Human in the Loop Enrichment of Product Graphs with Probabilistic Soft Logic

Marouene Sfar Gandoura
marouene.gandoura@relational.ai
RelationalAI

Zografoula Vagena
zografoula.vagena@relational.ai
RelationalAI

Nikolaos Vasiloglou
nik.vasiloglou@relational.ai
RelationalAI

ABSTRACT

Product graphs have emerged as a powerful tool for online retailers to enhance product semantic search, catalog navigation, and recommendations. Their versatility stems from the fact that they can uniformly store and represent different relationships between products, their attributes, concepts or abstractions etc, in an actionable form. Such information may come from many, heterogeneous, disparate, and mostly unstructured data sources, rendering the product graph creation task a major undertaking. Our work complements existing efforts on product graph creation, by enabling field experts to directly control the graph completion process. We focus on the subtask of enriching product graphs with product attributes and we employ statistical relational learning coupled with a novel human in the loop enhanced inference workflow based on Probabilistic Soft Logic (PSL), to reliably predict product-attribute relationships. Our preliminary experiments demonstrate the viability, practicality and effectiveness of our approach and its competitiveness comparing with alternative methods. As a by-product, our method generates probabilistic fact validity labels from an originally unlabeled database that can subsequently be leveraged by other graph completion methods.

KEYWORDS

product graph, statistical relational learning, text similarity

ACM Reference Format:
Marouene Sfar Gandoura, Zografoula Vagena, and Nikolaos Vasiloglou. 2020. Human in the Loop Enrichment of Product Graphs with Probabilistic Soft Logic. In *Proceedings of Knowledge Graphs and E-commerce (KDD'20).* ACM, San Diego, CA, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

E-Commerce catalogs are an essential tool for every retailer; they are the stepping stone for online purchases as they enable and facilitate semantic search, product recommendations, etc. Catalogs are usually created by sourcing data from sellers, suppliers and brands. However, the data is often incomplete, sometimes missing crucial bits of information that customers are looking for. Relevant information can be buried within raw text (e.g. titles, product descriptions) or images. Additional relevant information may exist on the internet in the form of product manuals, product reviews, blogs, social media sites, etc.

To tackle data completeness issues and to also take advantage of the abundance of relevant information on the web, many retailers [4, 13] are currently looking into creating product graphs, i.e. knowledge graphs that capture all the relevant information about each of their products and related entities.

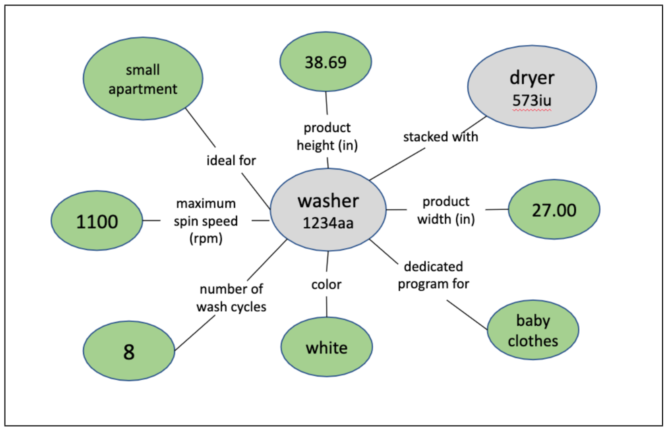


Figure 1: Product Graph for a Washer. Product Entities are in gray and attribute values in green

Product graphs can capture the different types of connections related to each product. For example, they can represent the relationships between a product and its parts or attributes (e.g. the color, and price of a fridge, the sensitivity of the speakers). They can also represent the connection between products and real world entities such as objects, things, concepts or abstractions that exist (e.g. kitchen, still life painting, bright colors, summer vacation, gluten-free, cottage style). Different types of relationships help answer different queries: the former can answer questions of the form "'silver washer"' while the latter can answer questions of the form "'backyard furniture"'. Additionally, the product graph can represent relationships among the products themselves, such as substitutes and complements (accessories, compatible products, etc). As an example in Figure 1 we show the projection of a product graph pertaining to a particular washer.

For a product graph to be useful it has to be complete (i.e. to cover and represent as much of the domain knowledge as possible) and correct (i.e. to contain accurate information and, if possible, coupled with actionable accuracy measures). Taking into consideration the many data sources that need to be consulted, their heterogeneity, their varying degrees or quality, as well as the sheer volume of data that need to be collected and consolidated it becomes obvious that product graph construction is a major undertaking. Currently ideas

KDD'20, August 2020, San Diego, CA, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and techniques from many domains, including schema matching [20], knowledge/information extraction and fusion [6, 26], entity resolution [18], named entity recognition [32], entity matching [15], and entity linking [12], have been leveraged and combined to extract the product graph related facts from the available sources. This extracted knowledge is usually incomplete (and at times inconsistent). Thus an additional step, usually referred to as knowledge graph completion, is needed. Current techniques either are heuristic (e.g. random walks), build involved statistical models [16], or can be too computationally expensive [29]. What's more, a major issue for model-based techniques is the need for a large amount of labeled data. This latter problem is typically addressed with weak supervision techniques [9, 19], ideas from which have motivated some of the solutions in this paper.

