**AutoKnow: Self-Driving Knowledge Collection for Products of Thousands of Types**

Xin Luna Dong¹, Xiang He¹, Andrey Kan¹, Xian Li, Yan Liang¹, Jun Ma¹, Yifan Ethan Xu¹, Chenwei Zhang³, Tong Zhao¹, Gabriel Blanco Saldana¹, Saurabh Deshpande¹, Alexandre Michetti Manduca¹, Jay Ren¹, Surender Pal Singh¹, Fan Xiao¹, Haw-Shiuan Chang², Giannis Karamanolakis³, Yuning Mao⁴, Yaqing Wang⁵, Christos Faloutsos⁶, Andrew McCallum², Jiawei Han⁴

¹Amazon ²University of Massachusetts Amherst ³Columbia University ⁴University of Illinois at Urbana-Champaign ⁵State University of New York at Buffalo ⁶Carnegie Mellon University

**ABSTRACT**

Can one build a knowledge graph (KG) for all products in the world? Knowledge graphs have firmly established themselves as valuable sources of information for search and question answering, and it is natural to wonder if a KG can contain information about products offered at online retail sites. There have been several successful examples of generic KGs, but organizing information about products poses many additional challenges, including sparsity and noise of structured data for products, complexity of the domain with millions of product types and thousands of attributes, heterogeneity across large number of categories, as well as large and constantly growing number of products.

We describe AutoKnow, our automatic (self-driving) system that addresses these challenges. The system includes a suite of novel techniques for taxonomy construction, product property identification, knowledge extraction, anomaly detection, and synonym discovery. AutoKnow is (a) automatic, requiring little human intervention, (b) multi-scalable, scalable in multiple dimensions (many domains, many products, and many attributes), and (c) integrative, exploiting rich customer behavior logs. AutoKnow has been operational in collecting product knowledge for over 11K product types.

**CCS CONCEPTS**

- Information systems → Graph-based database models.

**KEYWORDS**

knowledge graphs, taxonomy enrichment, attribute importance, data imputation, data cleaning, synonym finding

---

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

*KDD ’20, August 23–27, 2020, Virtual Event, CA, USA* © 2020 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7998-4/20/08.

https://doi.org/10.1145/3394486.3403323

1 INTRODUCTION

A knowledge graph (KG) describes entities and relations between them; for example, between entities Amazon and Seattle, there can be a headquarters_located_at relation. The past decade has witnessed broad applications of KG in search (e.g., by Google and Bing) and question answering (e.g., by Amazon Alexa or Google Home).

How to automatically build a knowledge graph with comprehensive and high-quality data has been a hot topic for research and industry practice in recent years. In this paper, we answer this question for the Retail Product domain. Rich product knowledge can significantly improve e-Business shopping experiences through product search, recommendation, and navigation.

Existing industry success for knowledge curation is mainly for popular domains such as Music, Movie, and Sport [4, 12]. Two common features for such domains make them pioneer domains for knowledge collection. First, there are already rich data in structured form and of decent quality for these domains. Taking Movie as an example, in addition to common knowledge sources such as Wikipedia and WikiData, other authoritative movie data sources include IMDb¹, and so on. Second, the complexity of the domain schema is manageable. Continuing with the Movie domain, the Freebase knowledge graph [4] contains 52 entity types and 155 relationships [36] for this domain. An ontology to describe these types and relationships can be manually defined within weeks, especially by leveraging existing data sources.

The retail product domain presents a set of new challenges for knowledge collection.

---

¹https://www.imdb.com/
C1- Structure-sparsity: First, except for a few categories such as electronics, structured data are sparse and noisy across nearly all data sources. This is because the majority of product data reside in catalogs from e-Business websites such as Amazon, Ebay, and Walmart, and they often rely on data contributed by retailers. In contrast to publishers for digital products like movies and books, contrast to publishers for digital products like movies and books, in the retail business retailers mainly list product features in titles and descriptions instead of providing structured attribute information, and may even abuse those structured attribute fields for convenience in selling products [35, 37]. As a result, structured knowledge needs to be mined from textual product profiles (e.g., titles and descriptions). Thousands of product attributes, billions of existing products, and millions of new products emerging on a daily basis, require fully automatic and efficient knowledge discovery and update mechanisms.

C2- Domain-complexity: Second, the domain is much more complex. The number of product types is towards millions and there are various relationships between the types like sub-types (e.g., swimsuit vs. athletic swimwear), synonyms (e.g., swimsuit vs. bathsuit), and overlapping types (e.g., fashion swimwear vs. two-piece swimwear). Product attributes vastly differ between types (e.g., compare TVs and dog food), and also evolve over time (e.g., older TVs did not have WiFi connectivity). All of these make it hard to design comprehensive ontology and keep it up-to-date, thus calling for automatic solutions.

C3- Product-type-variety: Third, the variety of different product types makes it even harder to train knowledge enrichment and cleaning models. Product attributes, value vocabularies, text patterns in product titles and descriptions often differ for different types. Even neighboring product types can have different attributes; for example, Coffee and Tea, which share the same parent Drink, describe size using different vocabularies and patterns (e.g., “Ground Coffee, 20 Ounce Bag, Rainforest Alliance Certified” vs. “Classic Tea Variety Box, 48 Count (Pack of 1)”). On the one hand, training one single model is inadequate to achieve good results for all different types of products. On the other hand, collecting training data for each of the thousands to millions of product types is extremely expensive, and implausible for less-popular types. Maintaining a huge number of models also brings big system overhead.

With all of these challenges, the solutions in building existing KGs both in industry (e.g., Google Knowledge Graph, Bing Satori Graph [12]), and in research literature (e.g., Yago [10], NELL [6], Diadem [11], Knowledge Vault [8]), cannot directly apply to the retail product domain, as we will further discuss in Section 7.

In this paper, we present our solution, which we call AutoKnow (Figure 1). AutoKnow starts with building product type taxonomy (i.e., types and hypernym relationships) and deciding applicable product attributes for each type; after that, it imputes structured attributes, cleans up noisy values, and identifies synonym expressions for attribute values. Imagine how an autonomous-driving vehicle perceives and understands the environment using all the signals available with minimized human interventions. AutoKnow is self-driving with the following features.

- **Automatic**: First, it trains machine learning models and requires very little manual efforts. In addition, we leverage existing Catalog data and customer behavior logs to generate training data, eliminating the need for manual labeling for the majority of the models and allowing extension to new domains without extra efforts.
- **Multi-scalable**: Our system is scalable in multiple dimensions. It is extensible to new values and is not constrained to existing vocabularies in the training data. It is extensible to new types, as it trains one-size-fits-all models for thousands of types, and the models behave differently for different product types to achieve the best results.
- **Integrative**: Finally, the system applies self-guidance, and uses customer behavior logs to identify important product attributes to focus efforts on.

