# Product Taxonomy Matching in E-Commerce Environments

Master Thesis

presented by
Steffen Schmitz
Matriculation Number 1623016

submitted to the
Data and Web Science Group
Prof. Dr. Christian Bizer
University of Mannheim

May 2020

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| BoW | Bag-of-Words |
| CS | Cosine Similarity |
| DOM | Document Object Model |
| GPC | GS1 Product Catalog |
| HTML | HyperText Markup Language |
| JSON | JavaScript Object Notation |
| MLP | Multi-Layer Perceptron |
| OAEI | Ontology Alignment Evaluation Initiative |
| PD | Path Distance |
| PLD | Pay-Level Domain |
| RDF | Resource Description Framework |
| SGD | Stochastic Gradient Descent |
| SMOTE | Synthetic Minority Oversampling TEchnique |
| SVM | Support Vector Machine |
| URI | Unique Resource Identifier |
| WDC | Web Data Commons |
| XML | eXtensible Markup Language |

# Chapter 1

# Introduction and Motivation

Taxonomy matching, a particular case of ontology matching, is a recurring topic among researchers. Numerous novel approaches are tested in this domain and some achieve surprising results. Yet, most of them are applied to artificial datasets that do not represent the real world. In general, the goal of taxonomy matching is to create an alignment between two distinct categorization systems, e.g., the taxonomy of a library and a book shop. Although they may sell the same products, they probably use different approaches to catalog them. The outputs of the taxonomy matching algorithm create a semantic bridge between the labels used by both stores.

The Ontology Alignment Evaluation Initiative[1] (OAEI) exists since 2004 and publishes an ever-increasing number of datasets that can be used for the evaluation and comparison of ontology alignment algorithms. Contributions leveraging those datasets are numerous and are presented at the International Semantic Web Conference[2]. The existence of the OAEI establishes a need for large, real-world ontology matching datasets that challenge researchers and enable the development of new ontology alignment methods.

The Web Data Commons[3] (WDC) project shows that an increasing number of e-commerce websites annotate their pages semantically using schema.org and other shared vocabularies. These make a given webpage understandable to a machine and enable comparability of information across different domains. As a result, a large corpus of product information with annotated categories/classes that embody real-world taxonomies is available to everyone. Moreover, the Semantic Web offers a vast corpus for the ontology matching task. Each website may use its own dedicated taxonomy. This presents us with a possibly large, real-world dataset

[1] `http://oaei.ontologymatching.org`. Accessed: 01.05.2020
[2] `https://iswc2020.semanticweb.org`. Accessed: 01.05.2020
[3] `http://www.webdatacommons.org`. Accessed: 01.05.2020

that we can use for the evaluation of taxonomy matching algorithms.

Avesani et al. [8] as well as Angerman and Ramzan [6] identify the necessity of large-scale taxonomy matching datasets that represent the real-world. They state that most researchers use small or artificial datasets and that there is no agreed-upon reference dataset. Angerman and Ramzan explicitly express the need for freely available, large-scale datasets.

Hence, it is the goal of this Thesis to investigate if the semantic annotations can be leveraged to create an evaluation dataset for ontology matching tasks that stems from publicly available data. In this Thesis, we will create a gold standard of labelled class pairs that are part of multiple, distinct, real-world taxonomies and evaluate numerous taxonomy matching algorithms on this gold standard. The alignment tasks that the OAEI focuses on are mainly concerned with closest match problems. Instead, we build on the ideas of Giunchiglia et al. [14] and provide semantic labels. They predict not only equality of classes, but also if one class is more general than the other. This could enable improvements in product search on pages that feature results from multiple distinct pages, e.g., price-comparison sites, by enabling customers to broaden or focus their search more efficiently. It could also help in the task of catalog integration, as discussed in Meusel et al. [22] and Sabou et al. [34]. Instead of putting all products under the closest class in a target taxonomy, the need for an additional layer may become apparent and, therefore, improve the overall result. We also review and categorize relevant literature with a special focus on e-commerce taxonomy matching. Our main contributions are:

- an overview of relevant taxonomy matching literature,
- the creation of a realistic gold standard for semantic taxonomy matching in the e-commerce domain, and
- an evaluation of product taxonomy matching algorithms on a real-world dataset.

The rest of this Thesis is structured as follows. Chapter 2 introduces the fundamentals of this Thesis. Hence, the Semantic Web and taxonomy matching are defined. It also definitions that we will use throughout this Thesis. We review relevant contributions from current research in Chapter 3. There, we present contributions in the field of the Semantic Web and different taxonomy matching approaches. In the fourth Chapter, the creation of our gold standard is outlined. We describe the methods that are evaluated on the gold standard in Chapter 5 and evaluate the results in Chapter 6. Chapter 7 includes an analysis of the errors that individual methods make in the prediction. Finally, Chapter 8 summarizes our results and gives an outlook into possible future work.

# Chapter 2

# Relevant Concepts

This Section summarizes the relevant concepts which build the foundation of this Thesis. These include the Semantic Web with annotated and, therefore, machine-readable pages. We use the data given in those annotations to create a training data set and a foundation for the product taxonomy matching gold standard. In the second part of this Chapter, we explain ontologies and taxonomies and how they can be matched.

## 2.1 Semantic Web

### 2.1.1 Inception of the Semantic Web

At the beginning of this century, webpages were optimized for human readers. They contained a lot of text and some images. This unstructured information makes it hard for machines to understand the results and to act on them. Berners-Lee et al. [9] propose the Semantic Web, an extension to the current web, that is using annotations that make human-readable information accessible to machines. They propose to use the eXtensible Markup Language (XML) and the Resource Description Framework (RDF) to allow content-creators to make any part of their page machine-readable or to link it to any other content on the internet. This approach is very expressive and flexible, but has the limitation that the content consumer must know what the annotations are about and what they mean.

Berners-Lee et al. propose the RDF syntax based on Unique Resource Identifier (URI) that can be used as primary keys and combined with predicates. Assuming there is a database of persons (pe:) and a database of papers (pa:) in which a person can be an author of a paper (authorOf), we can state that ”pe:Berners-Lee authorOf pa:The-Semantic-Web”.

Usually, URIs point to web pages, e.g., `https://en.wikipedia.org/wiki/Tim\_Berners-Lee`.

### 2.1.2 Realization of the Semantic Web

This Subsection covers how the Semantic Web is implemented today. We focus on formats to annotate pages and shared vocabularies to compare the contents of different websites.

There are two options to add machine-readable content to a webpage. The content creator could either annotate the HyperText Markup Language (HTML) elements directly and keep the annotations close to the human-readable content or add a central property that includes all structured information about the current page. The following example from `target.com` illustrates the usage of a central object that describes the current webpage. The script-tag is placed somewhere inside of the HTML-body and contains all properties that are available for the given product, like the name, brand, and image. The type "application/ld+json" indicates that this is a JavaScript Object Notation (JSON) document containing linked data.

```
<script class type="application/ld+json">
    {
        "@context": "http://schema.org",
        "@type": "Product",
        "name": "Samsung 50\" Smart 4K UHD",
        "brand": "Samsung",
        "image": "<image-url>",
        "sku": "53832607"
    }
</script>
```

In the second example, we translated the above ld+json[1] into microdata annotations. The HTML snippet could then look like this:

```
<div itemtype="http://schema.org/Review">
    <span itemprop="name">Samsung 50\" Smart 4K UHD</span>
    <meta itemprop="brand" content="Samsung" />
    <img itemprop="image" src="<image-url>" />
    <meta itemprop="sku" content="53832607" />
</div>
```

[1] `https://json-ld.org`. Accessed: 01.05.2020

This would show the product name and the product image, while the other two tags would be hidden from the user. Nevertheless, the annotation is still closer to human-readable content and, since this content is reused during the annotation, there can not exist any discrepancies between the machine-readable content and the human-readable content.

One can imagine that both implementations provide a great deal of flexibility and can lead to data silos if there is no agreement between different pages to use a shared vocabulary.

That is where vocabularies like schema.org or the opengraphprotocol.org come into play. They provide a shared vocabulary that can be used across webpages to indicate the same meaning. The examples above already visualized that there are multiple ways to annotate the HTML file. In the following example, we will compare the machine-readable side with the human-readable one. Figure 2.1 shows an


Figure 2.1: schema.org Review example[3]. On the left hand side is the human-readable webpage and on the right hand side is the annotated HTML code.

annotated HTML snippet with schema.org annotations and the corresponding webpage. The surrounding div-tag declares via the itemtype property that it contains a schema.org/Review. The definition of a review contains properties like 'itemReviewed' and 'reviewRating'. In some of the enclosed elements we see the itemprop field, which also maps to the properties specified in the schema.org/Review.

Now, a computer can parse the same code that is presented to a human and can infer that the itemReviewed got a reviewRating of 5 stars. Search engines like Google and DuckDuckGo either use those annotations to enhance the expressiveness of their results or to immediately respond to questions instead of forwarding the customer to another webpage [36].

[3]Review source: `https://www.cdw.com/product/apple-macbook-pro-13-core-i7-2.8ghz-16gb-512gb-touch-bar-space-gray/5578874`. Accessed: 01.05.2020

### 2.1.3 Working with the Semantic Web

The Semantic Web offers an immense potential for data analysis and knowledge generation. However, due to the distributed setup of the internet, there exists no single source to access all data. The Billion Triple Challenge [16] is an attempt at a unified access point, but still contains only a subset of all available data on the web. While Herrera et al. [16] provide only raw triples, the WDC project [21], too, provides triples, but also schema.org class-specific subsets[4]. This enables researchers to focus on specific data, like Hotels or Product Reviews, without the overhead of managing all other data. The WDC project is based on Common Crawl[5] data, a pre-crawled collection of HTML pages. The advantage of the Common Crawl dataset is that a researcher does not have to access millions of web pages himself, but can download the HTML context of those pages from a single source.

In the case that none of the above datasets is sufficient, it is also possible to gather the data directly from the web. Using frameworks like scrapy[6] and LDSpider [15], one can retrieve data from a specific set of webpages or get a broad overview of the web. LDSpider is also used internally for the Billion Triple Challenge.

In this Section, we have seen how semantic annotations make web pages machine-readable and how shared vocabularies make them also machine-understandable. The Semantic Web provides infinite opportunities for data-based projects.

## 2.2 Ontologies and Taxonomies

In this Section, we will introduce the concepts of ontologies and, their special case, taxonomies.

In general, ontologies "can be viewed as a set of assertions that are meant to model a particular domain. Usually, they define a vocabulary used by a particular application" [13, p. 25]. A more intuitive way to describe them would be that an ontology is a format or description that can be used to describe a collection, e.g., database schemata. Ontologies can range from very informal and expressive, like directories, to a very strict and formal language, like XML schemas or entity-relation schemas for databases.

According to Euzenat and Shvaiko [13, p. 34], ontologies usually consist of the following entities:

[4] `http://www.webdatacommons.org/structureddata/2019-12/stats/schema_org_subsets.html`. Accessed: 01.05.2020

[5] `http://commoncrawl.org`. Accessed: 01.05.2020

[6] `https://scrapy.org`. Accessed: 01.05.2020

- "**Classes** or concepts are the main entities of an ontology. These are interpreted as a set of individuals in the domain. [...]
- **Individuals** or objects or instances are interpreted as a particular individual of a domain. [...]
- **Relations** are the ideal notion of a relation independently to what it applies. Relations are interpreted as a subset of the product of the domain. [...]
- **Data types** are particular parts of the domain that specify values. Contrary to individuals, values do not have identities. [...]
- **Data values** are simple values. [...]"

In this Thesis, we will focus on very informal ontologies, namely taxonomies or directories: "A taxonomy is a partially ordered set of taxons (classes) in which one taxon is greater than another only if what the former denotes includes what is denoted by the latter. Directories or classifications are taxonomies that are used by companies for presenting goods on sale, by libraries for storing books, or by individuals to classify files on a personal computer" [13, p. 27].

Let us look at two examples that are illustrative for the cases above. The first one is a directory structure on a computer. Below, we show the layout of our University directory. The root folder, UniMA, contains all directories listed beneath, e.g., Master Thesis and Organizational. Those, in turn, contain other directories, which may contain more files.

```
> tree -L 2 .
UniMA
|- Master\ Thesis
|   |- Application
|   |- Code
|   |- Data
|   |- Literature
|   |- Thesis
|-  Organizational
    |- FSS18
    |- FSS19
    |- HWS17-18
    |- HWS18-19
    |- HWS19-20
    |- Module\ Catalog\ 17.pdf
    |- Module\ Catalog\ 18.pdf
```

Inside each directory, there is a clear hierarchy, i.e., it is partially ordered, but we can not infer any relationships across folders. For example, we can not say how FSS18 relates to Literature.

Another typical example are online shop taxonomies. Figure 2.2 shows a prod-

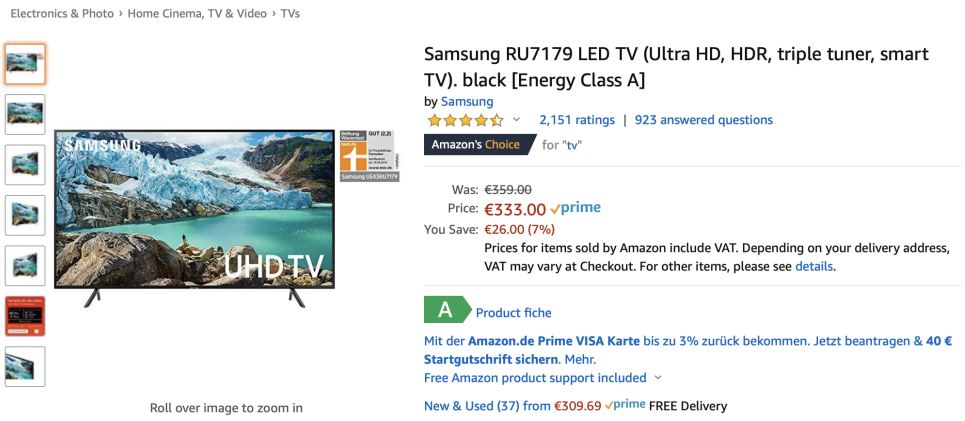


Figure 2.2: Amazon Product Screenshot[7]

uct on amazon.com. In the upper left corner, we see the category of the product, in this case, ”Electronics & Photo > Home, Cinema, TV & Video > TVs”. Imagine a separate taxonomy in the domain of ”Clothing”. While we can say that ”Electronics & Photo” is more general than TVs, the relation between TVs and T-Shirts (”Clothing > Men > Tops, T-Shirts & Shirts > T-Shirts”) is still undefined.

Usually, those taxonomies highly depend on the domain in which they are used. While shops with a big inventory, like amazon.com, may use a very high-level taxonomy for their products, a small, specialized shop may use a completely different set.

For the remainder of this Thesis, we will use the term *class* or *class-label* if we refer to the complete string, e.g., ”Clothing > Men > T-Shirts”, *category* if we mean a specific part of the *class-label*, e.g., ”Clothing”, and *taxonomy* if we mean the tree that is spanned by the *class-labels*.

Returning to the five entities mentioned above, a category or class would be a class in the ontology, while the products would map to the individuals. A special case in a taxonomy is that there exists only one relation type, the *is-a* relation. We can only denote that something is a subclass of something else. Data types and data values are of minor value in a taxonomy, since we are usually restricted to strings, i.e., text-only individuals.

[7]Source: `https://www.amazon.de/dp/B07PGBHP37/`. Accessed: 01.05.2020

Now that we have established the meaning of ontologies and, in particular, taxonomies, we can continue with the topic of ontology and taxonomy matching.