In our work we ameliorate the aforementioned product graph construction problems by enabling domain expertise to directly affect the modeling process. Our work bootstraps the process with zero labels through a *human in the loop* process that enriches an existing product graph with additional or corrected information about the product attributes for each of the existing products in the graph. We start with a product graph derived from a real database of washers from a large retailer and we enrich it using public information from other retailers' websites.

We use a scalable statistical relational learning framework called Probabilistic Soft Logic (PSL) [1] to enable domain knowledge to be injected in the defined enrichment model. The reason we use PSL is that it allows for expressive reasoning and ease of use by domain experts which are critical in our application scenario. Domain knowledge can be expressed in PSL in the form of logical rules that are easy to understand and reason against, while at the same time being expressive enough to represent different types of information. The underlying inference engine enables fast computations. The modeling process is enhanced with a novel, fully-automated human feedback workflow that achieves good accuracy at the product graph enrichment task.

The rest of the paper is organized as follows: in Section 2 we summarize recent related work on product (or knowledge) graph completion, while in Section 3 we provide some background information on PSL. In Section 4 we formally define the graph enrichment problem and in Section 5 we describe the new knowledge acquisition process. In Section 6 we present in detail the proposed solution and in Section 7 we present our experimental evaluation. Finally we conclude in Section 9 with some ideas for future work.

2 RELATED WORK

Statistical Relational Learning (SRL) [7] methods are concerned with the statistical analysis of relational, or graph-structured, data and naturally lend themselves to the study of knowledge graphs. Indeed a few proposals already exist [16] that utilize statistical graphical models to predict new facts. However those methods involve sophisticated statistical models that (1) are computationally expensive and (2) are not easily customized on specific domains. In our work we devise a novel framework that combines PSL, an efficient SRL method[1, 11], within an modeling architecture that (1) enables injecting domain knowledge directly in the model and (2) scales to large data volumes.

Various works have already appeared to tackle the graph completion issue for a general knowledge graph. An early work [17] employs tensor factorization to capture the structure of data in knowledge graphs. Another promising approach utilizes embedding-based methods that project a knowledge graph onto a continuous vector space while preserving key information and characteristics of the graph, and a host of relevant proposals [2, 14, 23, 24, 27, 29] have already appeared. Finally, methods based on deep convolutional [5, 22] or graph [31] neural networks have also been proposed. All these methods require a large number of training entity pairs for every relation. However, such training pairs are very often limited in practice, thus hampering the applicability of those methods.

Recently one-shot [28] and few-shot [30] learning frameworks have been proposed to substantially decrease the number of necessary training relations. Our approach is complementary to those efforts as it employs domain expert knowledge to eliminate the need for additional training information outside of existing relations.

3 BACKGROUND

In this section, we provide information on PSL, our main rule management and statistical inference engine. We first describe the underling statistical model and its inference process and then give specific examples of PSL programs.

3.1 Probabilistic Soft Logic (PSL)

Probabilistic Soft Logic (PSL) is a probabilistic programming language that uses a first-order logical syntax to define a graphical model [1]. PSL uses continuous random variables in the [0, 1] unit interval and specifies factors using convex functions, allowing tractable and efficient inference. It defines a Markov Random Field (MRF) associated with a conditional probability density function over random variables Y conditioned on evidence X,

$$P(Y|X) \propto \exp(-\sum_{j=1}^{m} w_j \phi_j(X,Y)) \tag{1}$$

where ϕ_j is a convex potential function and w_j is an associated weight which determines the importance of ϕ_j in the model. The potential ϕ_j takes the form of a hinge-loss:

$$\phi_j(Y|X) = (max\{0, l_j(X,Y)\})^{p_j} \tag{2}$$

Here, j is a linear function of X and Y, and $p_j \in 1,2$ optionally squares the potential, resulting in a squared-loss. The resulting probability distribution is log-concave in Y. PSL derives the objective function by translating logical rules specifying dependencies between variables and evidence into hinge-loss functions. PSL achieves this translation by using the Lukasiewicz norm and co-norm to provide a relaxation of Boolean logical connectives [11]:

$$\begin{aligned} p \wedge q &= max(0, p+q-1) \\ p \vee q &= min(1, p+q) \\ \neg p &= 1-p \end{aligned} \tag{3}$$