A few techniques play a critical role to allow us to scale up to the large number of product types we need to generate knowledge for. First, we leverage the graph structure that naturally applies to knowledge graphs (entities can be considered as nodes and relationships can be considered as edges) and taxonomy (types and hypernym relationships form a tree structure), and apply Graph Neural Network (GNN) for learning. Second, we take product categorization as input signals to train our models, and combine our tasks with product categorization for multi-task training to allow better performance. Third, we strive to learn with limited labels to alleviate the burden of manual training data creation, relying heavily on weak supervision (e.g., distant supervision) and on semi-supervised learning. Fourth, we mine both facts and heterogeneous expressions for the same concept (i.e., type, attribute value) from customer behavior logs, abundant in the retail domain.

More specifically, we make the following contributions.

1. **Operational system**: We describe AutoKnow, a comprehensive end-to-end solution for product knowledge collection, covering components from ontology construction and enrichment, to data extraction, cleaning, and normalization. A large part of AUTO KNOW has been deployed and operational in collecting over 1B product knowledge facts for over 11K distinct product types, and the knowledge has been used for Amazon search and product detail pages.

2. **Technical novelty**: We invented a suite of novel techniques that together allow us to scale up knowledge discovery to thousands of product types. The techniques range from NLP
and graph mining to anomaly detection, and leverage state-of-the-art techniques in GNN, transformer, and multi-task learning.

(3) Empirical study: We describe our practice on real-world e-Business data from Amazon, showing that we are able to extend the existing ontology by 2.9X, and considerably increase the quality of structured data, on average improving precision by 7.6% and recall by 16.4%.

Whereas our paper focuses on retail domain and our experiments were conducted on Amazon data, the techniques can be easily applied to other e-Commerce datasets, and adapted to other domains with hierarchical taxonomy, rich text profiles, and customer behavior logs, such as finance, phylogenetics, and biomedical studies.

2 DEFINITION AND SYSTEM OVERVIEW

2.1 Product Knowledge Graph

A KG is a set of triples in the form of (subject, predicate, object). The subject is an entity with an ID, and this entity belongs to one or multiple types. The object can be an entity or an atomic value, such as a string or a number. The predicate describes the relation between the subject and the object. For example, (prod_id, has-Brand, brand_id) is a triple between two entities, whereas (prod_id, hasSugarsPerServing, “32”) is a triple between an entity and an atomic value. One can consider the entities and atomic values as nodes in the graph, and predicates as edges that connect the nodes.

For simplicity of problem definition, in this paper we focus on a special type of knowledge graph, which we call a broad graph. The broad graph is a bipartite graph $G = (N_1, N_2, E)$, where nodes in $N_1$ represent entities of one particular type, called the topic type, nodes in $N_2$ represent attribute values (that can be entities or atomic values), and edges in $E$ connect each entity with its attribute values.

The edges are labeled with corresponding attribute names (Figure 1). In other words, a broad graph contains only two layers, and thus contains attribute values only for entities of the topic type. We focus on broad graphs where the topic type is product. Once a broad graph is built, one can imagine stacking broad graphs layer by layer to include knowledge about other types of entities (e.g., brand), and eventually arrive at a rich, comprehensive graph.

Product types form a tree-structured taxonomy, where the root represents all products, each node represents a product type, and each edge represents a sub-type relationship. For example, Coffee is a sub-type of Drink.

We assume two sources of input. First, we assume existence of a product Catalog, including a product taxonomy, a set of product attributes (not necessarily distinguished between different product types), a set of products, and attribute values for each product. We assume that each product has a product profile that includes title, description, and bullet points, and in addition a set of structured values, where title is required and other fields are optional. Second, we assume existence of customer behavior logs, such as the query and purchase log, customer reviews, and Q&A. We next formally define the problem we solve in this paper.

Problem definition: Let $C = (T, A, P)$ be a product Catalog, where (1) $T = (T, H)$ denotes a product taxonomy with a set of product types $T$ and the hypernym relationships $H$ between types in $T$; (2) $A$ denotes a set of product attributes, and (3) $P = \{\text{PID}, (T), \{(A, V)\}\}$ contains for each product (PID is the ID) a set of product types $\{T\}$ and a set of attribute-value pairs $\{(A, V)\}$. Let $L$ denote customer behavior logs. Product Knowledge Discovery takes $C$ and $L$ as input, enriches the product knowledge by adding new types and hypernym relationships to $T$, and new product types and attribute values for each product in $P$.

2.2 System Architecture

Figure 2 depicts the architecture of our AutoKnow system. It has five components, categorized into two function suites.

Ontology suite: The ontology suite contains two components: Taxonomy enrichment and Relation discovery. Taxonomy enrichment identifies new product types not existing in input taxonomy $T$ and decides the hypernym relationships between the newly discovered types and existing types, using them to enrich $T$. Relation discovery decides for each product type $T \in T$ and attribute $A \in A$, whether $A$ applies to type $T$ and if so, how important $A$ is when customers make purchase decisions for these products, captured by an importance score.

Data suite: The data suite contains three components: Data imputation, Data cleaning, and Synonym discovery. Data imputation derives new (attribute, value) pairs for each product in $P$ from product profiles and existing structured attributes. Data cleaning identifies anomalies from existing data in $P$ and newly imputed values. Synonym discovery associates synonyms between product types and attribute values.

Each component is independent, automatic and multi-scalable; on the other hand, the components are well pieced together. The early components provide guidance and richer data for later components; for example, relation discovery identifies important and meaningful relations for the data suite, and data imputation provides richer data for synonym discovery. The later components fix errors from early parts of the pipeline; for example, data cleaning removes mistakes from data imputation.

Table 1 summarizes how each component of AutoKnow employs the aforementioned techniques. We next describe in Section 3 how we build the ontology and in Section 4 how we improve the data. To facilitate understanding of our design choices, for each component we present comparison of our proposed solution with the state-of-the-arts, show ablation study, and show real examples in Appendix A. Unless otherwise mentioned, we use the Grocery domain in the US market and the flavor attribute to illustrate our results, but we have observed the same trend throughout different domains and attributes. We describe detail of the experimental...
setup and end-to-end empirical study in Section 5 and lessons we learned in Section 6.

3 ENRICHING THE ONTOLOGY

3.1 Taxonomy Enrichment

Problem definition: Online catalog taxonomies are often built and maintained manually by human experts (e.g., taxonomists), which is labor-intensive and hard to scale up, leaving out fine-grained product types. How to discover new product types and attach them to the existing taxonomy in a scalable, automated fashion is critical to address the C2- Domain-complexity challenge.

Formally, given an existing product taxonomy \( T = (T, H) \), Taxonomy Enrichment extends it with \( T' = (T \cup T', H \cup H') \), where \( T' \) is the set of new product types, and \( H' \) is the additional set of hypernym relationships between types in \( T \) and in \( T' \).

Key techniques: The product domain proposes its unique challenges for taxonomy construction. A product type and its instances, or a type and its sub-types, are unlikely to be mentioned in the same sentence as in other domains such as "big US cities like Seattle", so traditional methods like Hearst patterns do not apply. Our key intuition is that since product types are very important, they are frequently mentioned in product titles (see Table 2 for an example) and search queries (i.e., "k-cups dunkin donuts dark"); we thus leverage existing resources such as product profiles in the Catalog C or search queries in behavior logs L to effectively supervise the taxonomy enrichment process.