## 2.3 Ontology and Taxonomy Matching

This Section introduces ontology matching with a special focus on taxonomy matching. Since taxonomies are a simpler subset of ontologies, we can not always use general ontology matching methods, because general ontology matching algorithms might rely on additional properties. Whatsoever, there exist also some specialized taxonomy matching methods that use the simpler structure of taxonomies.

Euzenat and Shvaiko define the goals of ontology matching as "finding correspondences between semantically related entities for different ontologies. These correspondences may stand for equivalence as well as other relations, such as consequence, subsumption, or disjointness, between ontology entities." [13, p. viii] In our case, we focus on finding correspondences between product categories. The result of the ontology matching process is called an alignment.

The following example should serve as a motivation for taxonomy matching. Agrawal and Srikant [4] describe a scenario in which a big online shop (A) acquires another online shop (B) and wants to list the combined set of products on their site. Usually, the product hierarchies for A and B may overlap at a high level in their taxonomies, but there will be some differences on the lower level. It may even be possible that 90 percent of the products in B fall into one category of A. One could label all products in B manually to integrate them, but this is usually an expensive and time-consuming process. Another possibility would be to use a classifier trained on A to classify the products in B, taking the description, product name, and similar attributes into account. Both of those approaches discard the taxonomy of B, though. Agrawal and Srikant [4] show that they can improve the classification considerably if the two taxonomies correlate and the taxonomy of B is taken into account.

The problem stated above is called catalog integration and is viewed as a very common problem in the Semantic Web [4], [22], [38]. Often, a central catalog like the GS1 Product Catalog (GPC)[8] is used as the target, but we can also use the example of two distinct shops as demonstrated above.

Other problems that can be solved with ontology matching involve schema integrations for databases or for XML-templates. This Thesis will focus on approaches that infer relations between classes in two taxonomies.

[8] `https://www.gs1.org/standards/gpc`. Accessed: 01.05.2020

# Chapter 3

# Related Work

Multiple attempts have been undertaken to use the Semantic Web as it was envisioned by Berners-Lee et al. [9] as a research dataset. Thereby, the data that was previously hidden in the unstructured HTML files is used as an additional feature generation method for ontology matching and catalog integration [22, 34]. Today, the field of ontology matching is an active research area with contributions utilizing external corpora [14, 28, 2] like WordNet [24], embeddings [37, 31], and other machine learning approaches [12]. Some methods focus explicitly on the task of product taxonomy matching [36, 28, 2].

The first Section in this Chapter introduces contributions that use the Semantic Web as the underlying dataset for their research. They use Semantic Web data for ontology matching or with a focus on products from e-commerce platforms. We introduce taxonomy matching algorithms in the second Section and focus on product taxonomy matching in the third Section. We conclude this Chapter with a categorization of the presented approaches.

## 3.1 Semantic Web

Sabou et al. [34] propose the use of information encoded in the Semantic Web as background knowledge for ontology matching tasks. They name numerous ontology matching approaches that rely on an external ontology as background knowledge, e.g., the work of Aleksovski et al. [5]. In their article Sabou et al. validate the hypothesis that a set of ontologies extracted from the Semantic Web covers a larger universe and contains more knowledge than any single ontology which is currently available or can be manually crafted for a single experiment. Instead of using a single ontology as background knowledge, they use multiple, automatically selected ontologies. Their goal is to use the background ontology with the

best coverage of the given domain. Sabou et al. found that using the Semantic Web leads to results comparable to existing methods, although the semantic relations on the web are often erroneous, especially with regard to subsumption relationships. Overall, they conclude that ontologies extracted from the Semantic Web can serve as background knowledge in order to improve existing mapping methods.

In 2014 Meusel et al. [21] laid the foundation for many experiments using data from the Semantic Web by publishing the WDC Microdata, RDFa and Microformat Dataset Series. Based on the extracts published by the Common Crawl project, they produced a set of over 30 billion RDF quads. The layout of an RDF quad with additional examples is explained in Section 4.1. In addition to the raw extraction results, class-specific subsets are published annually[1]. The latest results based on the 2019 Common Crawl contains more than 900 million quads from more than 218,000 hosts for product-related classes. Multiple authors reuse those results for their own research, e.g., [22, 30, 38].

In a follow-up paper, Meusel et al. [22] reused those results to improve product categorization in e-commerce catalog integration tasks. First, they extracted statistics about properties that occur in the Product/Offer annotations and found that about 86 percent of the Pay-Level Domains (PLD) contain a name, 66 percent an image and/or a description, but only 2 percent of the PLDs annotate their products with a category or breadcrumb. The categories and breadcrumbs form the taxonomy of an individual PLD and contain the class of the given product. Those classes are relevant for the remainder of their paper and also, later, for our work. The goal of Meusel et al. is to use the annotated properties, especially the class-property, to classify products into the GPC's taxonomy. From the annotation data a gold standard was created and used to evaluate product classification methods. During the process, Meusel et al. encoded their product information as vectors with a Bag-of-Words (BoW) and a tf-idf approach and applied multiple classifiers, namely Naive Bayes, Decision Trees, and k-Nearest-Neighbors. They show that it is possible to predict the product class with an 80 percent accuracy in a supervised approach.

Zhang and Paramita [38] build on the dataset and ideas of Meusel et al. [22], but use new classification methods based on deep-learning models and achieve a significant improvement with regard to the F1-score. Instead of encoding the product data as vectors with BoW or tf-idf, they use a GloVe word-embedding model [29] that was pre-trained on the Common Crawl corpus. In their work, they also considered a word2vec word-embedding model [23] pre-trained on a Google News[2] corpus, but expected a higher coverage with the Common Crawl model since the

[1] `http://webdatacommons.org/structureddata/2019-12/stats/schema_org_subsets.html`. Accessed: 01.05.2020

[2] `https://code.google.com/archive/p/word2vec/`. Accessed: 01.05.2020

data is also extracted from the same corpus. Zhang and Paramita state that in most classification papers, the name property is combined with additional metadata as an input for the classifier. Contrary to this approach, they also use the description and the class-label as individual inputs and try combinations of those three features. Here, the class-labels are encoded as a weighted sum of the word-embeddings that make up the class-label. In addition, Zhang and Paramita experimented with different cleaning and normalization steps on the class-label and discovered that they were counterproductive for the product classification task since relevant contextual information was discarded. Overall, they demonstrate that word-embeddings on the class-labels of products are a useful input for classification tasks.

Another use of product data from the Semantic Web is given by Primpeli et al. [32, 30]. They use the class-specific product dataset published by the WDC project to create a large training set and gold standard for product matching on realistic, web-scale data. Afterwards, they apply existing approaches for entity resolution on this dataset and try to replicate the results from other contributions that were tested on smaller datasets. In order to create the training dataset Primpeli et al. pick schema.org properties that point to product identifiers, clean them, and perform a similarity search on the identifiers to find products that have the same underlying identity. They manually validate a subset of 2,200 products of those matches and, in the end, provide three datasets: the complete training dataset, a subset from English domains, and a manually verified gold standard. The complete training dataset consists of 26 million offers and the English subset of 16 million offers. Their baseline experiments on the newly created datasets achieve an F1-score of 0.9 for entity resolution and, therefore, prove the utility of this training data from more than 79,000 e-commerce platforms.

## 3.2 Ontology/Taxonomy Matching

We will focus on general ontology matching with a special focus on taxonomy matching in this Section. In the first part textbooks and surveys are introduced, which may serve as a reference for a more profound, general introduction into the topic of ontology matching. Next, catalog integration papers are introduced. Their relevance for this Thesis stems from the fact that properties of ontologies are used for a classification task. Towards the end of this Section, we will take a closer look at taxonomy matching approaches.

### 3.2.1 Overviews

The Ontology Matching textbook by Euzenat and Shvaiko [13] introduces ontology matching and includes a multitude of applications and models. If one is interested in the topic of ontology matching beyond the task of taxonomy matching, this should serve as a good starting point.

Euzenat and Shvaiko also provide a classification of ontology matching approaches shown in Figure 3.1. They define two high-level approaches to separate

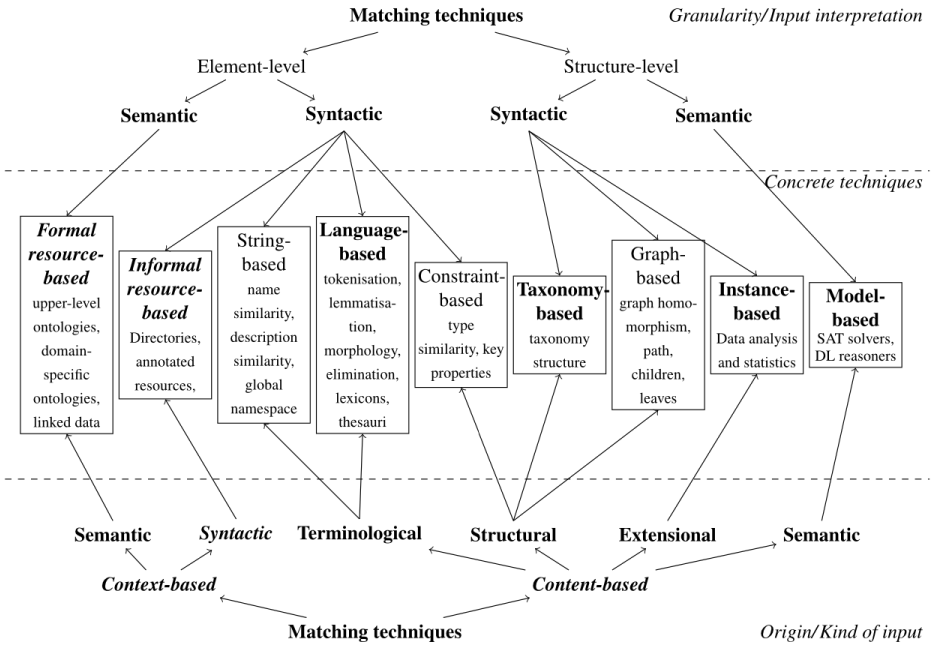


Figure 3.1: Ontology Matching Classification [13]

the specific techniques. At the top, we see the classification based on granularity and at the bottom the classification based on origin. Note that the techniques placed in the middle are shared among them. The classification based on granularity distinguishes between element-level matchers and structure-level matchers. The former compares only two entities at a time together with their properties, while the latter takes the whole ontology into account to predict the most probable alignment. The origin-based classification distinguishes based on the source of information that is used in the classifier. Content-based methods focus only on the content of the given ontologies, while context-based matchers allow external

knowledge resources, e.g., external ontologies from the Semantic Web as in [34].

Angerman and Ramzan published a textbook focusing exclusively on taxonomy matching approaches [6]. First, they introduce the problem of taxonomy matching by contrasting it to general ontology matching and motivate the work by introducing use cases for taxonomy matching in the real world. They also reuse ideas from Euzenat and Shvaiko [13], especially the categorization based on Granularity and the different sources of heterogeneity in taxonomies, namely terminological, conceptual, syntactical, and semiotic. They define those concepts as follows [6, p.18]:

- "*Terminological Heterogeneity* appears when the concepts of two taxonomies are expressed using different languages [, e.g., English and German].
- *Conceptual Heterogeneity* arises if two taxonomies use different relationships to describe an identical domain. [...].
- *Syntactical Heterogeneity* occurs when for the storage of the taxonomies different data languages/models are used. [...].
- *Semiotic Heterogeneity* emerges when persons misinterpret concepts, respectively the relationships inside the taxonomies along with the used labels for concepts. [...]."

In the second part of their book, they survey state-of-the-art taxonomy matching algorithms from recent OAEI campaigns (2011-2015). They break each contribution down into its individual components and show how each component tackles one of the four sources of heterogeneity defined above. For terminological heterogeneity, translators are used, while conceptual heterogeneity may be covered by external knowledge bases like WordNet or Wikipedia. For our work, especially models that cover conceptual heterogeneity are relevant since we only use English product classes, represent them in a common file format, and no human are involved in the matching.

In addition to a survey of taxonomy matching algorithms, Angerman and Ramzan also describe the individual tracks of the OAEI. The different tracks are discussed, namely the Ontology Track, the Multifarm Track, the Directories and Thesauri Track as well as the Interactive Matching Track. The Multifarm Track focuses on multi-language matching and the Interactive Matching Track on evaluating the factor of human inputs during the matching. The Ontology Track can be characterized as the most general task with 16 taxonomies while the Directories and Thesauri Track covers a matching task for libraries. Overall, the book provides a broad introduction into taxonomy matching and serves as a comprehensive overview of the field.

Avesani et al. [8] notice that current taxonomy matching papers do not use a common foundation to make their contributions comparable. They create a dataset which aims at representing real-world problems and evaluate existing taxonomy matching methods. In conclusion, they state that most evaluations do not hold up in the real-world using representative data.

### 3.2.2 Catalog Integration

The task of catalog integration is closely related to our task of taxonomy matching. For a given class-label, the aim is to identify the the closest class in a target taxonomy. Most methods rely on some similarity measure to find matching candidates that can be used to make predictions which are required for taxonomy matching.

One of the earliest contributions in this area, presented by Agrawal and Srikant [4], was published around the same time as the first proposal of the Semantic Web by Berners-Lee et al. [9]. The problem of integrating catalogs is, therefore, at least as old, if not older, as the problem of making the web accessible for computers. Agrawal and Srikant propose a use case around a corporate merger where the parts manufactured by the acquired company have to be integrated into the current catalog. They operate under the assumption that the vocabularies in both catalogs are similar and that they use a similar model to categorize the entities. Future algorithms use external ontologies or other corpora to enable alignments in cases where this assumption does not hold, but those were not available at that time. In their experiments, Agrawal and Srikant use a Naive Bayes classifier to predict the target class given a product-label as a baseline and compare it with a Naive Bayes classifier that takes product- and class-labels into account. They report that the accuracy increased significantly for most datasets and the performance of the enhanced classifier has never been worse than the baseline. This early contribution shows that the given class on a product is an important feature for the prediction of the most similar class in another taxonomy.

Papadimitriou et al. [27] use the same motivating example as Agrawal and Srikant [4], but focus on the structure of the complete source taxonomy to enhance their matcher. They use a text-based classifier for their matching and add the constraint that classes that are close in the source taxonomy should also be close in the target taxonomy. This, again, operates under the assumption that the taxonomies use at least a similar model to categorize instances. Yet, the requirement of a similar vocabulary is somehow lifted, since classes that should be close, but use varying terms for the same concept, are moved closer together. Essentially, they do not simply add class-labels as flat strings as additional inputs, but rather walk up the taxonomic structure in case no obvious match can be identified. Matching on a higher taxonomic level also has the advantage that the search space of possible

matches is pruned. This effect allows the algorithm to scale to very large input taxonomies. The first step of Papadimitriou et al.'s algorithm is a text-based classifier like Naive Bayes that takes the class-labels as inputs. Their new contribution is the addition of a second step that consists of an optimization problem that trades off the previous text-based predictions with a cost for separating classes that are close in the source taxonomy. This shows that not only the terms in the source class are relevant for the matching, but also the hierarchical structure of the source taxonomy as a whole.

Meusel et al. [22] also focus on catalog integration. Instead of using a corporate merger case, they use e-commerce platforms and the Semantic Web as a motivating example. Their contribution with regard to the Semantic Web was already discussed in Section 3.1. Similar to Agrawal and Srikant [4] they utilize standard classification algorithms like Naive Bayes and k-Nearest-Neighbor classifiers as their baseline and encode the properties of each product with BoW and tf-idf methods. Meusel et al. propose two new approaches that they benchmark against their baseline. First, they encode the target taxonomy and each input product as a vector and compute all distances between source- and target-classes using Cosine-similarity and the Jaccard coefficient. The closest match with a non-zero similarity is then taken as the prediction. In a second approach, they follow Papadimitriou et al. [27] in formulating a global optimization problem to compute the optimal matching. In addition to closeness in the vector space between the source- and the target-class, they require that products that are close should be in the same class. The major difference between Meusel et al.'s approach and Papadimitriou et al.'s approach is that the closeness between products is computed based on their vector encoding instead of the class-labels. All in all, Meusel et al. show that supervised methods outperform distantly supervised approaches, but both approaches are promising in classifying products into categories. They raise concerns about the scalability of the global optimization approach, though.