3.2 Anatomy of a PSL program

A PSL program[1] defines the rules and constraints that govern the dataset under consideration and consists of the following modules:

A set of logical rules. A logical rule is a sequence of atoms or negated atoms connected using the $\{or, and, \rightarrow\}$ connectives. Logical rules are either weighted or unweighted. If a logical rule is weighted, it is annotated with a non-negative weight. For example the rule:

```
10 : SameAttribute(att1, att2) and
     SameAttribute(att2, att3) ->
     SameAttribute(att1, att3)
```

captures the intuition that if there are three attributes called att1, att2, and att3, and we have strong evidence that att1 is the same attribute as att2 and att2 is the same as att3, then we can infer that att1 and att3 do refer to the same attribute as well. In other words, this is a transitivity rule where we have evidence for some values of the *SameAttribute* and we want to predict the values for the *SameAttribute* that are unknown. The number 10 in the beginning of the rule specifies the weight (or initial weight if weight learning is required) of the relevant potential function(s) in the underlying MRF.

A set of observations. Each observation is a grounded atom (i.e. a fact) accompanied with a number in the [0, 1] unit interval that represents the confidence of that fact.

A set of queries. Each query is a grounded atom (i.e. a fact) for which we want to estimate the confidence.

The ground truth (optional). If weight learning is required each query is also accompanied with a number in the [0, 1] unit interval that represents the confidence of that fact.

When grounded over a base of grounded atoms, a PSL program induces a HL-MRF conditioned on the specified observations.

4 PROBLEM DEFINITION

A product graph PG is represented as a collection of triplets $(p, r, a) \subseteq PxRxA$, where P is the set of products, A the set of product attribute values, and R the set of relations (i.e. the names of the attributes) between the products and the attribute values. Examples of relationship types are *washer depth*, *voltage*, *number of wash cycles*, *number or rinse cycles*, etc. Each triplet may be accompanied by a score which represents our confidence in the correctness of the triplet. We call those triplets *facts* from now on.

We define the product graph enrichment task as one of:

(1) Predict the product attribute value a given the product p and the relation $r : (p; r; ?)$,
(2) Predict the unseen relation r and attribute value for a product: $(p; ?; ?)$,
(3) Compute the confidence of a given fact.

[1]In all our experiments we used the software provided by the LINQS group https://github.com/linqs/psl

In our work we represent the product graph as a database table with the domain $(product, attribute, value)$. P comes from a database of a large retailer (we call this database the *anchor database* from now on) and includes all the products of a specific category (i.e. in this paper we project all the information about washers).

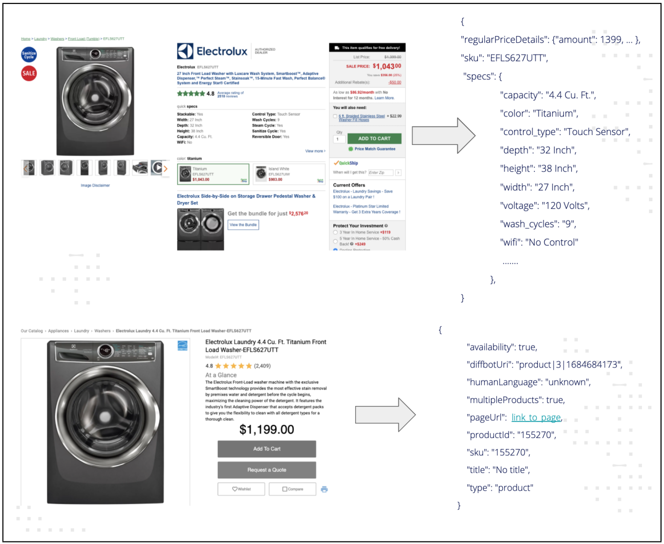
The candidate attributes and product-attribute relationships come from either the anchor database or can be discovered from an external data source, i.e. the websites of other retailers, which may have additional information about the products in the anchor database.

5 DATA ACQUISITION

The original anchor database contains 206 products (washers), 266 distinct product attribute names, and $34k$ facts. For simplicity and without loss of generality we opted to discard attributes that have long values or high cardinality like identifiers of images or videos. The curated database has 206 products, 210 distinct product attribute names, and $26k$ facts.

To create the dataset with additional relevant information (i.e. candidate attribute values) we crawl other retailers' websites and extract relevant $(product, attribute, value)$ facts from them. We will call this dataset the external dataset from now on. We used Diffbot [25], a service that uses computer vision and natural language processing techniques to parse a web-page and extract JSON formatted structured data from it, to create the external dataset. This service avoids the requirement of writing site-specific scrapers.