We enrich product taxonomy in two steps. We first discover new types \( T' \) from product titles or customer search queries by training a type extractor. Then, we attach candidate types in \( T' \) to the existing taxonomy \( T \) by solving a hypernym classification problem. We next briefly describe high-level ideas of each step and details can be found in [22].

Type extraction: Type extraction discovers new product types mentioned in product titles or search queries. For the purpose of recognizing new product types from product titles, it is critical that we are able to extract types not included in training data. Thus, we adopt an open-world tagging model and formulate type extraction as a "BIOE" sequential labeling problem. In particular, given the product’s title sequence \((x_1, x_2, ..., x_L)\), the model outputs the sequence of \((y_1, y_2, ..., y_L)\), where \( y_i \in \{B, I, O, E\} \), representing "begin", "inside", "outside", "end" respectively. Table 2 illustrates an example of sequential labels obtained using OpenTag [37]: "ice cream" is labeled as product type, and "black cherry cheesecake" as product flavor.

To train the model, we adopt distant supervision to generate the training labels. For product titles, we look for product types in Catalog provided by retailers (restricted to the existing product types), and generate BIOE tags when types are explicitly and exactly mentioned in their titles. For queries, we look for the type of purchased products in the query to generate BIOE tags. Once the extraction models are trained, we apply them on product titles and queries. New types from titles are taken as \( T' \), and those from queries, albeit noisier, will be used for type attachment.

Type Attachment: Type attachment organizes extracted types into the existing taxonomy. We thus solve a binary classification problem, where the classifier determines if the hypernym relationship exists between two types \( T \in T, T' \in T' \).

Our key intuition is to capture various signals from customer behaviors with a GNN-based module. In particular, we first construct a graph where the nodes represent types, products, and queries, and the edges represent various relationships including 1) product co-viewing, 2) a query leading to a product purchase, 3) the type mentioned in a query or a product (according to the extraction). The GNN model allows us to refine the node representation using the neighborhood information on the graph. Finally, the type representation for each \( T \in T \cup T' \) is combined with semantic features (e.g., word embedding) of the type names and fed to the classifier.

To train the model, we again apply distant supervision. We use the type hypernym pairs in the existing taxonomy as the supervision to generate positive labels, and generate five negative labels by randomly replacing the hyponym type with other product types.

Component Evaluation: For product type extraction in the Grocery domain, we obtained 87.5% precision according to MTurk evaluation; in comparison to state-of-the-art techniques, Noun Phrase (NP) chunking obtains 12.3% precision and AutoPhrase [30] obtains 20.9% precision.

For type attachment, we took hypernym relationships from existing taxonomy as ground truth, randomly sampled 80% for model training, 10% as validation set and 10% for testing. We measured both Edge-F1 (F-measure on parent-child relationship) and Ancestor-F1 (F-measure on ancestor-child relationship) [2, 21]. Table 3 shows that our GNN-based model significantly outperforms the state-of-the-art baselines, improving Edge-F1 by 54.3% over HiDir [33], and by 17.7% over MSejrKu [29]. Ablation tests show that both the multi-hop GNN and the user behavior increase the performance.

\begin{table}[h]
\centering
\caption{Key techniques employed by each component.}
\begin{tabular}{|c|c|c|c|c|}
\hline
Component & AK-Taxonomy & AK-Relations & AK-Imputation & AK-Cleaning & AK-Synonyms \\
\hline
Graph structure & X & X & & & \\
Taxonomy signal & & X & X & & \\
Distant supervision & & & X & X & \\
Behavior information & & & & X & \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{Example of input(text)/output(BIOE tag) sequences for the type and flavor of an ice cream product.}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
Input & Ben & Jerry’s & black & cherry & cheesecake & ice cream \\
Output & O & O & O & B-flavor & I-flavor & E-flavor & B-type & E-type \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{AK-Taxonomy improves over state-of-the-art by 17.7% on Edge-F1.}
\begin{tabular}{|c|c|c|}
\hline
Method & Edge-F1 & Ancestor-F1 \\
\hline
Substr [5] & 10.7 & 52.9 \\
HiDir [33] & 40.5 & 66.4 \\
MSejrKu [29] & 53.1 & 76.7 \\
\hline
Type-Attachment & 62.5 (84.2) & 75.9 (83.3) \\
\hline
\end{tabular}
\end{table}
3.2 Relation Discovery

Problem definition: In a catalog there are often thousands of product attributes; however, different sets of attributes apply to different product types (e.g., flavor applies to snacks, but not to shampoos), and among them, only a small portion have a big influence on customer shopping decisions (e.g., brand is more likely to affect shopping decisions for snacks, but less for fruits). Understanding applicability and importance will help filter values for non-applicable attributes and prioritize enrichment and cleaning for important attributes. Thus, how to identify applicable and important attributes for thousands of types is another key problem to solve to address the C2- Domain-complexity challenge.

Formally, given a product taxonomy $\mathcal{T} = (\mathcal{T}, \mathcal{H})$ and a set of product attributes $\mathcal{A}$, Relation Discovery decides for each $(T, A) \in \mathcal{T} \times \mathcal{A}$, (1) whether attribute $A$ applies to products of type $T$, denoted by $(T, A) \rightarrow \{0, 1\}$, and (2) how important $A$ is for purchase decisions on products of $T$, denoted by $(T, A) \rightarrow [0, 1]$. Here, we do not consider newly extracted types in $\mathcal{T}$, since they are often sparse.

Key techniques: Intuitively, important attributes will be frequently mentioned by sellers and buyers, whereas non-applicable attributes will appear rarely. Previous approaches explored this intuition, but either leveraged only one text source at a time (e.g., only customer reviews) or combined sources according to a pre-defined rule [15, 27, 31]. Here we train a classification model to decide attribute applicability, and a regression model to decide attribute importance. We use Random Forest for both models and employ two types of features reflecting behavior of the customers.

- Seller behavior, captured by coverage of attribute values for a particular product type, and frequency of mentioning attribute values in product profiles.
- Buyer behavior, captured by frequency of mentioning attribute values in search queries, reviews, Q&A sessions, etc.

For a given $(T, A)$ pair, we compute features that correspond to different signals (e.g., reviews, search logs, etc.). To this end, we estimate frequencies of mentions of attribute values in the corresponding text sources (see details in Appendix B). Note that sellers are required to provide certain applicable attributes (e.g., barcode). These attributes have high coverage, but they are not always important for shopping decisions and appear rarely in customer logs. We thus train two different models for applicability and importance to capture such subtle differences.