## 3.3 Semantic/Product Taxonomy Matching

Giunchiglia et al. [14] present S-Match, an approach for semantic schema matching. They use WordNet to derive the intended meaning of a class-label and combine this with multiple other matchers to assign a semantic label for the relation between two classes. This is the biggest differentiator to other contributions in the field. Furthermore, instead of solely inferring equality or finding the closest match between two taxonomies, they also predict if one class is more general than the other. A more detailed description of their algorithm is given in Section 5.2.1. At this point, we will just lay out the main ideas of their paper. In a first step,

S-Match computes the senses of each class-label using WordNet. Then, it pairs all labels in both taxonomies and computes the relation between them, which results in a propositional validity problem. They test each possible relation, i.e., *equal*, *contains*, *contained-in*, *disjoint*, together with their propositional statement and try to solve it. If the problem is solved, the tested relation is accepted, and in case it resolves to a conflict, it is rejected. In the latter case, they use eleven additional matching tools to make a prediction. Five of them are string-based and the other six use the WordNet glossaries to predict subsumption relationships. To the best of our knowledge, the solution by Giunchiglia et al. is the only one that is also focused on the prediction of a semantic label for all class-label pairs between two taxonomies.

Park and Kim [28] were the first to approach the taxonomy matching problem in the domain of e-commerce with WordNet. Their results are commonly used as a baseline for other contributions that focus on e-commerce. They focus on product search, namely returning a set of similar products in a target taxonomy given a product in a source taxonomy. This problem is then narrowed down to finding close classes in a taxonomy given a class-label from another taxonomy. Standard ontology methods have a high focus on precision or accuracy to avoid the introduction of mismatches. According to Park and Kim, in product search, the focus should be on a higher recall since it is preferable to present a somewhat related product to a client instead of returning no products at all. It may also come at a high cost for the retailer if a suitable product is not returned in a search, because the set of possible matches was narrowed down too quickly. Hence, their goal is to increase the recall while making minimal trade-offs with regard to precision. Similar to Giunchiglia et al. [14] they employ WordNet to figure out synonyms given the class-label. To avoid the inclusion of homonyms, i.e., words with the same spelling, but different meaning, they filter the synonyms by their senses, given the context of the current product. The context is given by the higher layers in the source taxonomy of the given class-label. Park and Kim empirically prove that their algorithms improves on recall for multiple datasets in comparison to the PROMPT algorithm by Noy and Musen [26].

Nederstigt et al. [25] propose SCHEMA as an extension of the ideas from Park and Kim [28] and describe their results in a series of papers [25, 2, 3]. The latest contribution [25] is the most exhaustive presentation with regard to the algorithm description and evaluation. The authors identify some shortcomings in the method of Park and Kim and provide improvements to achieve better overall results. The first advancement is with regard to composite categories. These frequently occur in product taxonomies and have a form like ”TVs, Notebooks & Monitors”. The method established by Park and Kim would use the complete label and, since there is no match in WordNet, would be unable to disambiguate the sense of the label

above. Nederstigt et al. propose the usage of a split term set, which treats all of those words individually and, therefore, increases the likelihood of a match in WordNet. Another shortcoming is the lack of context for short paths in the method by Park and Kim. This makes it hard to identify the true sense. SCHEMA does not only take the parent, but also child- and sibling-label into account to determine the context. In their experiments, Nederstigt et al. show that the recall improves compared to the method by Park and Kim, but the accuracy is lower. Since the authors of both papers consider the recall more important in the product domain, this result can be considered as an improvement on the previous method.

Vandic et al. [36] focus on the utility of the Semantic Web in the product search domain. They claim that current keyword-based index lookups are insufficient, because users are not able to search for specific product features. Therefore, shoppers must rely on the price as the sole indicator for a buying decision. As an alternative, they propose and implement xploreproducts.com, a product search engine that enables the user to query for additional attributes. They identify two challenges. First, the identification of identical products and, second, the mapping of distinct product categories. Since we rely on a simple join on the product identifier in this Thesis, we will not describe Vandic et al.'s contribution to the identity resolution task. Regarding the second challenge, their idea for matching the product category is based on string-similarity methods, namely Levenshtein- and Cosine-similarity, however, they add a hierarchical component. The categories at the end of the class-label are weighted higher than the ones further up the hierarchy. Given two classes $c_1 = (l_4, l_3, l_2, l_1)$ and $c_2 = (k_3, k_2, k_1)$, where the higher indexed categories are closer to the root, the similarities $(l_1, k_1)$, $(l_2, k_2)$, and $(l_3, k_3)$ are computed and aggregated by a weighted sum. If one class is deeper than the other, in this case $c_1$, all unmatched categories $(l_4)$ are ignored. In their experiments, this hierarchical approach outperforms the algorithm by Park and Kim [28].

## 3.4 Categorization of Taxonomy Matching Approaches

The classification trees presented by Euzenat and Shvaiko [13] that are shown in Figure 3.1 display two ways to classify ontology matching approaches. Since taxonomies are a subset of ontologies with very few restrictions, not all methods and paths of those categorization models apply to them. Thus, we present a simplified way to cluster taxonomy matching algorithms in this Section and use this classification method to categorize the contributions that have been cited earlier in this Chapter. In order to provide an overview of all taxonomy matching algorithms used throughout this Thesis, we will include the matching algorithms that will be introduced in Chapter 5. Our categorization system has a matrix form, where each

algorithm can be in one of the following four categories: static-internal, static-external, learning-internal, or learning-external. In the following, we explain those labels and assign the taxonomy matching methods to a category in Table 3.1.

The first group of algorithms is string-based. They use the class-labels as the input and assign a label to each pair based on string-similarity measures. This may include Levenshtein- or N-Gram-similarity on the full strings or some combination of those two. The methods may also use the hierarchical structure to build a weighted sum of similarities of the sub-categories. Using the Granularity interpretation of Euzenat and Shvaiko [13], this would correspond to a combination of "Element Level $>$ Syntactic $>$ String-Based" and "Structure-Level $>$ Syntactic $>$ Taxonomy-Based". We will categorize them as internal-static since they do not rely on external knowledge bases and, apart from some hyperparameter tuning, they do not benefit from training examples and, therefore, return static predictions that do not change over time.

The second group uses external knowledge bases to bridge semantic gaps in the taxonomies, which exist as a result of the usage of different vocabularies during the construction. The most common variant is the use of WordNet [24] to detect synonyms. Methods in this group often combine the results from the external knowledge base with the string-similarity measures in the preceding group, but are not limited to them. We call this group external-static since external knowledge bases are used during the prediction, but the effect of training examples is, again, limited to hyperparameter optimization. In the classification system shown in Figure 3.1 they would fall into the "Informal Resource-Based" and "Language-Based" group and may use methods from the "String-Based" approaches.

The last group of algorithms that we have identified and employ in this paper are machine learning algorithms. They use an encoding model, e.g., BoW or word2vec, to transform the class-labels into vectors and learn an optimal set of parameters from training data. We label those models as internal-learning. While they may use external data to train the embedding model, they do not use this external knowledge during the prediction phase. We can actually treat the encoding method as a black-box that simply produces a vector given a class-label. We consider them learning instead of static since they change their prediction based on the training examples they are presented with. The machine learning methods may also use the taxonomic structure of the class-labels during the translation into features vectors. This third group is not part of the categorization introduced by Euzenat and Shvaiko [13]. This may result from the recent rise in the capabilities of machine learning compared to the models that were available in 2011.

In conclusion, we identified three groups in the set of taxonomy matching approaches. Looking at the matrix structure of our classification approach, there may also be a group of external-learning algorithms, i.e., approaches that combine ma-

chine learning with external knowledge bases at runtime, but we are not aware of any contribution that falls into this domain. In Table 3.1, we categorize all contributions that are introduced in this Chapter and the methods used throughout this Thesis.

| | Internal | External |
|---|---|---|
| Learning | • Meusel et al. [22]<br>• Zhang and Paramita [38]<br>• Agrawal and Srikant [4]<br>• Papadimitriou et al. [27]<br>• Open Source Machine Learning Classifiers | |
| Static | • Levenshtein-Similarity<br>• N-Gram-similarity<br>• Vandic et al. [36] | • Park and Kim [28]<br>• SCHEMA [2]<br>• Sabou et al. [34]<br>• S-Match [14] |

Table 3.1: Classification of Taxonomy Matching Approaches.

In this Chapter we saw that there is a multitude of approaches for taxonomy matching. While we have tried to find a good distinction between the methods, there is still some overlap. Many contributions extend existing ideas, instead of approaching the problem from a completely new perspective. An example is the usage of WordNet in combination with string-similarity. An interesting extension would be the usage of external knowledge bases in combination with machine learning models.

# Chapter 4

# Training Data Creation

In this Chapter, we describe how we created our training dataset and the gold standard that we use in the upcoming Chapters to train and evaluate different taxonomy matching approaches. First, we will look at different approaches on how to use the Semantic Web to generate an extensive collection of product data and use annotated identifiers to match products across different e-commerce platforms. Next, we describe how those instance pairs were used to create a training dataset of two classes that are part of different e-commerce taxonomies and a corresponding label that indicates if those classes are equal to each other, if one class contains the other or if they are disjoint. Furthermore, we introduce a method to provide edge case examples that are close to our positive labels (*equal*, *contains*, *contained-in*), but are actually *disjoint*. This Chapter is concluded by an overview of the inherent statistics of our gold standard. Here, we describe the depth of the covered taxonomies and visualize the distribution of labels.

## 4.1 Retrieving Product Information from the Semantic Web

This Section outlines an approach to retrieve product specific information from the Semantic Web. The first approach is based on the work of the WDC project and a second approach crawls large retailers directly and extracts the data we are interested in.

### 4.1.1 Web Data Commons Product Dataset and Gold Standard

As outlined in Section 2.1 the Semantic Web provides machine-readable information in annotated HTML-code. The WDC project provides class-specific datasets that are based on schema.org data and extracted from the Common Crawl cor-

pus [21]. In addition, Petrovski et al. [30] created a gold standard that contains product matches based on shared identifiers and other indicators.

For this Thesis, the product class-specific dataset is used, which contains about 300 million products stored as quads. The file layout is "<subject> <predicate> <object> <source> .", e.g., "_:nodeabcd <http://schema.org/Product/name> 'Soccer Ball' <http://123stores.com/shopby/?brand=1214> .".

There are multiple ways to describe the class of a product in the schema.org notation. The most used properties are "Category", "Breadcrumb", and the use of a "BreadCrumbList" that links to the individual levels of the class hierarchy. As a first step, the quads were parsed and a set of <nodeId>, <breadcrumb> tuples created that were joined with the WDC gold standard for product matching [30]. This resulted in a dataset that contains the productId, the source URI, and the class of the product. Using the clusters of the WDC gold standard it was also possible to assign a high-level category to each product. Next, a self-join with the dataset is performed to create product pairs that can later be labelled.

Unfortunately, the resulting dataset is to small for the remaining analysis. Most PLDs contain only a few products. Hence, we favor the approach introduced in the next Section in which we prefer a deep crawl of individual pages over a broad crawl across the web.

### 4.1.2 Crawling Product Data from the Semantic Web

Another way to get product data from the web is a self-written crawler that targets e-commerce sites directly. The Common Crawl corpus has a broad coverage of different domains, but the number of pages per domain is comparatively small. A self-written crawler has the advantage that all sub-pages of a domain can be crawled and only the content of interest is extracted.

Scrapy is an "open source [...] framework for extracting the data you need from websites. In a fast, simple, yet extensible way" [1]. The data of the following PLDs was crawled as a foundation for our experiments: amazon.com, walmart.com, ebay.com, bestbuy.com, newegg.com, and overstock.com.

Since the number of PLDs that were crawled is small, there was no need for a generic schema.org extraction framework like LDSpider [15]. Instead, we use XPath to parse the Document Object Model (DOM)-tree of an HTML-file to get the relevant properties per page. Those were the identifiers, the class, and the source-URI of the product. Again, a self-join over the identifiers resulted in product pairs.

This Section introduced two approaches to generate a set of product-pairs that have the same underlying entity, identified by a set of product identifiers, and distinct categories that were assigned by the individual online-shops.

## 4.2 Assigning Semantic Labels to Class-Pairs

In this Section, a method is introduced to transform the product-pairs into class-pairs with a semantic label. A sample from the resulting training dataset is provided in Table 4.1. Table 4.1 consists of five columns. "pld_l" and "pld_r" indicate the

| pld_l | class_l | pld_r | class_r | label |
|---|---|---|---|---|
| amazon | [...] Women > Smartwatches | bestbuy | [...] All Smart-watches | contained-in |
| amazon | [...] Binocu-lar, Camera & Camcorder Straps | bestbuy | [...] Camera Straps | equal |

Table 4.1: Training Dataset Sample.

source of the left and right class labels. "class_l" and "class_r" are the hierarchical class-labels, where $>$ indicates the different categories. For the visualisation, we removed the first parts of the class-labels. Finally, "label" is the label that our approach assigned. The remainder of this Section will cover the creation of this training dataset based on the product-pairs.

Following Euzenat and Shvaiko [13, p. 113] "[the] easiest way to compare classes when they share instances is to test the intersection of their instance set A and B and to consider that these classes are very similar when $A \cap B = A = B$, more general when $A \cap B = B$ or $A \cap B = A$." It is, therefore, possible to infer the label with the set relation. $A \cap B = A = B$ indicates equality, $A \cap B = B$ and $A \cap B = A$ indicate that A contains B and A is contained in B, respectively, and, finally, $A \cap B = \emptyset$ indicates disjointness. The term *overlap* is used in case none of the above conditions hold [18]. An overlap occurs when the classes have some products in common, but both classes contain products that also have other classes in the other taxonomy.

As the next step, the labels for each class-label pair were computed. Algorithm 1 provides pseudo-code for the labelling of the training dataset.

For every PLD-pair, every possible class-label pair is computed. Then, for each class-label pair, the set of product-pairs with a match of the left class and the set of product-pairs with a match of the right class is computed. Those are called left and right in the algorithm. Finally, the set of product-pairs with a match of the left and the right class are computed and the set logic from above is applied to assign the labels.

Using this setup, each possible class-pair per PLD-pair gets a label that depends on the number of shared instances.

**Algorithm 1** Class-Label Pair Labelling

TRAINING DATASET($tuples$, $pld\_pairs$)

```
1: loop
2:   for pair ← pld_pairs do
3:     for class_l ← tuples.pld[pair[0]] do
4:       for class_r ← tuples.pld[pair[1]] do
5:         left ← tuples.class[class_l]
6:         right ← tuples.class[class_r]
7:         intersection ← tuples.class[class_l&class_r]
8:         if intersection = ∅ then
9:           return disjoint
10:        end if
11:        if intersection = left = right then
12:          return equal
13:        end if
14:        if intersection = left then
15:          return contained − in
16:        end if
17:        if intersection = right then
18:          return contains
19:        end if
20:        return overlap
21:      end for
22:    end for
23:  end for
24: end loop
```

In this Section a setup was introduced to transform the product-pairs from Section 4.1 into a labelled training dataset that consists of class-pairs with a label indicating that two classes are equal or disjoint or if one class is more general than the other.

