. Before using this model, we validated its quality for taxonomy induction qualitatively by retrieving nearest-neighbors of some domain-specific words such as 'appliance' and 'home' and verifying effectiveness.

Method 3 (*Retro.*): Retrofitted Embedding Transfer with Super-Type Classification. Finally, we use the retrofitting approach described in Section IV. As described in that section, we retrofit the embeddings in *Pre* using the (pre-processed) training taxonomy and the retrofitting algorithm described in [9]. We also obtain a binary *super-type classifier* (described further below) by training a random forest model. To use the super-type classifier, during the ranking phase (given a query concept), a two-step approach is adopted. First, we obtain the ranked list in descending order of cosine similarity scores, as described earlier. Next, we apply the classifier to determine the top[16] 3 most likely super-type candidates for the query node and move those top 3 candidates to the top of the ranked list. In preliminary experiments, this approach was found to yield empirical benefits to standard retrofitting.

We consider two 'supervised' methods that build on traditional machine learning classification and are only dependent on representation learning for their features. The idea behind the classifier is to predict the super-type of a concept node given a classifier model such as logistic regression or random forest. Since this is a supervised model, it needs to be trained, and is only applicable in the setting when a training taxonomy T' is available as background resource. Details are given below. We consider two different feature sets to fully assess the merits of the approach, described below.

Method 4 (S_{pre}): Super-type Classification using Pre-trained Embeddings. This model is trained using the pre-trained embeddings (*Pre*) as features. We use a random forest model with default parameters, but with number of estimators set to 200. We frame the problem as binary classification, using the difference of the embedding between the query node and a candidate node as the 'feature vector'. To train the model, we use the training taxonomy T'. Specifically, for each node

[13]https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip

[14]http://webdatacommons.org/structureddata/2018-12/stats/schema_org_subsets.html

[15]A key reference being https://fasttext.cc/docs/en/python-module.html#important-preprocessing-data--encoding-conventions.

[16]We tried multiple values for this 'reranking' parameter; 3 was found to work well across all datasets. Reranking was also found to yield superior results compared to using retrofittings without reranking.

c_i in T' that has a parent c_j (recall that there might be a set of upper-level nodes that may not have a parent), we construct the *translational* feature vector $\vec{c_j} - \vec{c_i}$ and assign it the positive label. We randomly sample a sibling of c_i (say, c_k) and assign the feature vector $\vec{c_k} - \vec{c_i}$ the negative label, thereby obtaining a balanced training set, using only concepts in T'. We obtain the full ranked list on the actual (i.e. the test) concept set C by ranking all candidates in $C - q$ (q being the query) according to the probability scores output by the classifier (higher probability indicating higher likelihood of that concept being a super-type of q).

Method 5 (S_{retro}): Super-type Classification using Retrofitted Embeddings. This model is identical to the above, except that we use the embeddings obtained from Method 3 (*Retro*) for deriving the features. This baseline also offers us the opportunity to study the difference between this method and Method 4 in a second-order context (where the embeddings are themselves input to a model), rather than used directly (via cosine similarity) for obtaining the rankings. Note that this model is also used in the second step of the two-step model previously described for *Retro*.

C. Methodology and Metrics

We stated in Section III that an Information Retrieval (IR)-based methodology could be used to evaluate the effectiveness of any ranking-based approach to local taxonomy construction. In the IR methodology, we treat each concept in the concept-set C as a *query*, and compute a ranking over all *other* concepts in the concept-set. All the methods described in the previous section can be tuned to return a ranking of all concepts, given a concept. Since we know the ground-truth taxonomy for the concept-set, we know which concepts in the ranking (for each 'query' concept) are 'relevant' (to use IR terminology); these are precisely the concepts that are neighbors of the query concept in the ground-truth taxonomy. IR is a well-studied field with several metrics; below, we consider two important ones.