We collect manual annotations for training, both in-house and using MTurk. In the latter case, for a given $(T, A)$ pair, we asked six MTurk workers whether the attribute $A$ applies to products of type $T$, and how likely $A$ will influence shopping decisions for products of type $T$. The applicability is decided by majority voting, and importance is decided by averaging influence likelihood grades. Once we trained the model, we apply it to all $(T, A)$ pairs to decide applicability and importance.

Component Evaluation: We collected two datasets. The first dataset contains 807 applicability and importance labels for 11 common attributes (e.g., brand, flavor, scent) and 79 randomly sampled product types. The second dataset contains 240 applicability labels for 7 product types (e.g., Shampoo, Coffee, Battery) and 180 attributes for which there are values in the Catalog. We combined the data, used 80% for training and 20% testing, and reported results in Table 4: AK-Relations outperforms using only coverage features on both applicability prediction (by 4.7%) and importance prediction (1.9X). Here “Seller features” does not include the “Coverage features”.

Table 4: AK-Relations outperforms using only coverage features on both applicability prediction (by 4.7%) and importance prediction (1.9X). Here “Seller features” does not include the “Coverage features”.

| Method          | Precision | Recall | F1  | Spearman |
|-----------------|-----------|--------|-----|----------|
| Coverage features | 0.90      | 0.80   | 0.85 | 0.39     |
| Seller features  | 0.90      | 0.84   | 0.87 | 0.72     |
| Buyer features   | 0.86      | 0.83   | 0.84 | 0.68     |
| All features     | 0.94      | 0.84   | 0.89 | 0.74     |

4 ENRICHING AND CLEANING KNOWLEDGE

4.1 Data Imputation

Problem definition: The Data Imputation component addresses the C1- Structure-sparsity challenge by extracting structured values from product profiles to increase coverage. Formally, given product information (PID, $(T, (A, V)))$, Data imputation extracts new $(A, V)$ pairs for each product from its profiles (i.e., title, description, and bullets).

State-of-the-art techniques have solved the problem for a type-attribute pair $(T, A)$, obtaining high extraction quality with BIOE sequential labeling combined with active learning [37]. Equation (1) shows sequential labeling with BiLSTM and CRF:

\[ (y_1, y_2, ..., y_L) = \text{CRF} (\text{BiLSTM}(e_{x_1}, e_{x_2}, ..., e_{x_L})), \]

where $e_{x_i}$ is the embedding of $x_i$, usually initialized with pre-trained word embedding such as GloVe [25], and fine-tuned during model training. As an example, the output sequence tags in Table 2 shows that “black cherry cheesecake” is a flavor of the product.

However, the technique does not scale up to thousands to millions of product types and tens to hundreds of attributes that apply to each type. How to train an extraction model that acknowledges the differences between different product types is critical to scale up sequential labeling to address the C3- Product-type-variety challenge.

Key techniques: We proposed an innovative taxonomy-aware sequence tagging approach that makes predictions conditioned on the product type. We describe the high-level ideas next and details can be found in [16].

We extended sequence tagging described in Equation (1) in two ways. First, we condition model predictions on product type $T \in \mathcal{T}$:

\[ (y_1, y_2, ..., y_L) = \text{CRF} (\text{CondSelfAtt} (\text{BiLSTM}(e_{x_1}, e_{x_2}, ..., e_{x_L}), e_T)), \]

(2)

where $e_T$ is the pre-trained hyperbolic-space embedding (Poincare [23]) of product type $T$, known to preserve the hierarchical relation between taxonomy nodes. CondSelfAtt is the conditional self-attention layer that allows $e_T$ to influence the attention weights.

Second, to better identify tokens that indicate the product type, and address the problem that products can be mis-classified or product type information can be missing in a catalog, we employ multi-task learning: training sequence tagging and product categorization at the same time, with a shared BiLSTM layer.
Table 5: AK-Imputation improves over state-of-the-art by 10.1% on F1 for flavor extraction across 4,000 types.

| Model     | Vocab size | Precision | Recall | F1    |
|-----------|------------|-----------|--------|-------|
| BiLSTM-CRF [37] | 6756       | 70.3      | 49.6   | 57.5  |
| AK-Imputation  | 13093      | 70.9      | 57.8   | 63.3  |
| w/o. CondSelfAtt | 9528       | 74.5      | 53.2   | 61.5  |
| w/o. MultiTask  | 12558      | 68.8      | 57.0   | 61.9  |

We again adopt the distant supervision approach to automatically generate the training sequence labels by text matching between product profiles and available attribute values in the Catalog. The trained model is then applied to all (PID, A) pairs for predicting missing values.

**Component Evaluation:** In Table 5, we show the performance evaluation and ablation studies of flavor extraction across 4000 types of products in the Grocery domain. Compared to the baseline BiLSTM-CRF model adopted by current state-of-the-art [37], both CondSelfAtt and MultiTask learning, when applied alone, improve F1 score by at least 7.0%; combination of the two together improved F1 by 10.1%.

### 4.2 Data Cleaning

**Problem definition:** The structured data contributed by retailer are often error-prone because of misunderstood or intentional abuse of the product attributes. Detecting anomalies and removing them is thus another important aspect to address the C1-Structure-sparsity challenge. Formally, given product information (PID, {T}, {(A, V)}), Data cleaning identifies (A, V) pairs that are incorrect for the product, such as (A = flavor, V = "1 lb. box") for a box of chocolate and (A = color, V = "100% Cotton") for a shirt.

Abedjan et al. [1] have made successes in cleaning values of types like numerical and date/time. We focus our discussion on textual attributes, which are often considered as most challenging in cleaning. Similar to data imputation, the key is to address the C3-Product-type-variety challenge such that we can scale up to nearly millions of types. In particular, we need to answer the question: how to identify anomaly values inconsistent with product profiles for a large number of product types?

**Key techniques:** Our intuition is that an attribute value shall be consistent with the contexts provided by the product profiles. We propose a transformer-based [32] neural net model that jointly processes signals from textual product profile (D) and the product taxonomy T via a multi-head attention mechanism to decide if a triple (PID, A, V) is correct (i.e., whether V is the correct value of attribute A for product PID). The model is capable of learning from the raw textual input without extensive feature engineering, making it ideal for scaling to thousands of types.

The raw input of the model is the concatenation of token sequences in D, T and V. For the i-th token in the sequence, a learnable embedding vector $e_i$ is constructed by summing up three embedding vectors of the same dimension:

$$e_i = e_i^{FastText} + e_i^{Segment} + e_i^{Position},$$

where $e_i^{FastText}$ is the pre-trained FastText embedding [3] of token i, $e_i^{Segment}$ is a segment embedding vector that indicates to which source sequence (D, T or V) token i belongs, and $e_i^{Position}$ is a positional embedding [32] that indicates the relative location of token i in the sequence. The sequence of embeddings $[e_1, e_2, \ldots]$ is propagated through a multi-layer transformer model whose output embedding vector $e^{Out}$ captures the distilled representations of all three input sequences. Finally, $e^{Out}$ passes through a dense layer followed by a sigmoid node to produce a single score between 0 and 1, indicating the consistency of D and V; in other words, the likelihood of the input triple (PID, A, V) being correct (see Appendix C for details and Figure 4 for the model architecture).