## 4.3 Generating Corner-Cases

During the creation of the training dataset, we pair all different class labels between two taxonomies and assign a label to each of those pairs stating if they are equal or if one contains the other or if they are disjoint. We plan to use this labelled data for the training and evaluation of taxonomy matching methods.

This results in a huge amount of negative data samples since it will also include obvious mismatches like "Clothing > Pants > Mens Jeans" and "Electronics > Camera & Photo > Digital Cameras". Hence, it is likely that the algorithm will have an easy time to label those as *disjoint* examples and may not be able to identify the most discriminating features to classify positive classes.

A common approach to tackle this problem is the inclusion of edge-cases in the training data. Those are samples that are very close to the positively labelled instance, but should receive a negative prediction from the matcher. Therefore, we generate artificial edge-cases for the *disjoint* label and add those to our training data. Since we are interested in samples that are close, but *disjoint*, we use the taxonomic structure of the instances that are already part of our training dataset. The tree structure below illustrates a fictitious taxonomy from an electronic e-commerce platform.

```
Electronics
|- Cameras
|   |- Digital Cameras
|   |- Polaroid Cameras
|- TVs
```

Instead of searching across two taxonomies for edge-cases, we simply add leaves in the taxonomy tree that share a parent and label them as *disjoint*. In the example above, the class-labels for "Electronics > Cameras > Digital Cameras" and for "Electronics > Cameras > Polaroid Cameras" are close, but we can be certain that they are *disjoint*.

We search through the complete training dataset that we derived from the Semantic Web for class-label pairs that share their immediate parent and add those as additional edge-cases to the training dataset.

## 4.4 Class Label Balancing

Including the edge-cases, as described in Section 4.3, the training set consists of 1,057 positive examples and 1,030,497 negative examples. Out of those 1,030,497 negative examples, 25,674 were artificially generated. This results in a ratio of 1 positive example for 947 negative examples.

Many machine learning algorithms struggle if the number of class-labels are significantly imbalanced [10]. On the one hand, the training time increases significantly, since more examples have to be processed, but most of the negative examples may have a low information content, as the disjoint example given at the beginning of the previous Section illustrates. On the other hand, there may not be enough positive examples to learn relevant features for the classification. We will introduce the two approaches we use to mitigate those problems in this Section.

First, we use under-sampling approaches to reduce the number of negative samples [10]. Out of all actual *disjoint* pairs (excluding artificially generated edge-cases), we retain 0.5 percent. From the generated *disjoint* pairs, we retain 10 percent. We provide a fixed random-seed for the sampling to make sure that the experiment results are deterministic across multiple runs and are not influenced by different samples. Overall, this reduces the number of *disjoint* pairs in our training set to 5784 and results in a ratio of 1 positive example for about 6 negative examples.

We use the resulting, smaller dataset for the training and evaluation of all text-based taxonomy matching models. Since the training data is used to find an optimal decision threshold, the value added by additional positive samples (over-sampling) is negligible.

The machine learning algorithms also profit from additional positive examples. In addition to the under-sampling of the majority class, we add over-sampling of the minority classes. Chawla et al. [10] introduce the Synthetic Minority Over-sampling TEchnique (SMOTE) and show that the ”combination of SMOTE and under-sampling performs better than plain under-sampling”. They also prove that synthetic minority examples generated by SMOTE improve the classification performance of multiple machine learning models compared to over-sampling by replicating positive instances.

To generate new samples, SMOTE takes multiple instances of the minority class in a nearest-neighbors fashion and interpolates a new example at a random point between these two. This creates artificial positive examples to balance the class-distribution during the training of machine learning models.

In this Section, we introduced two approaches for dealing with our imbalanced training data. We used random under-sampling during the training of text-based classifiers and, in addition, over-sampling with SMOTE for the machine learning

models. We expect that under-sampling of *disjoint* class-label pairs reduces the overall training time without sacrificing performance. The over-sampling based on SMOTE should increase the performance of the machine learning models since they will have more positive training data.

## 4.5 Gold Standard Creation and Dataset Statistics

In addition to the automatically labelled product class-label pairs, we manually checked 739 positive product class-label pairs and updated their label if necessary. Since two classes are only labelled as *disjoint* by Algorithm 1 if they have no instances in common, and we also create artificial corner-cases that are guaranteed to be disjoint, we have high confidence that *disjoint* labels are assigned correctly. Yet, we found that the instance-based automatic annotation assigns wrong labels if one e-commerce platform has many products in a certain class and the other only a few. In those cases, an *equal* class-label pair may still receive a *contains* or *contained-in* label. To avoid those pitfalls and get a high-quality dataset for our evaluation, we created a gold standard from the training dataset.

In the remainder of this Section, we present the properties of the gold standard that we use for the evaluation of the taxonomy matching algorithm that we describe in the upcoming Chapter. Overall it contains 241 *equal*, 257 *contains*, 241 *contained-in*, and 5045 *disjoint* class-label pairs.

Figure 4.1 illustrates the distribution of the positive labels between different PLD-pairs. We can see that the PLD-pairs "(amazon, ebay)" and "(bestbuy, ebay)" have comparatively few *contains* class-label pairs. This indicates that amazon and bestbuy use higher-level classes, compared to fine-granular class-labels for ebay. Apart from those two PLD-pairs, every positive label has at least ten instances for each PLD-pair.

We use those PLD-pairs for nested cross-validation during our experiments. To use all gold standard examples during the training and the evaluation, we hold out one of the seven PLD-pairs for each training session and use it to evaluate the models afterwards. We aggregate those intermediate results to arrive at the numbers we present in Chapter 6. In each iteration, we use the remaining six PLD-pairs to train our models and perform $k$-fold cross-validation to optimize hyperparameters.

While the GPC standard uses only three hierarchy-levels for their product class-labels, our dataset indicates that real-world e-commerce platforms use deeper taxonomies. Figure 4.2 visualizes the number of categories per class-label. We see that a class-label consists of 4 to 5 categories on average. This indicates that previous contributions in the taxonomy matching domain that use the GPC may make the simplifying assumption that a class-label consists of at most three categories.

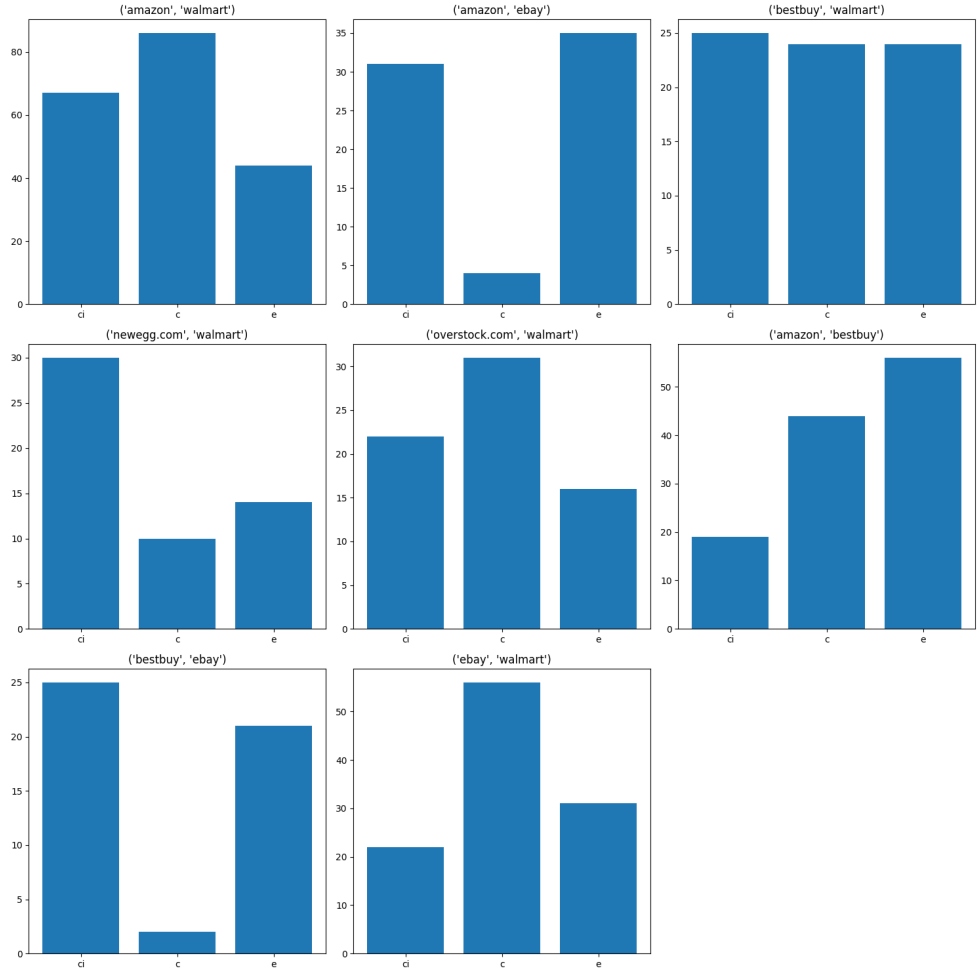


Figure 4.1: Label Distribution across PLD-Pairs.

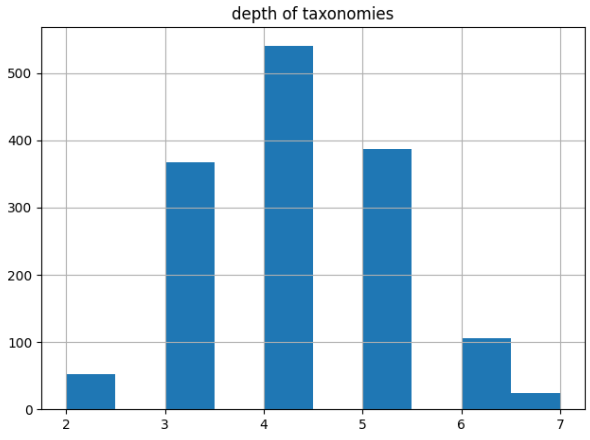


Figure 4.2: Depth Distribution of Class-Labels.

Therefore, their results may not hold up on real-world taxonomies from large e-commerce platforms.

# Chapter 5

# Product Taxonomy Matching Methods

This Chapter focuses on methods to produce matches in taxonomies and we will describe concrete implementations. The focus lies on three subcategories of algorithms. We start with very simple ones that we can use as baselines in our experiments, then move on to unsupervised algorithms that use external knowledge, like WordNet[1], and, finally, describe supervised learning algorithms. They are trained on our dataset and then make predictions on previously unseen pairs.

The goal of each method is to assign a label to a pair of two classes that indicates if they are equal to each other, if one contains the other, or if they are disjoint, i.e., describing a disjunct set of entities.

In our implementation, all methods are implemented as normalized similarity measures instead of distances, i.e., the results are in a range between 0 and 1 and very similar class-labels are closer to 1. A distance is 0 when the two inputs match and increases when their similarity decreases.

## 5.1 Baseline Methods

In this Section, we will present simple models that mostly work on the raw category string. We start with the Levenshtein- or Edit-similarity, move on to the N-Gram-similarity and conclude with a Path Distance (PD) measure that extends the Levenshtein- and N-Gram-similarity to hierarchies.

[1] `https://wordnet.princeton.edu`. Accessed: 01.05.2020

### 5.1.1 Levenshtein- or Edit-Similarity

The Levenshtein- or Edit-distance is a simple way to measure the distance between two strings. It supports a specific set of mutations, and the distance is the number of mutations to get from one input to another. The allowed operations are substitution, addition, and deletion of individual characters [20]. Let us compute the distance between "rose" and "close". First, we can replace the "r" with an "l" to arrive at "lose" and then add a "c" to the left. The edit distance between "rose" and "close" is, therefore, two.

Since the edit distance is highly dependent on the length of the word, we use a normalized distance where we subtract the number of edits divided by the longer string from one. This subtraction result converts the distance measure into a similarity.

$$Levenshtein_{sim}(s_1, s_2) = 1 - \frac{Levenshtein_{dist}(s_1, s_2)}{\max(\text{len}(s_1), \text{len}(s_2))}$$

We will use the term Levenshtein-similarity from now on to describe this method. The Levenshtein-similarity returns a result between 0 and 1, with 0 meaning that there is no relation at all and 1 meaning that the two strings are the same.

We test multiple thresholds of similarity values on a training dataset and select the best-performing threshold for the final prediction. At this point, we need to integrate another trick since we not only want to predict similarity, but also if one category contains the other or vice versa. We will apply this trick to all coming measures that only provide a similarity and point to this explanation for reference.

To predict the label, we compute multiple distances. Obviously, we predict one label between the two unaltered class-labels, but, additionally, we remove the lowest category in the left and right class-label and compare it with the unaltered right and left class-label. If such a redacted label is closer to the other class-label, we predict containment instead of equality. Of those three scores, we take the maximum and, if it exceeds the similarity threshold, we use it to label the category pair.

The following example will illustrate this. Assume that we want to compare "Clothing, Shoes & Jewelry > Women > Watches > Smartwatches" with "Wearable Technology > Smartwatches & Accessories > All Smartwatches". We compute the following edit distances:

- "Clothing, Shoes & Jewelry > Women > Watches > Smartwatches" with "Wearable Technology > Smartwatches & Accessories > All Smartwatches"
- "Clothing, Shoes & Jewelry > Women > Watches" with "Wearable Technology > Smartwatches & Accessories > All Smartwatches"

- ”Clothing, Shoes & Jewelry > Women > Watches > Smartwatches” with ”Wearable Technology > Smartwatches & Accessories”

This results in the following tuple of predictions: (0.34, 0.28, 0.3). In the example above, the Levenshtein-similarity would assign the label *equal* if the threshold is exceeded.

### 5.1.2 N-Gram-Similarity

This Subsection describes the N-Gram-similarity. Similar to the Levenshtein-similarity, it only compares the string without any background knowledge about the hierarchical structure of the input categories.

Euzenat and Shvaiko describe the N-Gram-similarity as follows: ”The n-gram similarity is often used in comparing strings. It computes the number of common n-grams, i.e.,, strings of n characters, between them. For instance, trigrams for the string ”article” are ”art”, ”rti”, ”tic”, ”icl”, ”cle”.” [13, p. 90] The formula for the normalized N-Gram-similarity is given as:

$$\bar{\sigma}(s,t) = \frac{|ngram(s,n) \cap ngram(t,n)|}{\min(|s|,|t|) - n + 1}$$

Again, we test multiple thresholds for the similarity and, since we have another parameter n, also multiple values for n and select the best parameter combination on the training set.

To predict not only equality, but also *contains* and *contained-in*, we follow the same procedure as already described in Subsection 5.1.1.

We conclude the N-Gram description by giving an example of how the similarity works. We compare ”Clothing, Shoes & Jewelry > Women > Watches > Smartwatches” with ”Wearable Technology > Smartwatches & Accessories > All Smartwatches” with $n = 3$. Splitting the first string into trigrams results in the following set:

```
{('e', 's', ' '), ('J', 'e', 'w'), ('o', 'm', 'e'), ('h', 'e', 's'),
('a', 'r', 't'), ('c', 'h', 'e'), ('m', 'a', 'r'), (' ', 'S', 'h'),
('C', 'l', 'o'), ('o', 'e', 's'), (' ', '>', ' '), ('h', 'i', 'n'),
('n', ' ', '>'), ('h', 'o', 'e'), ('w', 'e', 'l'), ('e', 'n', ' '),
('l', 'o', 't'), (',', ' ', 'S'), (' ', '&', ' '), ('r', 't', 'w'),
('i', 'n', 'g'), ('s', ' ', '&'), ('>', ' ', 'W'), ('s', ' ', '>'),
('t', 'h', 'i'), ('e', 'w', 'e'), ('y', ' ', '>'), ('&', ' ', 'J'),
('r', 'y', ' '), ('S', 'm', 'a'), ('l', 'r', 'y'), ('>', ' ', 'S'),
('m', 'e', 'n'), (' ', 'W', 'a'), ('g', ',', ' '), ('o', 't', 'h'),
('t', 'c', 'h'), ('w', 'a', 't'), ('W', 'a', 't'), ('S', 'h', 'o'),
```

```
('t', 'w', 'a'), ('W', 'o', 'm'), ('a', 't', 'c'), (' ', 'W', 'o'),
(' ', 'J', 'e'), ('n', 'g', ','), ('e', 'l', 'r'), (' ', 'S', 'm')}
```

We carry out the same for the second class and compute the number of intersecting trigrams for both sets. We plug this into the nominator of the formula above, calculate the denominator and arrive at a similarity of $0.321$ for the two strings.