The data acquisition process is shown in the left part of Figure 2 and can be summarized as follows: For every product in the curated anchor database we issue the {model number, title} web keyword search query (using a popular search engine) and gather the top 5 returned URLs. We then call the Diffbot API on the acquired URLs and get a JSON file for each URL.



```
{
"regularPriceDetails": {"amount": 1399, ... },
"sku": "EFLS627UTT",
 "specs": {
        "capacity": "4.4 Cu. Ft.",
        "color": "Titanium",
        "control_type": "Touch Sensor",
        "depth": "32 Inch",
        "height": "38 Inch",
        "width": "27 Inch",
        "voltage": "120 Volts",
        "wash_cycles": "9",
        "wifi": "No Control"
         .......
      },
}
```

```
{
    "availability": true,
    "diffbotUri": "product|3|1684684173",
    "humanLanguage": "unknown",
    "multipleProducts": true,
    "pageUrl": link to page,
    "productId": "155270",
    "sku": "155270",
    "title": "No title",
    "type": "product"
  }
```

Figure 3: Top: Example of successful information product parsing. Bottom: Example of unsuccessful product information parsing.

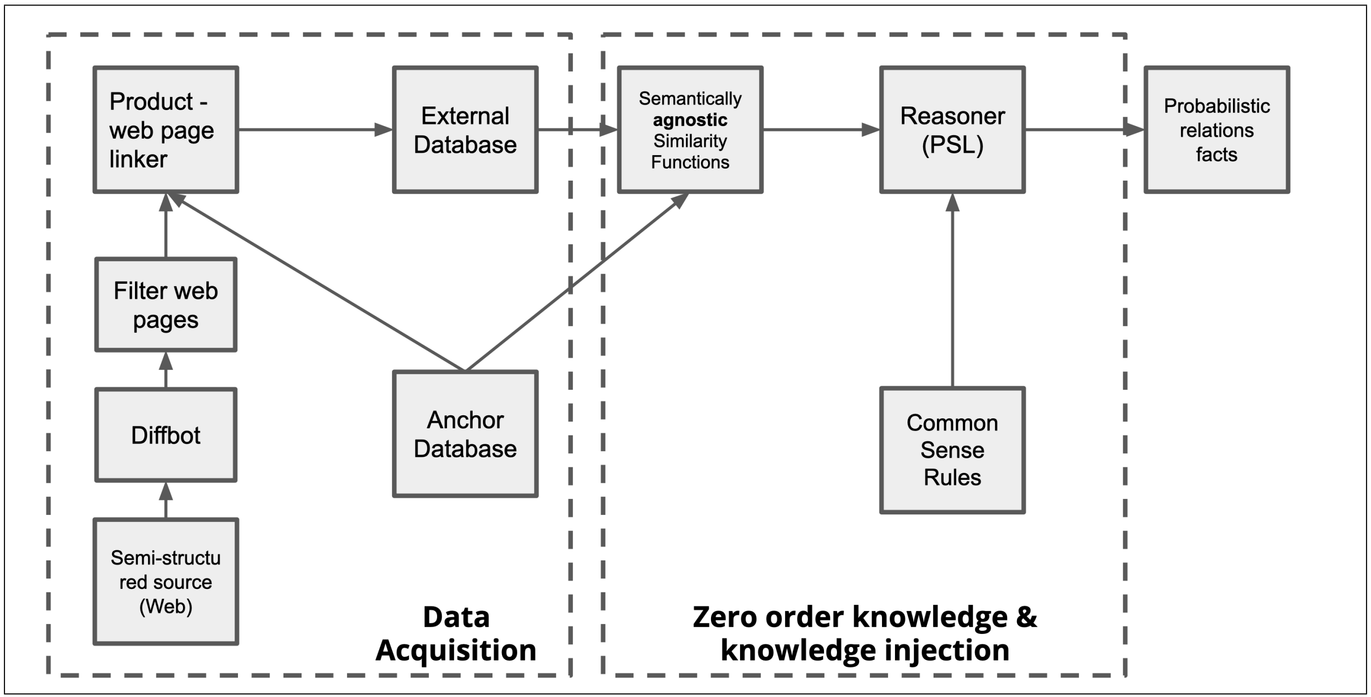

Product - web page linker
External Database
Filter web pages
Diffbot
Anchor Database
Semi-structu red source (Web)
Data Acquisition
Semantically agnostic Similarity Functions
Reasoner (PSL)
Probabilistic relations facts
Common Sense Rules
Zero order knowledge & knowledge injection

Figure 2: Proposed system architecture

Not all such files are useful i.e. contain product-attribute information. Refer to Figure 3 for actual example of useful and not useful retrieved files. To solve this issue we devised an automated filtering process that maintains as many relevant JSON files as possible.