To train the cleaning model, we adopt distant supervision to automatically generate training labels from the input Catalog. We generate positive examples by selecting high-frequency values that appear in multiple brands, then for each positive example we randomly apply one of the following three procedures to generate a negative example: 1) We build a vocabulary vocab(A) for each attribute A and replace a catalog value V of A with a randomly selected value from vocab(A); 2) We randomly select n-grams from the product title that does not contain the catalog value V, where n is a random number drawn according to the distribution of lengths of tokens in vocab(A); 3) We randomly pick the value of another attribute A’ ≠ A to replace V. At inference time, we apply our model to every (PID, A, V) triple and consider those with a low confidence as incorrect.

**Component Evaluation:** As shown in Table 6, evaluation on the flavor attribute for the Grocery domain on 2230 labels across 223 types shows that our model improves PRAUC over state-of-the-art anomaly detection technique [18] by 75.3%, and considering the product taxonomy in addition to product profiles improved PRAUC by 6.7%.

### 4.3 Synonym Finding

We finally very briefly discuss how we identify synonyms with the same semantic meaning, including spelling variants (e.g., Reese’s vs. reese), acronyms or abbreviation (e.g., FTOs vs. fair trade organic), and semantic equivalence (e.g., lite vs. low sugar). Aligning synonym values is another important aspect to address the C1-Structure-sparsity challenge, and how to train a domain-specific model to distinguish identical values and highly-similar values is a key question to answer.

Our method has two stages. First, we apply collaborative filtering [17] on customer co-view behavior signals to retain product pairs with high similarity, and take their attribute values as candidate pairs for synonyms. Such a candidate set is very noisy, hence requires heavy filtering. Second, we train a simple logistic regression model to decide if a candidate pair has exactly the same mean-

| Model     | PRAUC | R@.7P | R@.8P | R@.9P | R@.95P |
|-----------|-------|-------|-------|-------|-------|
| Anomaly Detection [18] | 32.0  | 2.4   | 1.3   | 1.3   | 1.3   |
| AK-Cleaning  | 56.1  | 59.6  | 39.8  | 26.0  | 20.7  |
| w/o. Taxonomy | 52.6  | 52.6  | 36.2  | 22.4  | 3.0   |
| w/o. Taxonomy | 52.6  | 52.6  | 36.2  | 22.4  | 3.0   |
versa, and words shared by the two candidates. Between every two out of these three sets, edit distance and embedding similarity are computed and used as features.

An experiment on 2500 candidate pairs (143 positive; half used for training) shows a PRAUC of 0.83 on Grocery flavor; removing the distinct-word features will reduce the PRAUC to 0.79.

5 EXPERIMENTAL RESULTS

We now present our product knowledge graph (PG) produced by the AutoKnow pipeline. We show that we built a graph with over 1 billion triples for over 11K product types and significantly improved accuracy and completeness of the data. Note that we have already compared each individual component with state-of-the-art in previous sections, so here we only compare PG with the raw input data.

5.1 Input Data and Resulting Graph

Raw Data: AutoKnow takes Amazon Product Catalog, including the product type taxonomy, as input. We chose products from four domains (i.e., high-level categories): Grocery, Health, Beauty, and Baby. These domains have the commonality that they contain quite sparse structured data; on the other hand, the numbers of types and the sizes vary from domain to domain. We consider products that have at least one page view in a randomly chosen month.

Table 7 shows statistics of the domains. For each domain, there are hundreds to thousands of types in the Catalog taxonomy, and the median number of products per type is thousands to tens of thousands. Amazon Catalog contains thousands of attributes; however, for each individual product most of the attributes do not apply. Thus for each product, there are typically tens to hundreds of populated values. We say an attribute is covered by a type if at least one product of that type has a value in Catalog for the attribute. As shown in the statistics, each domain involves thousands of attributes that are fairly popular to all 4 domains, and evaluated each (domain, attribute) pairs on 200 samples (Table 10). Note that even though they are popular among all text-valued attributes except brand, the applicability is still fairly low (<10% most of the cases), showing the big diversity of each domain. We made three observations. First, we significantly increased the quality of the data (precision up by 7.6%, recall up by 16.4%, F-measure up by 14.1%, and defect rate down by 14.4%). Second, the quality of the generated data is often correlated with the precision of the raw data.

MoE increase for triples with attribute values. For triples involving products, we used product popularity weighted sampling.

Type triples: Table 9 shows the quality of product-type triples measured by MTurk workers on 300 labels per domain. MoE shows the margin of error with a confidence level of 95%. AutoKnow obtained an average precision of 87.7% and increased the number of types in each domain by 2.9X on average.

Attribute triples: We start with choosing three text-valued attributes that are fairly popular to all 4 domains, and evaluated each (domain, attribute) pairs on 200 samples (Table 10). Note that even though they are popular among all text-valued attributes except brand, the applicability is still fairly low (<10% most of the cases), showing the big diversity of each domain. We made three observations. First, we significantly increased the quality of the data (precision up by 7.6%, recall up by 16.4%, F-measure up by 14.1%, and defect rate down by 14.4%). Second, the quality of the generated data is often correlated with the precision of the raw data.

Table 7: Statistics for raw data used as input to AutoKnow.

| Product Domain | Grocery | Health | Beauty | Baby |
|----------------|---------|--------|--------|------|
| #types         | 3,169   | 1,850  | 990    | 697  |
| med. # products/type | 1,760   | 18,320 | 27,150 | 28,700 |
| #attributes    | 1,243   | 1,824  | 1,657  | 1,511 |
| med. # attrs/type | 113     | 195    | 228    | 206   |

Table 8: Aggregated statistics describing our PG on four product domains (Grocery, Baby, Beauty, Health).

|                     | Grocery | Health | Beauty | Baby | Avg |
|---------------------|---------|--------|--------|------|-----|
| #Triples            | 3.40%   | 2.71%  | 4.08%  | 4.32%| 3.9X|
| #Attributes         | 1.1X    | 3.9X   | 4.1X   | 2.4X | 2.9X|
| #Types              | 990     | 4102   | 4778   | 18,320|
| Avg                 | 84.60%  | 82.24% | 89.97% | 87.68%|
| Defect Rate         | 2.4X    | 3.40%  | 72.76% | 3.63%|
| MoE                 | 93.89%  | 91.74% | 93.89% | 90.13%|
| #Types              | 3368    | 4102   | 4368   | 4778 |
| increase            | 1.1X    | 3.9X   | 4.1X   | 2.4X |

Table 9: AutoKnow achieved 87.7% type precision and increased the number of types by 2.9X.
### Table 10: PG improves over input data on average by 7.6% (percentage point) on precision, 16.4% on recall, and 14.4% on defect rate, with average MoE of 6.56% on precision/recall.