### 5.1.3 Path Similarity

The two baselines methods that we described previously take the full string into account without considering the hierarchical structure of it. We introduce a measure that extends those methods to hierarchical strings in this Subsection.

Classes at the top of a hierarchy are, per definition, broader than classes at the lower levels. In case we want to check if two classes contain the same products, the lower levels are, therefore, more relevant to our computation.

The path similarity takes this into account and puts high weight on the comparison of the last elements while reducing the importance of the remainder. It is derived from the path distance given in [13, p. 95], but we implemented it using similarities. We define path similarity as:

$$\sigma(s^n, t^m) = \lambda \cdot \hat{\sigma}(s^n, t^m) + (1 - \lambda) \cdot \sigma(s^{(n-1)}, t^{(m-1)})$$

where $s^n$ and $t^m$ are sequences of strings and $\hat{\sigma}$ is a string-based similarity function, like one of those introduced in Subsection 5.1.1 and 5.1.2. If one sequence is empty, $\sigma$ returns 0 as the similarity.

Again, we apply the similarity function to "Clothing, Shoes & Jewelry $>$ Women $>$ Watches $>$ Smartwatches" with "Wearable Technology $>$ Smartwatches & Accessories $>$ All Smartwatches" and use the Levenshtein-similarity internally. We assume that $\lambda = 0.7$. First, we compare "Smartwatches" with "All Smartwatches" with the Levenshtein-similarity and multiply this with $\lambda$. This results in an intermediary result of 0.525. Next, we call the path-similarity recursively with the remainder, i.e., "Clothing, Shoes & Jewelry $>$ Women $>$ Watches" and "Wearable Technology $>$ Smartwatches & Accessories". The result is multiplied with 0.3 and added to the already computed 0.525. In the last recursive call, we compare "Clothing, Shoes & Jewelry" with nothing and, in this case, simply return 0.

The overall similarity between our input strings is 0.583. In our experiments, we will use the Levenshtein- and N-Gram-similarity for $\hat{\sigma}$.

In this Section, we introduced three baseline methods that use simple string similarity measures and one that uses the hierarchical structure to compute a similarity between two classes. They will serve as a baseline to compare the more advanced algorithms to, which we will present in the coming sections.

## 5.2 WordNet-Based Matching Methods

This Section covers unsupervised taxonomy matching methods, i.e., methods that take as input the raw classes and other static third-party resources like WordNet [24]. We only use the training dataset to optimize certain hyperparameters. First, we introduce S-Match or Semantic Match [14], which is a generic taxonomy mapping algorithm that provides semantic annotations, i.e., not only *equal* or best-match predictions, but also labels for *contains*, *contained-in*, and *disjoint* relations. The second algorithm is an extension of the product taxonomy mapping algorithm by Park and Kim [28] and is called SCHEMA [2].

### 5.2.1 S-Match

Giunchiglia et al. [14] state that S-Match is based on two fundamental ideas:

- Discover mappings by computing semantic relations.
- Determine semantic relations by analyzing the meaning.

Contrary to the approaches we described earlier, there is no threshold or hyperparameter that we can tune. Instead of computing a similarity, the algorithm translates the taxonomies into a satisfiability problem and computes a relation directly.

The authors use specific terms to identify parts of the taxonomy. We will first introduce the terms and then describe the individual steps of the algorithm. A taxonomy is a tree-like structure and consists of *labels* and *nodes*. Labels are individual edges in the tree, e.g., ”Clothing, Shoes & Jewelry $>$ Women $>$ Watches $>$ Smartwatches” contains the labels ”Women”, ”Watches”, etc. The whole string is called a node. In our case ”node” translates to ”class-label” and ”label” to ”category”. We will continue the explanation using our terms.

S-Match has four steps that are used to match two taxonomies:

1. Compute the concepts of all categories in the tree.
2. Compute the concepts of all class-labels in the tree.
3. Compute relation between all concepts of category-pairs.
4. Compute relation between all concepts of class-label pairs.

Step 1 takes each category, e.g., ”Watches”, and computes the meaning of the category. To extract the meaning, S-Match relies on an external source or oracle, in this case, WordNet. Instead of using ”Clothing, Shoes & Jewelry” directly, the terms are first split up (tokenized), then reduced to a normalized form

(stemmed) and then mapped to one or more WordNet senses. The provided example would result in ”Clothing — Shoes — Jewelry” ⇒ ”Cloth — Shoe — Jewelry” ⇒ ”cloth.n.01 — shoe.n.01 — jewelry.n.01”. In the case of multiple terms, we take the logical disjunction of them and use it as the concept of the category. The concept of the category ”Clothing, Shoes & Jewelry” would, therefore, be ”cloth.n.0.1 ∨ shoe.n.01 ∨ jewelry.n.01”.

The next step combines the concept of all categories in a class-label into the concept of a class-label. We use the conjunction of category concepts, which would result in the following logical formula for our example: ”(cloth.n.0.1 ∨ shoe.n.01 ∨ jewelry.n.01) ∧ (women.n.01) ∧ (watch.n.01) ∧ (smartwach.n.01)”.

All of the steps above only rely on information present in a single taxonomy and the corresponding oracle. Hence, we can consider them as ”offline” since they can be precomputed. The last two steps combine information from multiple taxonomies and are, therefore, considered to be ”online”.

In the third step, we build up a relation matrix between the concepts of categories. Every category in the one taxonomy is matched with every category from the other taxonomy using a group of matchers.

| **Matcher name** | **Execution order** | **Approximation level** | **Matcher type** | **Schema info** |
|---|---|---|---|---|
| Prefix | 2 | 2 | String-based | Labels |
| Suffix | 3 | 2 | String-based | Labels |
| Edit distance | 4 | 2 | String-based | Labels |
| Ngram | 5 | 2 | String-based | Labels |
| Text corpus | 12 | 3 | String-based | Labels + corpus |
| WordNet | 1 | 1 | Sense-based | WordNet senses |
| Hierarchy distance | 6 | 3 | Sense-based | WordNet senses |
| WordNet gloss | 7 | 3 | Gloss-based | WordNet senses |
| Extended WordNet gloss | 8 | 3 | Gloss-based | WordNet senses |
| Gloss comparison | 9 | 3 | Gloss-based | WordNet senses |
| Extended gloss comparison | 10 | 3 | Gloss-based | WordNet senses |
| Extended semantic gloss comparison | 11 | 3 | Gloss-based | WordNet senses |

Figure 5.1: S-Match Matching Methods [14]

As we can see in Figure 5.1 the matchers are categorized into three approximation levels. The first level relies strictly on the WordNet meaning computed in step 1 and relations returned by it are always correct. In case a matcher on one level does not provide a confident prediction, there is an automatic fallback to the next method and level. The second approximation level relies mostly on the actual strings of the category. String-based methods like the Levenshtein-similarity are used, and in case they are unable to provide a relation additional, but more unreliable, features of WordNet are applied. Some of those are inspired by COMA++ by

Aumueller et al. [7].

Now, that the category relations are available, we can match class-labels. To do this, we will compare the class-label already given above with "Jewelry & Watches > Watches, Parts & Accessories". S-Match predicts a relation between the first two categories in both nodes, but is not sure which relations holds. It gives a firm prediction that "Watches" is contained-in "Watches, Parts & Accessories". This is translated into a satisfiability problem of the form $axiom \rightarrow rel(concept_l, concept_r)$ where $rel$ is the relation we want to prove. In this case, we will simply try the *contained-in* relation, formulate the negation $axiom \wedge \neg rel(concept_l, concept_r)$. At this point in time, DPLL [11] is used to check for satisfiability, and if it finds a contradiction, we predict the given relation.

The authors provide an implementation that is hosted on GitHub[2], which we will reuse for our experiments.

### 5.2.2 SCHEMA

The second algorithm in this Section is SCHEMA by Aanen et al. [2]. It extends on the ideas of Park and Kim [28] in the sense that they also use word sense disambiguation on top of WordNet [24]. Since SCHEMA is supposed to be a mapping algorithm that simply finds the best-match between two nodes in a taxonomy, we deviate slightly from the algorithm described by Aanen et al. The algorithm also provides a semantic match in the sense that it compares a source and a target node and predicts if the source node contains the target node. To map this to our desired outputs, we take each node as a source *and* as a target node. If they are both labelled as supersets of each other, we predict *equal*. If only one is a superset of the other, we label it as *contains*, and if none of the above conditions apply, we label the pair as *disjoint*.

SCHEMA is implemented in three steps:

1. Source Category Disambiguation.
2. Candidate Target Category Selection.
3. Candidate Target Path Key Comparison.

The first step takes the source category and tries to find its intended meaning. Composite classes frequently occur in product taxonomies. We already saw an example in the previous Section: "Clothing, Shoes & Jewelry". All of those could be a class in itself and may lead to problems, since another online shop may combine their

[2] `https://github.com/opendatatrentino/s-match`. Accessed: 01.05.2020

products in different composite classes. Hence, they are separated into a split-term set.

Now, the sense for every split-term is computed. The authors propose to use WordNet to get synonyms of the current term and, to exclude misleading senses, use the terms in the parent element of the hierarchy to provide a context to filter the synonyms. Taking "Watches > Bands" as an example, we would take "Bands" as the split term set and compute its synonyms. Now, it might be possible that a "Band" in terms of a music-making group is included there. SCHEMA also adds the context of "Watches" into this consideration and, therefore, the music-making group can be excluded, while the sense of a watch-band is retained. The foundation of this idea is taken from Lesk [19].

Given the senses and synonyms of the source category, close categories in the target taxonomy can be found. For each pair of categories, a semantic match is computed that labels the target category as containing the source category or the two categories as *disjoint*. Similar to S-Match 5.2.1 there are multiple matchers used in this prediction. The target category contains the source category, if the senses of the source category are a subset of it, e.g., "Shoe" as the source category would be contained-in "Clothing, Shoes & Jewelry". If this is ambiguous, the Levenshtein-similarity is applied, and if it exceeds a given threshold, the source category is also labelled as being a subset of the target category.

Out of the target categories that contain the source relation, SCHEMA would try to find the best match in the third step to create a mapping between them. Since we want to compare two categories directly, we skip the third step and, instead, switch source and target categories and run the prediction again. This results in an *equal*, *contains*, *contained-in* or *disjoint* prediction.

Our implementation is based on an open-source implementation of SCHEMA with a few extensions to our use case. See GitHub[3] for the reference implementation.

## 5.3 Supervised Taxonomy Matching Methods

One of the major drawbacks of the WordNet-based taxonomy matching methods is that the WordNet corpus is kind of small compared to the number of possible words that may be used in taxonomies across the web. If the word is out of the given vocabulary, no synonyms and, therefore, no similarity can be computed.

We propose multiple methods in the following Section that introduce certain ways to avoid this limitation of WordNet-based models.

[3] `https://github.com/nudge/schema`. Accessed: 01.05.2020

At first, we will have a look at word embedding based models following Zhang et al. [37]. Then, we use standard Python libraries to derive features from the given classes and apply machine learning classification algorithms.

### 5.3.1 Ontology Matching with Word Embeddings

Zhang et al. [37] claim that the WordNet corpus is insufficient for taxonomy matching and propose to use word embeddings instead. They use word2vec [23] on the Wikipedia corpus to compute an embedding of each word and apply the Cosine similarity (CS) to calculate the similarity between words. For each entity in one taxonomy, they find the most similar entity in the other taxonomy given the entity's embedding.

While word embeddings are able to greatly enhance the available vocabulary, they may introduce new problems, since they tend to coalesce semantic similarity and conceptual association. "Horse" and "Harness" may be similar and in a related context, but they should not be matched in a taxonomy matching task [17].

Since Zhang et al. only try to find the most similar class and we intend to classify a pair of classes, we use a slightly different setup. We employ a pre-trained word2vec model trained on about 100 billion Google news word that can be downloaded via Gensim [33]. For every word in a class, we retrieve its embedding, a 300-dimensional vector, and take an average of the individual word vectors to represent the whole class. Afterwards, we compute the similarity for both complete class-labels and with the lowest category removed on each class as as performed in Section 5.1.1 and 5.1.2.

We use those similarity scores to make a prediction. If a word is still not included in our vocabulary, we simply ignore it. A possible future extension may be the usage of fastText[4] that also enables the embedding of out-of-vocabulary words. Another extension could be an increased weight of the lower-level labels in the class hierarchy.

### 5.3.2 Machine Learning Classification

The problem we cover in this Thesis is finding the most suitable label for a pair of classes that come from e-commerce product taxonomies. This can be reduced to a standard machine learning classification, where two inputs are given and an output label should be predicted. We, therefore, deviate from the approaches of the ontology matching literature and focus on standard machine learning classification algorithms as provided by scikit-learn[5].

[4] `https://fasttext.cc`. Accessed: 01.05.2020

[5] `https://scikit-learn.org/stable/index.html`. Accessed: 01.05.2020

To apply the machine learning algorithms, we have to transform our class strings into a numeric vector that can be fed into the models. We use the included CountVectorizer[6], which indicates basically how often a certain word appears in the given string. We also use the word2vec model to compare the results of both embedding models.

Imagine a dataset of the following strings:

- One Ring to rule them all, One Ring to find them,
- One Ring to bring them all and in the darkness bind them.

If we reduce our vocabulary to the strings [one, ring, rule, them, all, find, darkness, bind, bring] the two sentences result in the following vectors:

- $[2, 2, 1, 2, 1, 1, 0, 0, 0]$
- $[1, 1, 0, 2, 1, 0, 1, 1, 1]$

Since product taxonomies are structured, they do not contain any stop word that should usually be removed. We also expect words to be normally distributed and, therefore, do not weight the results by inverse document frequency (c.f. tf-idf). The resulting vector has a number of dimensions that is equal to the vocabulary size. Next, the input dataset is transformed by replacing the strings with vectors and concatenating the two vectors to produce a single feature vector.

For the prediction, we considered the following classification models. We started with multinomial Naive Bayes, "one of the two classic naive Bayes variants used in text classification"[7]. Based on the training data, a distribution is parameterized that predicts the probability of a label, given the evidence.

The next algorithm is Stochastic Gradient Descent (SGD)[8], which trains a linear classifier (Support Vector Machine (SVM)/Logistic Regression) depending on the provided loss function. In our case, the optimal loss function is derived via the grid search. A linear classifier aims to find a decision boundary or hyperplane that separates differently labelled points in the vector space and makes predictions based on this hyperplane.

We also use AdaBoost[9], an ensemble method that combines multiple classifiers to enhance its predictions. It relies on simple decision trees and, after finding a

[6] `https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html`. Accessed: 01.05.2020

[7] `https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes`. Accessed: 01.05.2020

[8] `https://scikit-learn.org/stable/modules/linear_model.html#stochastic-gradient-descent-sgd`. Accessed: 01.05.2020

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`. Accessed: 01.05.2020

good initial fit, it trains additional models with different parameters to optimize for edge cases.