For this purpose, we built an XGBoost[3] classifier that decides whether the JSON file should be kept or be discarded. The classifier uses the features below:

- Results from another classifier trained to distinguish washer product names from other text and used to score the page title,
- Number of attributes extracted from the page,
- Whether the page is from a list of retailers,
- Whether the page has a list of particular attributes (product number, product-name, etc.).

To automatically create labels for the classifier we used Snorkel, [21] a tool that produces probabilistic labels based on user-defined labeling functions (i.e. simple heuristics that generate noisy labels). The labeling functions that we defined take into consideration simple metrics such as the depth of the extracted JSON, the length of the attribute values, and the number of images that are extracted from the web page. The classifier achieved 87% AUC (tested against a set of about 200 hand-labeled pages).

As a final step we need to ensure that the information retrieved corresponds to the product under consideration (in some cases the search engine would not return the exact washer we inquired, but a similar model). For the purpose we defined a simple rule-based entity linker that compares the web page extractions (provided by Diffbot) and the product under consideration using the product name, manufacturer name, and product model number (whenever Diffbot can extract this information). If the web page extraction doesn't contain these attributes we discard it.

The resulting external dataset contains $32k$ extractions. An extraction is a $(source, product, attribute, value)$ tuple. Those cover $17k$ distinct $(product, attribute, value)$ triplets and $14k$ distinct $(product, attribute)$ pairs. The total number of unique attributes in the external database is 1639.

6 FACT GENERATION

Using the external dataset that we collected, we proceed to discover facts from it as well as assign confidence scores to those facts. A major component of this process is matching attribute instances in the external dataset with those that exist in the anchor database. These matches create the bootstrap seed of the fact generation process. Matching is based on (1) similarity scores between attributes and (2) logical rules that employ those scores along with domain knowledge to create facts. Our solution is divided into two phases that are executed in sequence to perform the aforementioned tasks. We call the first phase the creation of *zero order knowledge* and the second the *knowledge injection* phase.

6.1 Zero order knowledge

It is expected that the anchor database will not contain the exact same attributes as the acquired external dataset, due to different information encoded in the two data sources. Moreover even if the same attribute exists in the two data sources they may be encoded in a different way (e.g. due to different attribute names, format etc). In this paper we assume that we have no prior knowledge about attribute type semantics.

In order to create such a prior we resort to similarity scores between attributes. These similarity functions are generic and they have no semantic context of the domain we apply them to (washers). For that reason we call this step *zero order knowledge.* Such similarity scores are computed on the attribute names or on the attribute values. We conjecture that the similarity between the sets of values that two attributes can take signals similarity between the pair of attributes themselves. To leverage this fact, for each attribute we compute the bag of words of all the values that are assigned to it, vectorized using tf-idf. Subsequently for each pair of attributes we compute a similarity function between the relevant vectors using the Cosine and Radial basis function kernels.

As an example, the attributes *"Number of Wash Cycles"* and *"Number of Rinse Cycles"* have a Jaccard similarity of 0.6 base on the attribute name. The list of values these two attribute can take are *"10 9 20 16 ..."* and *"5 3 11 13 ..."* respectively. If we compute the tf*idf vectors of these two lists of values and compute the cosine similarity of these two vectors we get 0.72.

The similarity functions that we have employed are:

- Hamming distance
- Levenshtein distance
- Jaccard distance
- Cosine distance
- Word Mover's distance [10]

Word Mover's distance is a promising tool as it enables similarity scoring both on the semantic and syntactic levels. Nonetheless, it is computationally very expensive and would be impractical to use in our situation where we need to compute similarities for $3M$ attribute pairs. Thus we first compute attribute similarities using only the Hamming, Levenshtein, Jaccard, and cosine distances and only maintain the top $70k$ most similar attribute pairs. We then also compute the Word Mover's distance between those $70k$ pairs.

6.2 Knowledge Injection

In the *knowledge injection* phase we employ the similarities that we computed previously to discover facts from the external dataset. This phase is split into two stages. In the first stage we match product attribute names between the anchor database and the external dataset. In the second stage we assign confidence scores to candidate $(product, attribute, value)$ facts and check the validity of the highest scored facts.

6.2.1 *Stage 1: Attribute matching.*

The goal of the first stage is to assign affinity scores for pairs of (attribute1_name, attribute2_name). We use a PSL program to reason about the computed similarities, and compute affinity scores for the $70k$ attribute pairs created in section 6.1. The PSL program enforces two rules.

(1) similarity implies sameness
(2) transitivity of sameness

Transitivity and recursive rules in general, are an important part of our work. They help lower the affinity score of pairs of attributes that are hard to disambiguate (e.g. by looking at their similarities) like *product width* and *product depth* or that can only be compared based on their relationship with other attributes.