| Attribute 1 | Attribute 2 | Attribute 3 |
|-------------|-------------|-------------|
| Grocery     | Input PG    | Input PG    | Input PG    |
| Applicability| 38.51%      | 7.53%       | 10.00%      |
| Precision   | 68.61%      | 82.59%      | 49.94%      |
| Recall      | 37.17%      | 83.15%      | 14.3%       |
| F-measure   | 48.22%      | 82.87%      | 2.78%       |
| Defect Rate | 62.91%      | 21.14%      | 98.58%      |
| Health      | Input PG    | Input PG    | Input PG    |
| Applicability| 1.33%       | 0.59%       | 57.92%      |
| Precision   | 70.54%      | 84.00%      | 59.11%      |
| Recall      | 59.69%      | 49.92%      | 69.50%      |
| F-measure   | 64.66%      | 62.62%      | 63.89%      |
| Defect Rate | 49.69%      | 52.34%      | 48.31%      |
| Beauty      | Input PG    | Input PG    | Input PG    |
| Applicability| 0.04%       | 0.54%       | 4.82%       |
| Precision   | 18.83%      | 48.00%      | 71.98%      |
| Recall      | 69.44%      | 69.44%      | 65.21%      |
| F-measure   | 29.62%      | 56.76%      | 68.43%      |
| Defect Rate | 82.35%      | 59.02%      | 40.19%      |
| Baby        | Input PG    | Input PG    | Input PG    |
| Applicability| 0.0011%     | 0.09%       | 55.82%      |
| Precision   | 1.45%       | 0.03%       | 8.22%       |
| Recall      | 9.79%       | 7.97%       | 3.83%       |
| F-measure   | 2.53%       | 0.06%       | 5.23%       |
| Defect Rate | 98.72%      | 99.97%      | 97.30%      |

### Table 11: AutoKnow cleaned 1.77M incorrect values for two attributes with a precision of 90%.

| Attribute 1 | Recall@ 90% | Recall@ 80% | #Removed | %Removed |
|-------------|-------------|-------------|----------|----------|
| Grocery     | 36.58%      | 58.20%      | 1,381,277| 55.06%   |
| Health      | 76.72%      | 85.93%      | 30,215   | 32.63%   |
| Beauty      | 94.33%      | 97.16%      | 10,926   | 62.31%   |
| Baby        | 87.24%      | 95.31%      | 2,673    | 66.17%   |

Table 12: Precision for triples representing relations between entities. Precision for synonym relations is reported on two representative attributes.

| Attribute 1 | Attribute 2 | Product Types |
|-------------|-------------|---------------|
| Precision   | 91.6%       | 93.7%         | 88.1%      |
| #Pairs      | 6,610       | 10,926        | 21,900     |

For example, the precision of the data in the Baby domain is very low; as a result, the PG data also have low precision. On the other hand, recall tends to have less effect; for example, Attribute 2 has a recall of 1.4% for Grocery, but we are able to boost it to 81.0% with reasonable precision (77.3%). Third, there is still huge space for improvement: the defect rate is at best 21.1%, and can be over 90% in certain cases. We also note that production requires high accuracy, so we trade recall with precision in the production system.

Next, we randomly chose 8 (type, attr) pairs for top-5 important attributes to report our results at a finer granularity (Table 18 in Appendix A). The attribute values are categorical (much smaller vocabulary) or binary, leading to higher extraction quality. AutoKnow obtained an average precision of 95.0% and improved the recall by 4.3X.

To highlight how data cleaning improves the data quality, we show in Table 11 the noisy values we have removed for the same two attributes as in Table 10. At a precision of 90% (i.e., 9 out of 10 removed values are indeed incorrect), we achieve a recall of 73.7% for Attribute 1 and 27.8% for Attribute 2. In total we removed 1.3M values for these two attributes, accounting for 21% of Catalog values and 64.6% of AK-Imputation.

### Relation triples:
Finally, we show precision of relation triples in Table 12, including hypernym relations between product types and synonym relations between attribute values. We observed very high precision for value synonyms (>90%) and fairly high for hypernyms (88.1%) when we consider attaching to any ancestor node (not necessarily to the leaf) as correct.

### 6 LESSONS WE LEARNED
There are many lessons in building the Product KG, pointing to interesting research directions.

**Non-tree product hierarchies:** First, we may need to fundamentally reconsider the way we model taxonomy and classify products. Common practice is to structure product types into a tree structure for "subtypes", and classify each product into a single (ideally leaf) type. However, we miss multiple parents; for example, "knife" has subtypes “chef’s knife”, “hunting knife”, which correspondingly is also a subtype of "Cutlery & Knife" and "Hunting kits". Also, there is often no clear cut for product types: one product can be both fashion swimwear and two-piece swimwear. In general, we need to extend concepts to model broadly subtype, synonym, and overlapping relationships; for each product, we can simply extract the type from its title, and infer other types according to the relationship between product types.

**Noisy data:** Second, the large volume of noises can deteriorate the performance of the imputation and cleaning models. This can be observed from our lower quality of knowledge in the Baby domain, caused by wrong product types and many inapplicable values in Catalog. We propose aggressive data cleaning before using the data for training, and training a multi-task end-to-end model that imputes missing values and identifies wrong or inapplicable values.

**More than text:** Third, product profile is not the only source of product knowledge. A study on randomly sampled 22 (type, attribute) pairs shows that 71.3% values can be derived from Amazon product profiles, an additional 3.8% can be derived from Amazon product images, and 24.9% have to be sought from external sources such as manufacturer websites. This observation hints that a promising direction is to enhance AutoKnow with image processing capabilities and web extraction.

### 7 RELATED WORK
Industry KG systems typically rely on manually defined ontology and curate knowledge from Wikipedia and a number of structured sources (e.g., Freebase [4], Bing Satori [12], YAGO [10], YAGO2 [14]). Research systems conduct web extraction, but again observing
pre-defined ontology and focus on named entities such as people, movies, companies (e.g., NELL [6], Knowledge Vault [8], DeepDive [7]). By comparison, this work extracts product knowledge from product profiles, where the structure, sparsity and noise level are very different from webpages; many attribute values are free texts or numbers instead of named entities. Incorporating taxonomy knowledge into machine learning models and utilizing customer behavior signals for supervision are two themes employed throughout this work to improve model performance.

The product knowledge graph described in [34] differs from our work as it focuses on training product embeddings to represent co-view/complement/substrate relationship defined therein, while this work focuses on collecting factual knowledge about products (e.g., product types and attribute values). Recent product property extraction systems [35, 37] apply tagging on product profiles, but consider a single product type. Web extraction systems [26, 28] extract product knowledge from semi-structured websites, and the techniques are orthogonal to ours.

In addition to end-to-end systems, there have been solutions for individual components, including ontology definition [4, 10], entity identification [10], relation extraction [20], hierarchical embedding [23], linkage [13, 24], and knowledge fusion [8, 9]. We apply these techniques whenever appropriate, and improve them to address the unique challenges for the product domain.