Finally, we train a Multi-Layer Perceptron (MLP)[10], a kind of Neural Network. It combines multiple layers of neurons that use a non-linear activation function and is trained similarly to the SGD models. Sonoda and Murata [35] show that a Neural Network with sufficient neurons is a universal approximator, i.e., it can represent any function arbitrarily well. Hence, with sufficient training data and a sufficiently complex model, we can represent the training data perfectly. Given the limited size of our gold standard, we will use simple Neural Network models in our evaluation.

To optimize the hyperparameters of the machine learning models, we use grid search, i.e., different hyperparameters are tested on a subset of the actual training data and the final model is trained using the optimal hyperparameters and the full training set. In Table 5.1 we present the parameters that were tested for each model described above. We will only list the parameters in which we deviated from the default or, in case the default is among the possible values, highlight it in bold. For a description of the individual model parameters, we refer the reader to the excellent scikit-learn documentation referenced in the footnote of the specific model.

| Model | Parameter | Values |
|---|---|---|
| SGDClassier | alpha | 0.01, 0.001 |
| | loss | **hinge**, modified_huber |
| AdaBoostClassifier | learning_rate | 0.3, **1.0**, 1.7 |
| | algorithm | SAMME, **SAMME.R** |
| MLPClassifier | activation | logistic, **relu** |
| | hidden_layer_sizes | (50,), (25, 25), (50, 25) |
| | alpha | **0.0001**, 0.0003, 0.001 |

Table 5.1: Grid Search Hyperparameters.

## 5.4 Summary

In this Chapter, we introduced multiple classification methods for class-label pairs derived from e-commerce taxonomies. We used naive string matching methods as a baseline and an enhanced version that takes the hierarchical structure into account. Two suitable algorithms from related literature were introduced that use WordNet as an oracle for providing synonyms. Finally, a method based on current research

[10] `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`. Accessed: 01.05.2020

using word embeddings to circumvent WordNet's limitations and an ensemble of open source machine learning classification methods were employed.

# Chapter 6

# Experiment Results

The content of this Chapter is the evaluation of the taxonomy matching algorithms. We will use the algorithms described in Chapter 5 on our dataset from Chapter 4 and report the results. This Chapter focuses on the raw data and a description of the experiments and the coming Chapter 7 analyzes possible misclassifications of some algorithms in more detail and also discusses those results.

Certain metrics are considered important when evaluating an algorithm. We are interested in the overall precision, recall, and F1-score, i.e., how many class-label pairs are correctly classified and how many positive tuples we missed.

The F1-score and precision and recall are usually considered for binary classification problems, especially if the classes are imbalanced. Since our problem is a multi-label classification problem, we compute the precision, recall, and F1-score for each class and report them individually. For the hyperparameter tuning, we use an unweighted average of the F1-score of the positive labels. This is called the macro-F1-score.

In general, we consider it more relevant to detect a relationship between two categories, meaning that a misclassification between *equal*, *contains*, and *contained-in* should be preferred over a *disjoint* label since those are under-represented. Even if we return a certain number of false positives, i.e., disjoint pairs labelled as *equal*, *contains*, or *contained-in*, the workload for a human to reclassify them would be greatly reduced, compared to complete manual labelling.

This is also the approach taken by Park and Kim [28] who accept a loss of precision for higher recall.

The confusion matrices in this Chapter follow the conventions of scikit-learn. "By definition a confusion matrix $C$ is such that $C_{i,j}$ is equal to the number of observations known to be in group $i$ and predicted to be in group $j$"[1]. We will

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.`

illustrate this using the confusion matrix for the Levenshtein-similarity in Table 6.4. Along the diagonal from top-left to bottom-right, we find all true positives, i.e., the number of cases where the true label is predicted. The value 91 in cell $C_{2,1}$ indicates that 91 *contains* labels are mistakenly predicted as *equal*.

This Chapter will follow the same structure as Chapter 5. We will start with the baseline method evaluations and move on to the advanced unsupervised and supervised models.

The precision, recall, and F1-score per method are shown in Tables 6.1, 6.2, and 6.3.

| Method | Precision | Recall | F1-score |
|---|---|---|---|
| Levenshtein | 0.038 | 0.34 | 0.067 |
| N-Gram | 0.341 | 0.26 | 0.288 |
| Levenshtein (PD) | 0.035 | 0.41 | 0.064 |
| N-Gram (PD) | 0.055 | 0.386 | 0.083 |
| SCHEMA | 0.105 | 0.091 | 0.086 |
| Embedding CSS | 0.147 | 0.378 | 0.207 |
| AdaBoost BoW | 0.049 | 0.177 | 0.073 |
| Naive Bayes | 0.067 | 0.238 | 0.103 |
| SGD BoW | 0.088 | 0.094 | 0.08 |
| S-Match | 0.051 | 0.014 | 0.021 |
| AdaBoost Embedding | 0.044 | 0.169 | 0.068 |
| SGD Embedding | 0.069 | 0.219 | 0.101 |
| MLP | 0.206 | 0.162 | 0.177 |

Table 6.1: Precision, Recall, and F1-score for Equal Label.

## 6.1 Baseline Methods

### 6.1.1 Levenshtein- or Edit-Similarity

The Levenshtein-similarity is one of the simplest approaches for taxonomy matching and, therefore, we expect mediocre results. It has one of the highest results for true positives with regard to the *equal* label with 143 out of 241 labels being predicted correctly. This is also reflected in the recall of 0.34. This, comparatively, good recall comes with low precision, though. 2,180 actual disjoint labels were predicted as *equal*. In general, we observe a high number of false positives with

metrics.confusion_matrix.html.Accessed:01.05.2020

| Method | Precision | Recall | F1-score |
|---|---|---|---|
| Levenshtein | 0.065 | 0.017 | 0.09 |
| N-Gram | 0.112 | 0.149 | 0.119 |
| Levenshtein (PD) | 0.264 | 0.169 | 0.188 |
| N-Gram (PD) | 0.16 | 0.128 | 0.128 |
| SCHEMA | 0.049 | 0.067 | 0.048 |
| Embedding CSS | 0.046 | 0.245 | 0.086 |
| AdaBoost BoW | 0.035 | 0.067 | 0.043 |
| Naive Bayes | 0.091 | 0.141 | 0.102 |
| SGD BoW | 0.079 | 0.077 | 0.066 |
| S-Match | 0.036 | 0.026 | 0.028 |
| AdaBoost Embedding | 0.04 | 0.089 | 0.051 |
| SGD Embedding | 0.052 | 0.12 | 0.068 |
| MLP | 0.085 | 0.045 | 0.054 |

Table 6.2: Precision, Recall, and F1-score for Contains Label.

| Method | Precision | Recall | F1-score |
|---|---|---|---|
| Levenshtein | 0.077 | 0.198 | 0.109 |
| N-Gram | 0.016 | 0.183 | 0.03 |
| Levenshtein (PD) | 0.321 | 0.228 | 0.258 |
| N-Gram (PD) | 0.24 | 0.205 | 0.207 |
| SCHEMA | 0.042 | 0.078 | 0.053 |
| Embedding CSS | 0.054 | 0.245 | 0.086 |
| AdaBoost BoW | 0.096 | 0.176 | 0.111 |
| Naive Bayes | 0.114 | 0.223 | 0.144 |
| SGD BoW | 0.13 | 0.17 | 0.138 |
| S-Match | 0.072 | 0.1 | 0.08 |
| AdaBoost Embedding | 0.082 | 0.18 | 0.107 |
| SGD Embedding | 0.121 | 0.267 | 0.163 |
| MLP | 0.281 | 0.208 | 0.228 |

Table 6.3: Precision, Recall, and F1-score for Contained-In Label.

the Levenshtein-similarity. 3370 out of 5045 disjoint labels received a positive label. On the other hand, 109 out of 739 positive examples are labelled as disjoint. Table 6.4 contains the full confusion matrix.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 143 | 43 | 37 | 18 |
| c | 91 | 93 | 31 | 42 |
| ci | 68 | 49 | 75 | 49 |
| d | 2180 | 646 | 544 | 1675 |

Table 6.4: Levenshtein Confusion Matrix.

### 6.1.2 N-Gram-Similarity

According to the F1-Score, the N-Gram-similarity model is the best for predicting the *equal* label and among the best for *contains*. Only *contained-in* has a very low precision due to a high number of false positives. Again, about 3000 negatives received a positive label, but those cases are more concentrated among the *contains*/*contained-in* label, with the latter making up the majority. The full confusion matrix is shown in Table 6.5.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 101 | 71 | 46 | 23 |
| c | 32 | 93 | 69 | 63 |
| ci | 33 | 90 | 73 | 45 |
| d | 30 | 259 | 2679 | 2077 |

Table 6.5: N-Gram Confusion Matrix.

### 6.1.3 Path Similarity

The result for the path-similarity metric based on the Levenshtein-similarity is very similar for the *equal* label, but the path similarity significantly outperforms the Levenshtein-similarity for *contains* and *contained-in*. For both of those labels, it ranks among the best approaches. As with the Levenshtein-similarity, we observe a high number (2707) of disjoint labels that received an *equal* label.

The surprisingly good F1-score for the *equal* label with the N-Gram-similarity are not replicated with the path distance similarity based on the N-Gram-similarity.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 161 | 34 | 24 | 22 |
| c | 84 | 89 | 14 | 70 |
| ci | 67 | 28 | 88 | 58 |
| d | 2707 | 203 | 127 | 2008 |

Table 6.6: Levenshtein Path Distance Confusion Matrix.

Instead, the precision, recall, and F1-score are close to, but slightly better, to both Levenshtein approaches. The path distance F1-score on the *contains* label is slightly better than with pure N-Gram-similarity, and the F1-score for *contained-in* outperforms the basic N-Gram-similarity significantly. This is due to a high difference in the precision of 0.224. The high number of misclassified disjoint labels that we observed for *contained-in* labels with the standard N-Gram-similarity shifted to the *equal* label. This shift also explains the decline in the F1-score for the *equal* label. See Table 6.6 and 6.7 for the path-distance based on Levenshtein and on N-Gram, respectively.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 147 | 26 | 35 | 33 |
| c | 94 | 50 | 26 | 87 |
| ci | 74 | 17 | 83 | 67 |
| d | 2154 | 146 | 185 | 2560 |

Table 6.7: N-Gram Path Distance Confusion Matrix.

## 6.2 WordNet-Based Matching Methods

### 6.2.1 S-Match

The semantic match algorithm introduced by Giunchiglia et al. [14] consistently ranks among the worst performers across all three positive labels. In the confusion matrix shown in Table 6.8 we can see that there is a strong tendency to label pairs as disjoint. Out of 739 actual positive labels, only 131 are labelled as such. That is before factoring in misclassifications among the positive labels. On the other hand, the number of disjoint pairs that are classified as positive is one of the lowest we observe. The accuracy would, therefore, be high, but, as we and other authors stated previously, the recall is the more important metric for product taxonomy

matching. One surprising result is that neither *contains* nor *contained-in* pairs are classified as *equal*. Although this might be due to the overall low number of *equal* predictions.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 7 | 28 | 31 | 175 |
| c | 0 | 19 | 4 | 234 |
| ci | 0 | 2 | 40 | 199 |
| d | 55 | 250 | 232 | 4508 |

Table 6.8: S-Match Confusion Matrix.

### 6.2.2 SCHEMA

SCHEMA results are comparable to the ones of S-Match that we described in the previous Subsection. The algorithm assigns more positive labels overall, but the precision, recall, and F1-scores are similar. Again, the accuracy would be high for the overall predictions. The confusion matrix for SCHEMA is shown in Table 6.9.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 33 | 56 | 25 | 127 |
| c | 11 | 41 | 32 | 173 |
| ci | 8 | 52 | 28 | 153 |
| d | 158 | 484 | 307 | 4096 |

Table 6.9: SCHEMA Confusion Matrix.

## 6.3 Supervised Taxonomy Matching Methods

### 6.3.1 Ontology Matching with Word Embeddings

Using word2vec together with Cosine similarity results in a comparatively high recall across all labels. No method we evaluated so far has shown good F1-scores across all three labels. For the *equal* label, the embedding model also has a comparatively good precision leading to the second-best F1-score after the N-Gram-similarity. The confusion matrix for word embeddings with Cosine similarity is shown in Table 6.10.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 153 | 26 | 44 | 18 |
| c | 40 | 74 | 83 | 60 |
| ci | 61 | 18 | 100 | 62 |
| d | 368 | 928 | 916 | 2833 |

Table 6.10: Embedding CS Confusion Matrix.

### 6.3.2 AdaBoost

The AdaBoost model based on BoW vectors performs best for the *contained-in* label according to the F1-score. It also shows good results for *equal-* and *contained-in* recall, but a high number of *contains* pairs are misclassified. See Table 6.11 for the confusion matrix.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 64 | 13 | 58 | 106 |
| c | 36 | 58 | 54 | 109 |
| ci | 58 | 36 | 74 | 73 |
| d | 822 | 675 | 921 | 2627 |

Table 6.11: AdaBoost BoW Confusion Matrix.

The results for AdaBoost based on word2vec embeddings is very close to the WordCount version, as we can see in the precision, recall, and F1-score, but also in the confusion matrix in Table 6.12. It seems that the embedding algorithm only has a minor influence on this type of model.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 65 | 19 | 39 | 118 |
| c | 57 | 45 | 21 | 134 |
| ci | 55 | 20 | 68 | 98 |
| d | 870 | 692 | 594 | 2889 |

Table 6.12: AdaBoost Embedding Confusion Matrix.

### 6.3.3 Naive Bayes

The Naive Bayes model we trained on the CountVectorized class-labels achieves consistent scores for precision, recall, and F1-score across all positive labels. In the confusion matrix in Table 6.13 we can see that about 75 percent of disjoint pairs are predicted correctly, putting the Naive Bayes model into the range of the WordNet-based models with regard to this. It outperforms them on all measures except for the precision on the *equal* label, where SCHEMA has a better result. While the other models usually had two labels with similar results and one negative outlier, i.e., one label with a worse F1-score, Naive Bayes has a positive outlier. For the *contained-in* label, it achieves a slightly higher precision than for the other two.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 96 | 8 | 37 | 100 |
| c | 33 | 53 | 35 | 136 |
| ci | 39 | 23 | 72 | 107 |
| d | 671 | 286 | 336 | 3752 |

Table 6.13: Naive Bayes Confusion Matrix.

### 6.3.4 Stochastic Gradient Descent

The SGD models were also trained on the BoW and the word2vec embeddings. In contrast to the AdaBoost model, where both types of embeddings led to similar results, the word2vec embeddings outperform the BoW embeddings on the F1-score for all labels. For the *equal* and *contained-in* pairs, the word2vec model is about 10 percent better than the WordCount based model. This is due to a higher number of correctly predicted positive pairs, but there is also a slight incline of the number of false positives. The confusion matrices for the SGD models are presented in Tables 6.14 and 6.15.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 35 | 15 | 31 | 160 |
| c | 20 | 37 | 14 | 186 |
| ci | 20 | 11 | 63 | 147 |
| d | 408 | 446 | 345 | 3846 |

Table 6.14: SGD BoW Confusion Matrix.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 88 | 24 | 35 | 94 |
| c | 41 | 50 | 26 | 140 |
| ci | 43 | 15 | 97 | 86 |
| d | 736 | 621 | 566 | 3122 |

Table 6.15: SGD Embedding Confusion Matrix.