The PSL rules we used are

```
SimilarAttribute(att1, att2, similarity) ->
    SameAttribute(att1, att2)

SameAttribute(att1, att2) and
    SameAttribute(att2, att3) ->
    SameAttribute(att1, att3)
```

`SimilarAttribute` is an observed predicate and we are predicting `SameAttribute`.

After running stage 1, we end up with affinity scores for attribute pairs according to how much they satisfy the rules that we want to enforce (through the PSL program) and use these results in the subsequent stage to assign confidence scores to candidate facts.

6.2.2 *Stage 2: Scoring facts.*

Using the `SameAttribute` predicate that we computed from stage 1, we construct a set of possible product records and we run a second PSL program to score each candidate fact (product,attribute,value).

We construct this set of candidate facts by joining the facts in the anchor database with the facts in the external dataset using the previously discovered same attribute pairs as join keys. This set may contain multiple facts for the same {product, attribute_name} pair. To further disambiguate the facts we compute textual similarities like Hamming, Jaccard, and Cosine between pairs of attribute values that appear in the candidate set of facts.

We use all this information in a PSL program that computes confidence scores for facts:

```
AnchorRecord(prod-id, att1, val1) and
    SameAttribute(att1, att2) ->
    ProductRecord(prod-id, att2, val1)

AnchorRecord(prod-id, att1, val1) and
    ExternalRecord(prod-id, att2, val2) and
    SameAttribute(att1, att2) and
    SimilarValue(val1, val2, similarity) ->
    ProductRecord(prod-id, att2, val2)

ProductRecord(prod-id, att1, val1) and
    ProductRecord(prod-id, att2, val2) and
    SameAttribute(att1, att2) ->
    SameValue(val1, val2)

ProductRecord(prod-id, att1, val1) and
    SameAttribute(att1, att2) and
    SameValue(val1, val2) ->
    ProductRecord(prod-id, att2, val2)
```

In the previous rules, `AnchorRecord`, `ExternalRecord`, `SameAttribute` and `SimilarValue` are observations.
`AnchorRecord` and `ExternalRecord` represent facts in the anchor database and the external dataset respectively.
`SameAttribute` is the predicate computed in the previous stage.
`SimilarValues` represents normalized similarity values between the attribute values.
`ProductRecord` and `SameValue` are the unobserved predicates that we are predicting.

As we explain in detail in Section 6.2.3 the two stages are executed multiple times (iterations).

Table 1: Example of scored (product, attribute, value) fact

product	attribute	value	score
1	Certifications and Listings	CSA Listed	1.0
2	Returnable	Non-Returnable	0.99
3	Appliance Type	Top Load Washer	0.98
3	Amperage (amps)	15	0.85
5	Washer Tub/Drum Material	Stainless steel	0.75

6.2.3 *Human in the Loop.*

Due to uncertainties in the data selection and matching process as well as the probabilistic nature of the PSL programs we do not expect the latter to return 100% accurate results. Thus we employ PSL programs to narrow down the facts that are likely to be true to a manageable scale, so that a human expert can quickly provide feedback. After running the PSL program in stage 2 we end up with facts and their confidence scores (we show examples of such facts in table 1).

For each {product, attribute_name} pair we pick the relevant fact with the highest confidence score. If this fact has the same value as the matched fact in the anchor database then we are certain about its correctness. If it does not have the same value or if we don't have a fact for that {product, attribute_name} pair in the anchor database we manually check the correctness of the extracted fact.

The extracted facts are fed as input of the next iteration. That significantly improves the inference of PSL, as will be explained in section 7.

We also score the attribute pairs that do not perform well. For example we noticed that *"Product Width (in.)"* for every product ends up having on average many distinct values. For example for product *"13"* and attribute *"Product Width (in.)"* we get

(1) *"46" "* extracted through *"product_width"* attribute
(2) *"27" "* extracted through *"product_depth"* attribute
(3) *"27.5 in"* extracted through *"packaged_depth"* attribute
(4) *"35 1/2 in"* extracted through *"product_height"* attribute

This is a sign that the attribute name in the anchor database has been matched with the wrong attribute names in the external dataset. For example, *"Product Width (in.)"* tends to be confused with *"product_depth"* because they have similar attribute names and attribute values.

The human expert introduces these negations in the next iteration so that the quality of the produced facts is further improved.

7 EXPERIMENTAL EVALUATION

In section 6.2 we presented an iterative process where a PSL program generates high confidence candidate facts and a human expert preserves the valid ones among the former. Bear in mind that some of those facts already exist in the anchor database while others are completely new.