8 CONCLUSIONS

This paper describes our experience in building a broad knowledge graph for products of thousands of types. We applied a suite of ML methods to automate ontology construction, knowledge enrichment and cleaning for a large number of products with frequent changes. With these techniques we built a knowledge graph that significantly improves completeness, accuracy, and consistency of data comparing to Catalog. Our efforts also shed light on how we may further improve by going both broader and deeper in product graph construction.

REFERENCES

[1] Ziaawas Abdeljan, Xu Chu, Dong Deng, Raul Castro Fernandez, Bab F Baya, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? Proceedings of the VLDB Endowment. 9, 12 (2016), 993–1004.
[2] Mohit Bansal, David Burkett, Gerard De Melo, and Dan Klein. 2014. Structured Learning for Taxonomy Induction with Belief Propagation. In ACL.
[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. TACL. 5 (2017), 135–146.
[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Tomas Mikolov. 2017. Knowledge and Data Engineering.
[5] Georg Papadakis, Jonathan Svorey, Avgidor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. Proceedings of the VLDB Endowment 9, 9 (2016), 684–695.
[6] Jeremy Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In EMNLP. 1532–1543.
[7] Shengjie Sun, Dong Yang, Hongchun Zhang, Yanxu Chen, and Xiaonan Wang. 2012. DIADEM: domain-centric, intelligent, automated data extraction methodology. In SIGKDD.
[8] Tim Purone, Georg Gottlob, Giovanni Grasso, Omer Gunes, Xiaobian Gao, Armand Kravchenko, Giorgio Orri, Christian Schallhart, Andrew Sellers, and Cheng Wang. 2012. DIADEM: domain-centric, intelligent, automated data extraction methodology. In WWW. ACM, 267–270.
[9] Yuqing Gao, Jisheng Liang, Benjamin Han, Mohamed Yakout, and Ahmed Mohamed. 2018. Building a large-scale, accurate and fresh knowledge graph. In SIGKDD.
[10] Pankaj Guha, Adam Mladen, Rupesh Mehta, Jeyashankar Ramamirtham, Rajeev Rastogi, Sandeep Satpal, Shinivasan H Sengamedu, Ashwin Tengli, and Charu Tiwari. 2011. Web-scale information extraction with Vertex. In ICDE. 1209–1220.
[11] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. Artificial Intelligence 194 (2013), 26–61.
[12] Andrew Hopkinsson, Amit Guradesani, Dave Palffy, and Arpit Mittal. 2018. Demand-Weighted Completeness Prediction for a Knowledge Base. In NAACL. 200–207.
[13] Giannis Karamanolakis, Jun Ma, and Xin Luna Dong. 2020. TTract: Taxonomy-Aware Knowledge Extraction for Thousands of Product Categories. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics.
[14] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. c.com recommendations: Item-to-item collaborative filtering. IEEE Internet computing, 1, 1 (2003), 76–80.
[15] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In 2008 Eighth IEEE International Conference on Data Mining. IEEE, 413–422.
[16] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In ACL. 4478–4496.
[17] Colin Lockard, Xin Luna Dong, Arash Einolghozati, and Prashant Shiralkar. 2018. CERES: Distantly Supervised Relation Extraction from the Semi-structured Web. PVLDB (2018), 1084–1096.
[18] Xunie Mao, Xiang Ren, Jianming Shen, Xiaotao Gu, and Jiawei Han. 2018. End-to-End Reinforcement Learning for Automatic Taxonomy Induction. In ACL. 2462–2472.
[19] Xunie Mao, Tong Zhou, Andrey Kan, Chenwei Zhang, Xin Luna Dong, Christos Faloutsos, and Jiawei Han. 2020. OCETF: Online Catalog Taxonomy Enrichment with Self-Supervision. In SigKDD.
[20] Maximilian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In NIPS. 6358–6367.
[21] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Tomas Mikolov. 2017. Comparative analysis of approximate blocking techniques for entity resolution. Proceedings of the VLDB Endowment. 9, 9 (2016), 684–695.
[22] Jeremy Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In EMNLP. 1532–1543.
[23] Shengjie Sun, Xiang Ren, Jianming Shen, Xiaotao Gu, and Jiawei Han. 2012. DIADEM: domain-centric, intelligent, automated data extraction methodology. In SIGKDD.
[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in neural information processing systems. 5998–6008.
[25] Jingjing Wang, Changsung Kang, Yi Chang, and Jiawei Han. 2014. A hierarchical dirichlet model for taxonomy expansion for search engines. In WWW.
[26] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achanta. 2020. Product Knowledge Graph Embedding for E-commerce. In Proceedings of the 13th International Conference on Web Search and Data Mining. 672–680.
[27] Huimin Xu, Wenting Wang, Xinnian Mao, Xinyu Jiang, and Man Lan. 2019. Scaling up Open Tagging from Tens to Thousands: Comprehension Empowered Attribute Value Extraction from Product Title. In ACL. 6338–6347.
[28] Dongxu Zhang, Subhabrata Mukherjee, Colin Lockard, Xin Luna Dong, and Andrew McCallum. 2019. OPEN-E: Integrating Open Information Extraction and Knowledge Bases with Relation Inference. arXiv preprint arXiv:1904.12600 (2019).
[29] Guineg Zheng, Subhabrata Mukherjee, Xin Luna Dong, and Feifei Li. 2018. OpenTag: Open Attribute Value Extraction from Product Profiles. In SigKDD.
A EXAMPLES

Here we provide example outputs produced by each of the five components (Figure 2). Taxonomy enrichment and relation discovery results are shown in Tables 13, 14 and 15. Next, data imputation, cleaning and synonym finding results are shown in Figure 3 and Tables 16 and 17. Finally, we also show additional evaluation results for the entire pipeline on a sample of type-attribute pairs in Table 18 (see evaluation details in Section 5).

Table 13: Examples of type extraction results.

| Source | Text | Product Type |
|--------|------|--------------|
| Product | 4 Country Pasta Homemade Style Egg Pasta - 16-oz bag | Egg Pasta |
| Product | Hamburger Helper Lasagna Pasta, Four Cheese, 10.3 Ounce (Pack of 6) | Lasagna Pasta |
| Product | COFFEE MATE The Original Powder Coffee Creamer 35.3 Oz. Canister Non-dairy, Lactose Free, Gluten Free Creamer | Coffee Creamer |
| Query | mccormick paprika 8.5 ounce | paprika |
| Query | flax seeds raw | flax seeds |

Table 14: Examples of detected product type hypernyms.

| Child Type | Parent Type |
|------------|-------------|
| Coconut flour | Baking flours & meals |
| Tilapia | Fresh fish |
| Fresh cut carnations | Fresh cut flowers |
| Bock beers | Lager & pilsner beers |
| Pinto beans | Dried beans |

Table 15: Attributes identified as most important for two example types.

| Cereals | Shampoo |
|---------|---------|
| brand | brand |
| ingredients | hair type |
| flavor | number of items |
| number of items | ingredients |
| energy content | liquid volume |

B ATTRIBUTE APPLICABILITY AND IMPORTANCE

Recall that for each (product type, attribute) pair we need to identify whether the attribute applies and how important the attribute is (e.g., whether color applies to Shoes, and how important is color for Shoes). To this end, we independently train a Random Forest classifier to predict applicability and a Random Forest Regressor to predict importance scores (for applicable attributes). In both cases, we consider each (product type, attribute) pair as an instance, and we label a sample of such pairs with either applicability or importance labels (Section 3.2). Sample prediction results for attribute importance are shown in Table 15.