### 6.3.5 Multi-Layer Perceptron

The MLP shows good results for the *equal* and *contained-in* pairs. It also classifies more disjoint labels correctly than any other model in our experiments. During the experiments, we observed warnings that the model has not converged yet. The training set may be too small for the given network. Hence, the overall performance may improve with more training data.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 64 | 17 | 24 | 136 |
| c | 24 | 31 | 13 | 189 |
| ci | 18 | 8 | 78 | 137 |
| d | 123 | 194 | 154 | 4574 |

Table 6.16: MLP Confusion Matrix.

## 6.4 Summary

In this Chapter we presented the results of different taxonomy matching approaches on our gold standard. Simple setups like N-Gram- and Cosine-similarity on word2vec embeddings achieved good results in predicting if two class-labels are equal to each other. On the other hand, no model performed well in predicting if one class contains the other, while the more complicated models, read supervised machine learning, did a good job in predicting if one class is a subset of another. This comes as a surprise to us, because we assumed that the model should be able to predict if one class is more general or less general equally well. We will look at the specific errors of each individual model in the next Chapter and discuss the results that we have presented here.

# Chapter 7

# Error Analysis and Discussion

In this Chapter, we will analyse the experimental results we presented in the previous Chapter. We look at specific errors the model makes, give examples, and discuss the results. Again, we start with the baseline models, continue with the WordNet-based algorithms, and conclude with the supervised methods.

For each algorithm, we will first name the different types of errors and provide a hypothesis on the root-cause of those. Then, we will give example predictions that support our hypothesis or allow us to reject it. Finally, we name scenarios in which the algorithm may prove useful.

## 7.1 Baseline Methods

### 7.1.1 Levenshtein- or Edit-Similarity

We saw that the Levenshtein-similarity is good at predicting equality for pairs that are actually *equal*, but also assigned an *equal* label to many *disjoint* pairs. Our first assumption, therefore, is that two classes which are *disjoint*, but have a very similar path, e.g., due to common categories in their class-label. We would also like to remind the reader that we generated negative corner-cases with a layout that provokes errors like this. See the description in Section 4.3 for all details. Further, we assume that the false positives for *contains* and *contained-in* stem from the same problem. A third hypothesis is that minor changes in the category order or wording may make a major difference during the prediction. Now, we will look at each of the three hypotheses in turn and give examples.

The first one is the effect of our generated edge-cases on the *disjoint* labels that were labelled as *equal*. Two examples of predictions that support this hypothesis are given in Table 7.1. Actually, we were not able to find an example of a generated

| | Class-Label |
|---|---|
| left | TV & Home Theater > TV & Home Theater Accessories > TV Antennas |
| right | TV & Home Theater > TV & Home Theater Accessories > Remote Controls |
| left | Clothing, Shoes & Jewelry > Boys > Clothing > Shorts |
| right | Clothing, Shoes & Jewelry > Boys > Clothing > Jeans |

Table 7.1: Levenshtein: Examples for False Positive Equal Predictions on Corner-Cases.

corner-case that was labelled correctly as *disjoint*.

Instead, the *disjoint* labels that were detected by the Levenshtein-similarity are rather obvious. We give a representative example in Table 7.2. The pair consists of a clothing product and something in the electronics domain.

| | Class-Label |
|---|---|
| left | Clothing, Shoes & Jewelry > Men > Clothing > Swim > Briefs |
| right | Cell Phones > Cellphone Accessories > Bluetooth Headsets |

Table 7.2: Levenshtein: Correctly Classified Disjoint Pair.

Our second assumption is that the corner-case issue extends to the two other positive labels. Examples are given in Table 7.3. In the left class-label of the

| | Class-Label |
|---|---|
| left | Electronics > Home Audio > Speakers > Center-Channel Speakers |
| right | Electronics > Home Audio > Speakers > Subwoofers |
| left | Electronics > Portable Audio > Portable Audio |
| right | Electronics > Portable Audio > Radios |
| left | Clothing, Shoes & Jewelry > Men > Clothing > Fashion Hoodies & Sweatshirts |
| right | Clothing, Shoes & Jewelry > Men > Clothing > Jackets & Coats |

Table 7.3: Levenshtein: Examples for False Positive Contains and Contained-In Predictions on Corner-Cases.

second example, we see another problem with taxonomies crawled from the Semantic Web. Sometimes they contain duplicates in the hierarchy string. Zhang

and Paramita [38] tried to find a good cleaning and normalization algorithm to reduce this type of duplication, but found that this is counterproductive to the actual matching, because it also removes subtle differences that are important for the classification.

Next, we will take a look at the misclassifications among the positive labels. Examples are shown in Table 7.3. The first example illustrates how the addition of

| | Class-Label |
|---|---|
| left | Cameras & Camcorders > Digital Cameras > Mirrorless Cameras |
| right | Electronics > Cameras & Camcorders > Mirrorless Cameras |
| left | Home > Apparel & Accessories > Watches > Watches > Tissot |
| right | Jewelry > Watches > Luxury Watches > Tissot Watches |
| left | Clothing, Shoes & Jewelry > Women > Clothing > Coats, Jackets & Vests |
| right | Clothing > Womens Clothing > Womens Coats & Jackets |
| left | Sports & Outdoors > Sports & Fitness > Golf |
| right | Sports & Outdoors > Sports > Golf Equipment > Golf Shirts |

Table 7.4: Levenshtein: Examples for Misclassifications among Positive Labels.

an intermediate level may change the interpretation completely. The actual label should be *equal*, but Levenshtein predicted that the left class-label is *contained-in* the right class-label. In the third example, "Vests" where added as a composite category. Therefore, the actual label should be *contained-in*, but due to the high similarity that those two labels still have, Levenshtein predicts this pair to be *equal*. The problems with composite categories were identified by Aanen et al. [2] during the development of the SCHEMA algorithm.

Overall the Levenshtein-similarity tends to predict a positive label if there is a slight resemblance between the two class-labels. On the other hand, it has problems with composite categories, where the order does not change the semantic meaning, but the layout of the string. The Levenshtein-similarity is not useful as a standalone taxonomy matcher, and due to the large number of false positives, it also does not ease the task of a human annotator significantly.

### 7.1.2 N-Gram-Similarity

Our second baseline method is the N-Gram-similarity, which achieved the best F1-score for the *equal* label and a comparatively high score for *contains*. In the prediction of the *contained-in* class-label pairs, it is among the worst performers. In this Section, we will cover plausible explanations for those results.

The high precision on the *equal* pairs combined with a good recall indicates that the N-Gram-similarity is good at detecting *equal* labels without misclassifying other pairs as *equal*. One issue that we have detected for the Levenshtein-similarity is the ordering of categories in the hierarchy and especially in composite categories. "Clothing & Jewelry" and "Jewelry & Clothing" are very different according to the Levenshtein-similarity, but due to the set-based nature, the N-Gram-similarity does not care about the order. It would only detect minor differences around the start, the end, and around the ampersand. Hence, our first assumption is that the N-Gram-similarity handles composite categories better than Levenshtein-similarity. This would explain the good recall.

A reason for the high precision on the *equal* class-label pairs, i.e., the low number of false positives, may be explained by the high number of false positives for the other two positive labels. Again, they may stem from the generated corner-cases and, if we check for containment there, we get a complete intersection of the sets of N-Grams and normalize afterwards. Those cases use a weak spot in our implementation of the string-based similarity measures, because we assume that one class $A$ is more general than another class $B$, if $A$ is very similar to $B$ without $B$'s last category. Therefore, the second assumption is that generated corner-cases are labelled as either *contains* or *contained-in*, actual *disjoint* class-label pairs are labelled as such, and only a small number of actual *disjoint*s receive an *equal* label. This hypothesis also covers the large number of false positives that we observe for the *contained-in* label.

To support the first hypothesis, we will look at classes with composite categories that were misclassified by the Levenshtein-similarity and check what the N-Gram-similarity predicted. Of the four examples presented in Table 7.4 the first three are predicted correctly, and only the last one receives a wrong label by the N-Gram-similarity. Instead of correctly labelling this pair as *contains*, both methods predict equality.

Next, we focus on the large number of *disjoint* class-label pairs that received a *contained-in* label. Examples are given in Table 7.5. We could present hundreds of examples that follow this scheme. Given our implementation, we can also explain the affinity for the *contained-in* label over the *contains* label. We compute a similarity score for each positive label and use the maximum value for the prediction if it exceeds a given threshold. If there is a draw, *equal* is picked over *contained-in* and *contained-in* over *contains*. Hence, almost all artificial corner-cases are labelled as *contained-in*, because the N-Gram-similarity is equal for *contains* and *contained-in* and higher as the prediction for *equal*. We could add a condition to return *disjoint* if this case occurs.

With this additional condition, we achieve the following results. The new confusion matrix is given in Table 7.6 and the precision, recall, and F1-scores are given

| | Class-Label |
|---|---|
| left | Home > Apparel & Accessories > Watches > Watches > Pulsar |
| right | Home > Apparel & Accessories > Watches > Watches > Anne Klein |
| left | Sports & Outdoors > Sports > Golf Equipment > Golf Clothing |
| right | Sports & Outdoors > Sports > Golf Equipment > Golf Shirts |
| left | Electronics > Home Audio > Speakers > Surround Sound Systems |
| right | Electronics > Home Audio > Speakers > Floorstanding Speakers |

Table 7.5: N-Gram: Examples for Disjoint Pairs Labelled as Contained-in.

in Table 7.7 for each label.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 101 | 71 | 43 | 26 |
| c | 32 | 92 | 65 | 68 |
| ci | 30 | 89 | 74 | 48 |
| d | 28 | 257 | 113 | 4647 |

Table 7.6: N-Gram Experiment Confusion Matrix.

| Label | Precision | Recall | F1-score |
|---|---|---|---|
| equal | 0.361 | 0.261 | 0.293 |
| contains | 0.113 | 0.148 | 0.119 |
| contained-in | 0.224 | 0.186 | 0.18 |

Table 7.7: N-Gram Experiment Precision, Recall, and F1-score.

We also ran this setup for the Levenshtein-similarity and the Path-similarity models, but it did not affect the F1-scores for them. We also decided to not use this optimization for the results presented in Chapter 6, because it came as a result of our evaluation. Since this is a special optimization for one single method, it would make the results less comparable than using the generic implementation.

Overall we see that the N-Gram-similarity produces competitive results, especially in predicting equality. Integrating the optimization, we found as part of this Section also improves the result for *contained-in* significantly. We think that the N-Gram-similarity already provides a tough baseline that we can use to evaluate further algorithms and could also assist a human annotator in the task of taxonomy matching.

### 7.1.3 Path Similarity

The path similarity methods diverge from their base similarity functions in the factor that they compare individual categories instead of the full class-label. This enables them to weight the hierarchy levels differently. During the training, we saw that the hyperparameter learned values for lambda, the weight of the lower category vs. the rest, between 0.2 and 0.5, and most of the time settled for 0.3. We did not observe any differences between the Levenshtein and the N-Gram variant. The parameter choice says that the similarity of the categories at the lower level makes up 30 percent of the total similarity, the category pair on the second-lowest level 21 percent, and so on. This factor already indicates that a high similarity of the bottom-categories does not directly translate to overall equality of the class-label pair.

The high weight on the remaining categories should also explain why the artificial edge-cases show up as false positives again. For both, Levenshtein- and N-Gram-Path-similarity, the corner cases make up most of the false positives.

The second thing we notice is that the N-Gram-Path-similarity can not reproduce the good results we recorded for the N-Gram-similarity, especially the variant we presented in this Chapter. We assume that the advantages of the set-based approach that is inherent to the N-Gram-similarity are lost when it is only applied to a single category pair.

To support this hypothesis, we will revisit the first example in Table 7.4. The path-distance version compares ”Mirrorless Cameras” with itself, then compare ”Digital Cameras” to ”Cameras & Camcorders”, and, finally, ”Electronics” to ”Cameras & Camcorders”. On the other hand, the standard N-Gram-similarity would also detect the high overlap between the ”Cameras & Camcorders” categories, because it operates globally.

Overall the path-similarity measure slightly improves the F1-score for *contains* and *contained-in* for the Levenshtein based model. With regards to the N-Gram-similarity, we do not see an advantage in using the path similarity approach. On the contrary, it actually reduces the effectiveness compared to the globally applied N-Gram-similarity.

Out of the baseline methods presented above, only the N-Gram-similarity did surprise us positively. Its set-based approach avoids fallacies inherent to the hierarchical nature of the class-labels under consideration. The Levenshtein- and Path-similarity models have a focus that is to narrow to effectively predict the class-label pairs.

## 7.2 WordNet-Based Matching Methods

### 7.2.1 S-Match

S-Match is one of two models we investigated that use WordNet to find word senses and detect synonyms to improve the taxonomy matching logic. Its performance is among the worst of all models we tested, especially for the *equal* class-label pairs. There are only seven true positives. Since we observe similar low scores for the predictions on our dataset with SCHEMA, the second WordNet-based method, an obvious hypothesis is that WordNet does not cover most words in our corpus. This would explain why they struggle with the given dataset. For S-Match, we use two ways to check this hypothesis. First, we look at the small number of true positives and look for a pattern and, second, we tokenize our corpus and check how many of the distinct words in there yield a result in the WordNet dictionary.

The first thing we observe for the class-label pairs that are *equal* and are predicted as such is that they are in the clothing domain and use simple, common words. Examples are given in Table 7.8 (first part). This is similar for *contains* and *contained-in*, which are shown in the second and third part of Table 7.8, respectively. Surprisingly the last example from Table 7.4 which was misclassified by the Levensthein- and N-Gram-similarity received the correct label by S-Match. The examples we have looked at confirm that S-Match likely handles simple class labels with good coverage in WordNet best.

Next, we checked the WordNet coverage of the gold standard we have used. We split all class-labels into their individual categories and then split words and composite categories. This results in a set of individual words per class-label. We union all of those sets and retain only words with a length greater than two to reduce noise. This results in a set of 1513 different words that make up our complete gold standard. For each of those, we perform a check if at least one synset, i.e., a set of synonyms, is returned by WordNet. Out of the 1513 words, 1261 are contained in WordNet. Hence, about 83 percent of our corpus yields a result. We did also perform this check using the total word count (including repetitions) instead of sets and yield a similar result. Due to the good coverage of WordNet, we have to reject the second hypothesis. The underwhelming results are not caused by a lack of matches in the external corpus.

In summary, we would not recommend the usage of S-Match for taxonomy matching. As Giunchiglia et al. [14] state in their conclusion, matching errors at higher levels, propagate down. The depth of the class-labels in our gold standard may increase the errors S-Match makes. To put that into perspective, we compared our class-labels with the GPC. It uses a maximum of three layers, while most of the class-labels in our gold standard have four or more layers, as we can see in

| | Class-Label |
|---|---|
| left | Clothing & Accessories > Men > Jeans |
| right | Clothing > Mens Clothing > Mens Jeans > Mens Jeans |
| left | Clothing, Shoes & Jewelry > Women > Clothing > Dresses |
| right | Clothing, Shoes & Accessories > Women > Women's Clothing > Dresses |
| left | Sports & Outdoors > Sports & Fitness > Golf |
| right | Sports & Outdoors > Sports > Golf Equipment > Golf Shirts |
| left | Audio > Home Audio > Speakers > In-Wall & In-Ceiling Speakers |
| right | Electronics > Home Audio & Theater > Home Audio > All Home Speakers > In-Wall and In-Ceiling Speakers > In-Ceiling Speakers |
| left | Electronics > Camera & Photo > Digital Cameras > DSLR Cameras |
| right | Cameras & Photo > Digital Cameras |
| left | Baby > Feeding > Bibs & Burp Cloths > Bibs |
| right | Baby > Feeding > Bib & Burp Cloth Sets |

Table 7.8: S-Match: Examples for True Positive Class-Label Pairs.

Figure 4.2.