Normalized Discounted Cumulative Gain (NDCG). To compute the NDCG, we first have to calculate the DCG for query q, defined by the following equation:

$$DCG_q = rel_1 + \sum_{p=2}^{n} \frac{rel_i}{log_2(i+1)} \quad (1)$$

Here, rel_i is the relevance of the i^{th} item in a ranked list of size n. In our case, this is either a 1 (if the concept at that rank is a neighbor of the query concept in the ground-truth taxonomy) or a 0. We can compute the DCG of both the actual ranking and of an *ideal* ranking (where all relevant items are ranked at the top), the latter denoted as the IDCG (Ideal DCG). The NDCG is then given by:

$$NDCG_q = \frac{DCG_q}{IDCG_q} \quad (2)$$

Since the DCG is always less than the IDCG, the NDCG is between 0 and 1. For the performance over the entire concept-set we average the NDCG over all queries.

Mean Average Precision (MAP). Intuitively similar to the NDCG in its theoretical rationale, the MAP is formally defined as the mean of the *Average Precision (AP)*, computed as follows. Given a ranked list of size n in response to query q and assuming m relevant items in the ground truth for q at ranks $r_1, \ldots, r_m$, we compute the *precision* at rank r_i as the total number of relevant items in the ranked sub-list $[1, \ldots, r_i]$ divided by r_i. The AP is the average of the precision computed at ranks $r_1, \ldots, r_m$. The MAP is the mean of all APs computed over the entire query set. More details on these metrics can be found in any standard reference on IR [18]. For each of the two metrics, we obtain a single number between 0 and 1 (with the higher number indicating better performance) *per query*. In principle, this permits us to compute the distribution of values, if necessary for further error analysis, though we only take the average over all queries for this study.

D. Results and Discussion

Table II contains the results for all combinations of training/test taxonomies using the IR-based methodology described in the previous section. From the table, we find that the maximum NDCG/MAP scores for each train/test setting always corresponds to one of the representation learning algorithms, as opposed to the super-type classifiers. This shows that, at least on a per-query basis, direct use of embeddings is a better choice than using the embeddings in a downstream machine learning model. Reinforcing this conclusion, the rank correlation between MAP and NDCG is also high i.e. if we rank the five approaches either by MAP or by NDCG we get very similar results.

Importantly, we observe that the retrofitting embedding *Retro* tends to outperform *Pre* when the training taxonomy is large (as in the case of Walmart) and when the testing taxonomy is small (as in the case of PriceGrabber). However, noise also plays a role, since the GPT taxonomy is (arguably) the 'cleanest' of all the taxonomies, having been meticulously designed at Google with several iterations over the years. We hypothesize that, for this reason, the best performance achieved by *Retro* is in the GPT/PriceGrabber setting rather than the Walmart/PriceGrabber setting, although it also does quite well (both absolutely and relatively) on the latter.

An overall comment that we make with respect to these results is that the numbers clearly show that there is usually at least one relevant entry in the top 3. Typically, this is necessary to achieve an NDCG or MAP of greater than 33%. This is encouraging, since it hints towards the feasibility of the problem. Furthermore, the inferior performance of *Dom.* compared to the other representation learning models suggests that that pre-trained embeddings are sufficient, and that it is not necessary to invest significant amounts of time in preprocessing and preparing domain-specific corpora for re-training embedding models.

TABLE II: Information retrieval metrics (NDCG/MAP) on each of the six train/test settings. Except for S_{pre} and *Retro* on the Walmart/PriceGrabber case (last row), all results are significant at the 95% level. Furthermore, except for *Dom* (GPT/PriceGrabber, Walmart/PriceGrabber), all results are significant at the 99% level. Maximum values at the row-level for each of the two metrics are in **bold**. Since not all methods use the 'training' taxonomy, some results may be repeated.