In both models, we use the same set of features that characterize how relevant the attribute is for the given product type. The first feature is coverage, which is the proportion of products that have a non-missing attribute value. Next, a range of features are based on frequency of attribute mentions in different text sources. Consider a text source s (e.g., product descriptions, reviews, etc.), and suppose that all products of the required type are indexed from 1 to n. Note that a particular product i, can have several pieces of text of type s (e.g., a product might have several reviews), and let l(s, i) denote the number of such pieces. A feature based on signal s is then defined as x(s) = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{1}{l(s, i)} \sum_{j=1}^{l(s, i)} M(s, i, j) \right) Here M(s, i, j) = 1 if the j-th piece of text of signal s associated with product i mentions the attribute (e.g., whether the j-th review of the i-th shoes mentions color), otherwise M(s, i, j) = 0.

We consider two implementations of M(s, i, j), and accordingly, for each s we compute two features. First, M(s, i, j) = 1 if text piece j contains attribute value of product i (e.g., whether the review for product i contains color of this product). Second, M(s, i, j) = 1 if
Table 16: Example errors found by the cleaning model.

| Product profile                                                                 | Attrib. | Attribute value                      |
|---------------------------------------------------------------------------------|---------|---------------------------------------|
| Love of Candy Bulk Candy - Pink Mint Chocolate Lentils - 6 lb Bag               | Flavor  | Pink                                  |
| Scott’s Cakes Dark Chocolate Fruit & Nut Cream Filling Candies with Burgundy Fois in a 1 Pound Snowflake Box | Flavor  | snowflake box                         |
| Lucky Baby - Baby Blanket Envelope Swaddle Winter Wrap Coral Fleece Newborn Blanket Sleeper Infant Stroller Wrap Toddlers Baby Sleeping Bag (color 1) | Flavor  | Color 1                               |
| ASUTRA Himalayan Sea Salt Body Scrub Exfoliator + Body Brush (Vitamin C), 12 oz | Scent   | vitamin c body scrub - 12oz & body brush |
| Folgers Simply Smooth Ground Coffee, 2 Count (Medium Roast), 31.1 Ounce         | Scent   | 2 Packages                           |

Table 17: Examples of discovered flavor and scent synonym pairs.

| flavor synonyms                      | scent synonyms                     |
|--------------------------------------|------------------------------------|
| herb and garlic                      | basil (sweet)                      |
| macadamia nut                        | rose flower                        |
| roasted oolong tea                  | aloe lubricant                      |
| decaffeinated honey lemon            | unscented                          |
| zero carb vanilla                    | moonlight path                     |
| herb & garlic                        | sweet basil                        |
| macadamia                            | rose                               |
| roasted oolong                        | aloe lube                          |
| decaf honey lemon                    | unscented                          |
| zero cal vanilla                      | moonlight path                     |

Table 18: AutoKnow obtained an average precision of 95.0% and improved the recall by 4.3X for important categorical/binary attributes.

| Scope | Prec | Recall | Recall gain |
|-------|------|--------|-------------|
| pair-1 | 91.05% | 70.18% | 7.3X        |
| pair-2 | 97.06% | 19.70% | 5.2X        |
| pair-3 | 97.12% | 36.13% | 1.3X        |
| pair-4 | 93.87% | 37.72% | 10.5X       |
| pair-5 | 95.88% | 25.01% | 1.2X        |
| pair-6 | 90.42% | 87.46% | 2.8X        |
| pair-7 | 97.97% | 55.95% | 1.4X        |
| pair-8 | 96.44% | 87.49% | 4.5X        |

text piece $j$ contains any common attribute value for this product type (i.e., whether the review for product $i$ contains any frequent color values among Shoes). We consider a value to be common if it is among the top 30 most frequent values within the given type. We consider several text signals (e.g., product titles, reviews, search queries, etc.) and compute 30 features as described above. Finally, for each feature we also consider an alternative where products are weighted by popularity, and thus in total we have 60 features.

C TAXONOMY-AWARE SEMANTIC CLEANING

The cleaning model detects whether or not a triple $(PID, A, V)$ is correct (i.e., whether $V$ is the correct value of attribute $A$ for product $PID$) by attending to its taxonomy node and semantic signals in product profile. Let $D = [d_1, \ldots, d_{n_D}], T = [t_1, \ldots, t_{n_T}]$ and $V = [v_1, \ldots, v_{n_V}]$ be the token sequences of the product description, product taxonomy, and target attribute value, respectively. We construct the input sequence $S$ by concatenating $D, T$ and $V$ and inserting special tokens “[CLS]” and “[SEP]” as follows:

$$S = \text{concat}([CLS], D, [SEP], T, [SEP], V) := [s_1, \ldots, s_{n_S}]$$  \hspace{1cm} (4)

where $n_S = n_D + n_T + n_V + 3$. We then map each $s_i \in S$ to an embedding vector $e_i \in \mathbb{R}^d$ as the summation of three embedding vectors of the same dimension $d$:

$$e_i = e_i^{\text{FastText}} + e_i^{\text{Segment}} + e_i^{\text{Position}}, \quad i = 1, \ldots, n_S$$  \hspace{1cm} (5)

where $e_i^{\text{FastText}}$ is the pretrained FastText embedding [32] of $s_i$, $e_i^{\text{Segment}}$ is a segment embedding vector defined as:

$$e_i^{\text{Segment}} = \begin{cases} e_i^D, & \text{if } s_i \in D \\ e_i^T, & \text{if } s_i \in T \\ e_i^V, & \text{if } s_i \in V, \end{cases}$$  \hspace{1cm} (6)

and $e_i^{\text{Position}}$ is the position embedding vector of the location of $s_i$ in the sequence (i.e. $i$), for which we adopt the same constructions used in [32]. Here $e_i^{\text{FastText}}$’s and $e_i^{\text{Segment}}$’s are frozen (not trainable), and $e_i^D, e_i^T, e_i^V$ are randomly initialized and jointly trained with other model parameters.

The embedding sequence $[e_i^{\text{FastText}}]_{i=1}^{n_S}$ is propagated through a multi-layer transformer model where number of layers, number of heads and hidden dimension are hyperparameters. The final embedding vector of the special token [CLS], denoted by $e^{\text{Out}}$, captures the distilled representations of all three input sequences. It is passed through a dense layer followed by a sigmoid node to produce a single score between 0 and 1, indicating the likelihood of the input triple (PID, A, V) being correct. See Figure 4 for an illustration of the model architecture.

In Table 16 we give examples of attribute value errors detected by the cleaning model.