Due to the lack of a ground truth for all the generated facts we opted to check the performance of the process using the generated facts that also exist in the anchor database. For those facts the iterative process has to match potentially different references of the underlying entities (e.g due to typos, synonyms, format differences). For example the depth of a washer can be reported as $('washer09123','depth', 30)$ in the anchor database and as part of a dimensions attribute $('washer09123','dimensions','27x35x30')$ in the external dataset. We call those facts the validated facts.

Table 2: Fact based metrics for the proposed methodology with 0.3 threshold

Iteration	Cumulative extracted facts	Precision	Recall	F1 score
1	3731	50.84%	72.67%	59.82%
2	3846	76.39%	60.59%	67.57%
3	4051	85.3%	53.21%	65.53%

Table 3: Fact based metrics for the proposed methodology with 0.5 threshold

Iteration	Cumulative extracted facts	Precision	Recall	F1 score
1	3052	65%	59.43%	62.09%
2	3261	83.2%	52.5%	64.37%
3	3574	98.2%	46.7%	63.29%

Table 4: Fact based metrics for the proposed methodology with 0.7 threshold

Iteration	Cumulative extracted facts	Precision	Recall	F1 score
1	2653	64.83%	51.67%	57.5%
2	2892	89.45%	45.89%	60.65%
3	3076	99.6%	41.38%	58.46%

After running the iterative process over the anchor database and the external dataset 4 times we extracted 3574 validated facts. In table 3 we present the cumulative number of validated facts collected on every iteration after stage 2.

For this experiment, we used the *LazyMPEInference* inference method for PSL, with a maximum number of inference round of *20*, and for each round a maximum number of iterations of *5000* for the *ADMMReasoner*.

To extract attribute values from the external dataset we proceed as follows: For each (product, attribute_name) pair in the anchor database we keep the attribute value (from the external dataset) with the highest PSL confidence. If that confidence is greater than a chosen threshold we choose the attribute value from the external database for that particular (product, attribute_name) pair. We then employ a human expert to manually confirm the generated valid facts. We repeated the confirmation process for 5135 facts.

We then compute the *precision*, *recall* and *f1 score*: if the extracted valid fact is confirmed by the human expert we consider it a true positive, otherwise a false positive. As a false negative we consider the case when we do not extract any value for a (product, attribute_name) pair with confidence greater than the chosen threshold but there exists a fact in the external dataset where the attribute name in the anchor database and attribute name in the external dataset and the human expert accepts this fact. The results for different thresholds can be found in tables 2, 3 and 4.

Table 5: XGBoost performance on fact sameness classification

Precision	Recall	F1	AUC
95.28%	90.38%	90.92%	99.5%

7.1 Comparison with other methods

As a next step we devised two alternative enrichment methods, one based on (1) keyword matching and the other on (2) classification, and measured their performance.

7.1.1 Approximate keyword matching. The algorithm comprises the following steps:

(1) For every attribute in the anchor database we compute all the similarities under consideration with all the attributes in the external dataset.
(2) For each pair of attributes if the lowest similarity value that is computed is > 0.85 then we consider the two attributes to be the same.
(3) We do the same analysis for the attribute values.
(4) If both (2) and (3) are positive we return the relevant fact from the external dataset.

This heuristic has a very high precision (99.45%) because we only match attributes and values if they are almost the same, but it has low recall (18.35%) and doesn't perform as well as PSL in finding attribute and value pairs that are the same but do not look the same. For example using keyword matching we cannot predict that *'depth'* and *'dimensions'* are the same attribute or *'Digital Controls'* and *'Electronic LED'* are the same values for the *'Controls'* attribute.

7.1.2 XGBoost classifier. We trained a classifier using XGBoost to predict whether two facts (one extracted from the external dataset and one from the anchor database) are the same. We use as features the similarities between anchor-attribute, external-attribute and between anchor-attribute_value, external-attribute_value. We used the labels that we constructed from our PSL runs. The results can be found in table 5.

Although the final, overall performance of XGBoost is better, this method depends on the PSL-based iterated process for the labels. Once those are generated, XGBoost could be used in coordination with the iterative process to further improve enrichment performance.

8 ABLATION STUDY

In the final experiment we investigate the effectiveness of the proposed method at enriching the anchor database with brand new facts.

In order to do that, and in the absence of a ground truth for such new facts, we remove some facts from the anchor database and check if and how well our method discovers them from the external database.

In this study we employ the same metrics as in section 7.1, on the removed (product, attribute_name) pairs in computing the precision, recall and f1 scores and use 0.5 as our confidence threshold.

Table 6: Fact metrics on ablated data

Number of ablated facts	Precision	Recall	F1
1500	88.45%	51.46%	65.06%
2000	90.81%	43.62%	58.92%
2500	92.19%	41.15%	56.9%

In table 6 we present the results for different numbers of facts removed. The results indicate that the proposed method performs equally well in finding brand new facts in the external dataset. Thus the method is effective in not only gaining confidence in facts that exist in our anchor database, but also in enriching the anchor database with new facts from the external dataset.