Train	**Test**	*Retro.*	*Pre.*	*Dom.*	S_{pre}	S_{retro}
GPT	PriceGrabber	**0.44/0.30**	0.38/0.23	0.37/0.22	0.26/0.10	0.34/0.18
GPT	Walmart	0.27/0.13	**0.29/0.16**	**0.29**/0.15	0.14/0.023	0.19/0.07
PriceGrabber	GPT	0.34/0.18	**0.38/0.23**	0.37/0.21	0.17/0.04	0.23/0.096
PriceGrabber	Walmart	0.25/0.11	**0.29/0.16**	**0.29**/0.15	0.13/0.017	0.18/0.056
Walmart	GPT	0.35/0.20	**0.38/0.23**	0.37/0.21	0.16/0.03	0.29/0.156
Walmart	PriceGrabber	**0.39/0.25**	0.38/0.23	0.37/0.22	0.23/0.07	0.38/0.23

VI. Future Work and Conclusion

This paper described the local taxonomy construction problem, which is a more tractable version of the global taxonomy induction problem, but still significantly more difficult than ordinary link prediction. We also found that representation learning algorithms, derived and adapted from the NLP community, can be viable solutions for the problem, but that there is considerable room for improvement.

An important avenue of future work is to develop methodologies and algorithms that are applicable to the global taxonomy induction problem. As we stated earlier, this problem is very different from standard link prediction and triples classification problems (in the knowledge graph community) due to the complete lack of initial structure or network. In fact, it is not even completely clear how one would evaluate such an induced taxonomy with respect to a 'ground-truth' taxonomy. IR cannot be applied in an obvious way, unlike the LTC problem described here, and neither can measures like tree edit distance and graph edit distance due to their quadratic time and space complexity (in the number of nodes). Hence, new metrics may be required to evaluate these algorithms.

References

[1] R. Snow, D. Jurafsky, and A. Y. Ng, "Semantic taxonomy induction from heterogenous evidence," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2006, pp. 801–808.

[2] Y. Mao, X. Ren, J. Shen, X. Gu, and J. Han, "End-to-end reinforcement learning for automatic taxonomy induction," *arXiv preprint arXiv:1805.04044*, 2018.

[3] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[6] L. Liu, X. Ren, Q. Zhu, S. Zhi, H. Gui, H. Ji, and J. Han, "Heterogeneous supervision for relation extraction: A representation learning approach," *arXiv preprint arXiv:1707.00166*, 2017.

[7] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1454–1467, 2018.

[8] S. M. Rezaeinia, R. Rahmani, A. Ghodsi, and H. Veisi, "Sentiment analysis based on improved pre-trained word embeddings," *Expert Systems with Applications*, vol. 117, pp. 139–147, 2019.

[9] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith, "Retrofitting word vectors to semantic lexicons," *arXiv preprint arXiv:1411.4166*, 2014.

[10] N. Mrkšić, D. O. Séaghdha, B. Thomson, M. Gašić, L. Rojas-Barahona, P.-H. Su, D. Vandyke, T.-H. Wen, and S. Young, "Counter-fitting word vectors to linguistic constraints," *arXiv preprint arXiv:1603.00892*, 2016.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[12] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *arXiv preprint arXiv:1607.01759*, 2016.

[13] F. Zhuang, X. Cheng, P. Luo, S. J. Pan, and Q. He, "Supervised representation learning: Transfer learning with deep autoencoders," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[14] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[15] P. Ristoski and H. Paulheim, "Rdf2vec: Rdf graph embeddings for data mining," in *International Semantic Web Conference*. Springer, 2016, pp. 498–514.

[16] P. Velardi, S. Faralli, and R. Navigli, "Ontolearn reloaded: A graph-based algorithm for taxonomy induction," *Computational Linguistics*, vol. 39, no. 3, pp. 665–707, 2013.

[17] A. Gupta, R. Lebret, H. Harkous, and K. Aberer, "Taxonomy induction using hypernym subsequences," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 1329–1338.

[18] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press, 2008.

[19] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[20] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[21] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "Biobert: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[22] N. Kim and H. J. Kim, "A study on the law2vec model for searching related law," *Journal of Digital Contents Society*, vol. 18, no. 7, pp. 1419–1425, 2017.

[23] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Systems with Applications*, vol. 41, no. 4, pp. 2065–2073, 2014.

[24] R. V. Guha, D. Brickley, and S. Macbeth, "Schema. org: evolution of structured data on the web," *Communications of the ACM*, vol. 59, no. 2, pp. 44–51, 2016.