9 CONCLUSION

In this paper we introduced a novel methodology for enriching product graphs that seamlessly combines smart web extractors (Diffbot), modern SRL tools (PSL), and human feedback in a practical way. Our preliminary experiments demonstrate that it is possible to enrich existing retail product graphs with both more precise facts as well as brand new facts and also augment those facts with confidence scores on their correctness. Using a real world dataset, we also generated a labeled dataset (as a by-product the proposed methodology) for washers that we intend to open source. We also intend to conduct further experiments on other product categories. One of the limitations of the current architecture is that it uses semantically agnostic similarity functions. This is because we start the enrichment of the product graph without any prior knowledge of the attribute semantics. It is possible to utilize the product descriptions to acquire more domain-specific embeddings. The relationships produced by the 3rd stage of the PSL model could also be leveraged to produce domain-specific embeddings using graph embedding algorithms [8]. We plan to investigate all of these directions as part of our future work.

REFERENCES

[1] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2017. Hinge-loss Markov Random Fields and Probabilistic Soft Logic. *Journal of Machine Learning Research (JMLR)* 18, 9 (2017), 1–67.

[2] Antoine Border, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NIPS*.

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[4] Karthik Deivasigamani. 2020. Retail Graph — Walmart's Product Knowledge Graph. https://medium.com/walmartlabs/retail-graph-walmarts-product-knowledge-graph-6ef7357963bc

[5] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *AAAI*.

[6] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge Vault: A Web-Scale Approach to Probabilistic Knowledge Fusion. *KDD* (2014).

[7] Lise Getoor and Ben Taskar. 2007. Statistical relational learning.

[8] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[9] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 541–550.

[10] Gao Huang, Chuan Guo, Matt J Kusner, Yu Sun, Fei Sha, and Kilian Q Weinberger. 2016. Supervised word mover's distance. In *Advances in Neural Information Processing Systems*. 4862–4870.

[11] Angelika Kimmig, Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2017. A short introduction to probabilistic soft logic. In *NIPS Workshop on Probabilistic Programming*.

[12] Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. 2018. End-to-End Neural Entity Linking. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Association for Computational Linguistics, Brussels, Belgium, 519–529. https://doi.org/10.18653/v1/K18-1050

[13] Arun Krishnan. 2018. Making search easier - How Amazon's Product Graph is helping customers find products more easily. https://blog.aboutamazon.com/innovation/making-search-easier

[14] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *AAAI*.

[15] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. *SIGMOD* (2018).

[16] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A Review of Relational Machine Learningfor Knowledge Graphs. *Proc. IEEE* 104, 1 (2016), 11–33.

[17] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *ICML*.

[18] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. 2018. The return of JedAI: End-to-End Entity Resolutionfor Structured and Semi-Structured Data. *VLDB* (2018).

[19] Marius Pasca. 2011. Web-based open-domain information extraction. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2605–2606.

[20] Erhard Rahm and Philip A. Bernstein. 2007. A survey of approaches to automatic schema matching. *The VLDB Journal* 10 (2007), 334–350.

[21] Alexander J Ratner, Stephen H Bach, Henry R Ehrenberg, and Chris Ré. 2017. Snorkel: Fast training set generation for information extraction. In *Proceedings of the 2017 ACM international conference on management of data*. 1683–1686.

[22] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC*.

[23] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. 2013. Reasoning With Neural Tensor Networksfor Knowledge Base Completion. In *NIPS*.

[24] Theo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillame Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *ICML*.

[25] Michael Upshall. 2016. Extracting Meaning from Web Content. *eLucidate* 13, 1 (2016).

[26] Xiaolan Wang, Xin Luna Dong, Alexandra Meliou, and Yang Li. 2019. MIDAS: Using the Wealth of Web Sources to Fill Knowledge Gaps. *ICDE* (2019).

[27] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAIA*.

[28] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. 2018. One-Shot Relational Learning for Knowledge Graphs. In *EMNLP*.

[29] Bishan Yang, Wen tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *ICLR*.

[30] Chuxu Zhang, Huaxiu Yao, Chao Huang, Meng Jiang, Zhenhui Li, and Nitesh V. Chawla. 2020. Few-Shot Knowledge Graph Completion. In *AAAI*.

[31] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. 2020. Efficient Probabilistic Logic Reasoning With Graph Neural Networks. In *ICLR*.

[32] Guineng Zheng, Subhabrata Mukherjee, Xin Luna Dong, and Feifei Li. 2018. OpenTag: Open Attribute Value Extraction from Product Profiles. In *KDD*.