### 7.2.2 SCHEMA

The results for SCHEMA are similar to the results of S-Match with a slight advantage for SCHEMA with regard to the recall. In contrast to S-Match, which was used as intended with code provided by the authors, we used a subset of the SCHEMA algorithm to make it applicable to the problem at hand. The parts we use should identify possible matches and should be followed by another ranking algorithm to select the best match among them. Hence, we would expect false positives, but almost no false negatives.

Since we have ruled out the influence of a low WordNet coverage in the previous Section, we will focus on the reason for the small number of positive predictions in combination with the fact that about 50 percent of the positive class-label pairs are labelled as *disjoint*.

For the correctly classified *equal* class-labels, we observe that there are only two examples where the lowest categories of the class-label pair do not match perfectly. Those are given in Table 7.9.

Those do not seem to be challenging and are also labelled correctly by all of our baseline methods. The *equal* class-label pairs that are predicted as *disjoint* look

| | Class-Label |
|---|---|
| left | Cell Phones & Accessories > Accessories > Virtual Reality Headsets |
| right | Consumer Electronics > Virtual Reality > Smartphone VR Headsets |
| left | Electronics > Car & Vehicle Electronics > Vehicle Electronics Accessories > Radar Detectors |
| right | Consumer Electronics > Vehicle Electronics & GPS > Radar & Laser Detectors |

Table 7.9: SCHEMA: Examples for True Positive Class-Label Pairs.

in fact very similar, as we can see in Table 7.10.

| | Class-Label |
|---|---|
| left | Electronics > Camera & Photo > Digital Cameras > Mirrorless Cameras |
| right | Cameras & Camcorders > Digital Cameras > Mirrorless Cameras |
| left | Clothing, Shoes & Jewelry > Women > Clothing > Dresses |
| right | Clothing, Shoes & Accessories > Women > Women's Clothing > Dresses |

Table 7.10: SCHEMA: Examples for False Negative Class-Label Pairs.

The problem is that the algorithm expects that the full extended split term set of the left category is a subset of the right one. This rarely happens. If we loosen up this restriction and require only a partial match between the left and the right category, we arrive at the improved results for SCHEMA that we present in Table 7.11 and Table 7.12.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 204 | 5 | 12 | 20 |
| c | 147 | 25 | 16 | 69 |
| ci | 119 | 14 | 7 | 101 |
| d | 1243 | 80 | 75 | 3647 |

Table 7.11: SCHEMA Experiment Confusion Matrix.

With a more optimistic matcher, the F1-score on the *equal* label improves significantly and achieves the best recall for all models. Unfortunately, this reduces the capability of detecting containment even further.

| Label | Precision | Recall | F1-score |
|---|---|---|---|
| equal | 0.075 | 0.507 | 0.128 |
| contains | 0.077 | 0.036 | 0.047 |
| contained-in | 0.033 | 0.018 | 0.023 |

Table 7.12: SCHEMA Experiment Precision, Recall, and F1-score.

In conclusion, we can say that SCHEMA does not handle the semantic taxonomy matching task well. With a slight modification, it may become useful as an input for another classifier, since it is good at predicting if a pair is related at all, without specifying the exact label.

## 7.3 Supervised Taxonomy Matching Methods

### 7.3.1 Ontology Matching with Word Embeddings

The embedding model encodes each class-label with word2vec and uses Cosine similarity to compare the resulting vectors. For such a simple method, it provides surprisingly accurate predictions and also has a low number of false negatives. The large number of false positives is again due to the artificial corner-cases that are mostly classified as either *contains* or *contained-in*.

Our interpretation is that the vectorization produces a similar global view as the set-based N-Gram-similarity algorithm and the mapping into a vector space detects semantic similarity if distinct vocabularies are used, e.g., when comparing ”Kids” and ”Children”.

Overall the embedding approach has one of the lowest numbers of false negatives and could be useful as a first filter to exclude the majority of negative class-label pairs.

### 7.3.2 AdaBoost

Both types of the AdaBoost classifier, the one using the CountVectorizer and the other using word2vec embeddings, achieve comparable results as presented in the previous Chapter. The classifier is based on a combination of decision trees and aggregates the results of the individual models to make a prediction.

We would have expected that the same model, on the same data, with similar results would imply that both variants make similar predictions, but surprisingly they diverge quite heavily. Only 2463 out of 5784 predictions of both models are

the same. Out of those 2463 overlapping predictions, 2185 are for *disjoint* class-label pairs. Even though the models seem very similar, they seldom agree in their prediction.

One drawback of machine learning methods is that we lack the ability to intuitively explain their predictions. Everything between the input class-label pair and the output label is a black box.

We will present some examples for true- and false positives for both variants and try to explain the influence of the embedding method on the prediction. Table 7.13 includes true positives for *equal* class-labels pairs. The first part of the table uses the BoW model and the second part the word2vec embeddings. In Table 7.14 we present false negatives for *equal* class-label pairs and follow the same layout.

| | Class-Label |
|---|---|
| left | Computers & Tablets > Monitors > All Monitors |
| right | Computers/Tablets & Networking > Monitors, Projectors & Accs > Monitors |
| left | Clothing, Shoes & Jewelry > Girls > Watches |
| right | Jewelry > Watches > Kids Watches > Girls Watches |
| left | Electronics > Car & Vehicle Electronics > Car Electronics > Car Audio > Speakers |
| right | Consumer Electronics > Vehicle Electronics & GPS > Car Audio > Car Speakers & Speaker Systems |
| left | Electronics > Car & Vehicle Electronics > Vehicle Electronics Accessories > Radar Detectors |
| right | Consumer Electronics > Vehicle Electronics & GPS > Radar & Laser Detectors |
| left | Electronics > Camera & Photo > Digital Cameras > Mirrorless Cameras |
| right | Cameras & Camcorders > Digital Cameras > Mirrorless Cameras |

Table 7.13: AdaBoost: Examples for True Positive Class-Label Pairs.

The third example in Table 7.13 is predicted correctly by both variants, but all other examples did receive contradicting labels by AdaBoost. Given the examples presented here and additional examples from our experiments, we could not find any explanation on the different true positive predictions. The same holds true for the examples in Table 7.14. Except for the second example from the top, both variants disagree on the predicted label. Also, we could not find a reasonable

| | Class-Label |
|---|---|
| left | Car Electronics & GPS > Car Audio > Car Speakers |
| right | Auto & Tires > Auto Electronics > Car Speakers > Car Speakers |
| left | Home > Apparel & Accessories > Watches > Watches > Tissot |
| right | Jewelry > Watches > Luxury Watches > Tissot Watches |
| left | Electronics > Headphones |
| right | Audio > Headphones > All Headphones |
| left | Clothing, Shoes & Jewelry > Girls > Watches |
| right | Jewelry > Watches > Kids Watches > Girls Watches |
| left | Home & Kitchen > Kitchen & Dining > Kitchen & Table Linens > Kitchen Rugs |
| right | Home > Decor > Rugs > Kitchen Rugs |
| left | Electronics > Portable Audio & Video > Portable DVD Players |
| right | TV & Home Theater > Blu-ray & DVD Players > Portable DVD Players |

Table 7.14: AdaBoost: Examples for False Negative Class-Label Pairs.

explanation why those apparently obvious matches are labelled as *disjoint* by the respective algorithm.

All-in-all AdaBoost gives mediocre results that are hard to explain. It behaves very differently depending on the encoding of the class-label pairs. We would not recommend the usage of AdaBoost in either variant for taxonomy matching, because it delivers predictions that are hard to interpret without compensating this with outstanding results.

### 7.3.3 Naive Bayes

Naive Bayes is a simple machine learning algorithm that is frequently used as a baseline whenever classification is the objective. Scikit-learn also recommends to try Naive Bayes in combination with the CountVectorizer[1]. It also has the advantage that it does not require a complex hyperparameter search, because it simply looks at the given examples to infer a posterior distribution to make predictions on previously unseen data.

We do not see any obvious problems in the confusion matrix and the precision, recall, and F1-scores of Naive Bayes. There are about twice as many *equal* labelled false positives than for the *contains* and *contained-in* labels, so we will take a look

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html`. Accessed: 01.05.2020

at some examples for those three cases and try to derive some conclusions. In the same manner, we will look at false negatives.

Examples of false positives are given in Table 7.15. Some errors stem from the

| | Class-Label |
|---|---|
| left | Electronics > Computers & Accessories > Tablet Accessories > Bags, Cases & Sleeves > Cases |
| right | Jewelry > Watches > Designer Watches > Guess Watches |
| left | Electronics > Computers & Accessories > Data Storage > Internal Solid State Drives |
| right | Electronics > TV & Video > TV Accessories > TV Antennas |
| left | Electronics > Home Audio > Speakers > Floorstanding Speakers |
| right | Electronics > Home Audio > Speakers > Outdoor Speakers |

Table 7.15: Naive Bayes: Examples for False Positive Class-Label Pairs.

generated corner-cases, but Naive Bayes is better than the string-based baselines methods in labelling those correctly. The other group of errors seems completely arbitrary. One reason for this behavior may be the independence assumption in Naive Bayes. It states that all input variables are expected to be conditionally independent. In our case, it may be possible that Naive Bayes only learns that a specific word in the left or right class-label indicates equality and another word containment, and so on. Therefore, Naive Bayes never takes the relationship between the two class-labels into account. Following this assumption, it would output a combination of the likelihood that the left and the right label usually have a specific relationship with other labels.

We observe a similar behavior for the false negatives that we present in Table 7.16.

| | Class-Label |
|---|---|
| left | Sports & Outdoors > Sports & Fitness > Golf |
| right | Sports & Outdoors > Sports > Golf Equipment > Golf Shirts |
| left | Home, Furniture & Office > Furniture & Decor > Clocks |
| right | Home > Decor > Clocks |

Table 7.16: Naive Bayes: Examples for False Negative Class-Label Pairs.

In addition to the drawback mentioned above, it is impossible to reuse the existing model in another domain, e.g., Food, with the closed vocabulary that we use

for training. Transferring it would require an additional set of labelled training data in the new domain.

Overall we see good results for Naive Bayes on the given training set, but we identified two possible limitations for any real-world usage. The closed vocabulary prohibits an application in a new domain without additional training data, and the independence assumption indicates that the likelihood is based on individual keywords instead of the relation between the two class-labels.

### 7.3.4 Stochastic Gradient Descent

The SGDClassifier uses an SVM model and trains it with SGD. In contrast to the AdaBoost models, the word2vec embedding enables substantially better predictions compared to the results based on the CountVectorizer. According to the documentation of the SGDClassifier, it supports sparse (CountVectorizer) and dense (word2vec) vectors as inputs. Hence, both types of encodings are supported equally well. The other differentiating factor between the two encodings is the vector dimension. While the word2vec model uses 300 dimensions, the CountVectorizer produces a sparse vector with the vocabulary size, i.e., more than 2000 dimensions.

Since SVM classifiers try to find an optimal hyperplane to separate the training examples, the algorithm may perform worse on higher dimensional vectors, since it becomes harder to find a suitable decision boundary. We may improve the CountVectorizer results with additional training examples, but we expect that the benefit will be small.

The word2vec approach has the additional advantage of a broader vocabulary. We could use the given model to make predictions in any domain that is covered by the Googe-News dataset that was used to train the word2vec model. Hence, we will focus on the word2vec based predictions in the remainder of this Subsection.

Since we also observe a high number of false positives for the word2vec based model, we would assume that this is due to the artificial corner cases. Contrary to our expectation, the majority of false positives are obviously *disjoint*, e.g., with one class-label being in the ”Electronics” and the other being in the ”Clothing” domain.

Following up on our conclusions from the Levenshtein- and N-Gram-similarity, which had problems with artificial edge-cases, we would like to try a two-phased approach for taxonomy matching. A string-based method with a low threshold could be used to filter the obviously *disjoint* pairs. Since the SGD model performs well on actual positive labels and artificial edge-cases, we would expect better results than with one of the standalone models.

To verify this hypothesis, we replaced all SGD predictions with *disjoint* if the Levenshtein-similarity did not exceed 0.3. We also tested 0.2 and 0.4 as thresholds

and found that 0.3 gave the best average F1-score. The prediction, recall, and F1-score are given in Table 7.17 and the confusion matrix in Table 7.18.

| Label | Precision | Recall | F1-score |
|---|---|---|---|
| equal | 0.081 | 0.208 | 0.112 |
| contains | 0.051 | 0.094 | 0.061 |
| contained-in | 0.126 | 0.212 | 0.156 |

Table 7.17: SGD Experiment Precision, Recall, and F1-score.

| | e | c | ci | d |
|---|---|---|---|---|
| e | 83 | 23 | 35 | 100 |
| c | 35 | 38 | 20 | 164 |
| ci | 39 | 10 | 76 | 116 |
| d | 437 | 379 | 278 | 3951 |

Table 7.18: SGD Experiment Confusion Matrix.

While the number of false positives decreases significantly, we also lose some true positives. Adding this restriction, therefore, increases overall prediction accuracy, but results in a slight decline of recall.

The word2vec based SGD model successfully predicts most of our artificial edge cases correctly and outperforms the AdaBoost model. It could be used on its own or be combined with the Levenshtein-similarity to trade-off recall for additional accuracy.

### 7.3.5 Multi-Layer Perceptron

As we have stated in the previous Chapter, the Neural Network had convergence issues, which are probably due to an insufficient amount of training data. We also tried smaller networks, but the issue persisted. In addition, we trained the model on the full, noisy dataset that we extracted from the Semantic Web and labelled based on shared instances, but this did not improve the results. We also saw that the instance-based approach produced multiple misclassifications among the positive labels, especially for class-label that contain many products on one e-commerce platform, but less on another. Hence, the only way to improve this model is to provide more manually annotated data.

On the other hand, we would be curious about how the given model would perform in a new domain, e.g., Food, without training it on examples in this domain.

Unfortunately, we lack a dataset for evaluating those. Using either the Clothing or the Electronics data in our corpus for training the model and testing it on the other data would not give reliable results for this experiment, because the amount of training data would be far too small.

Overall the neural network makes good predictions given the training data, but we would like to explore in the future how it performs in a new domain and if additional data would lead to significant improvements of the predictions.

## 7.4 Summary

In this Chapter we analyzed the errors that the tested models make and gave examples for cases that could be considered easy and for cases where the models struggle to make the correct prediction. At the end of each Subsection, we gave a recommendation for suitable use cases of the given model.

Overall the N-Gram-similarity and the word2vec embeddings with Cosine similarity have a good recall, and a comparatively good F1-score. They would certainly be helpful as an input for a human annotator, but are not precise enough for a standalone application.

The Neural Network did provide a good overall accuracy, but would benefit from additional training data. It would also be interesting to check if the embedding models (AdaBoost, SGD, and MLP) generalize to new domains without additional training, but we leave this for future work.

In our interpretation, we were also limited by the lack of explainable predictions for machine learning models. In the current form, they can only serve as a black box. It would aid the interpretability of the predictions if there were tools available that explain the internal behavior of a given model.

# Chapter 8

# Summary and Future Work

In this Chapter, we will review the main contributions and results of this Thesis. Afterwards, we will look at possible future extensions of the work presented here.

In Chapter 2 we gave a general introduction into the Semantic Web and the reasons behind its creation. We showed how annotations in the HTML-code can ensure that content that was created for humans becomes readable for machines. We also introduced the ideas behind ontology matching and, a special case of ontology matching, taxonomy matching. Taxonomy matching presents itself as especially useful in an e-commerce environments to enhance catalog integration tasks and product searches.

Relevant contributions in the field of the Semantic Web, ontology matching, and product taxonomy matching are presented in Chapter 3. In addition to a description of the relevant methods, we designed a categorization system for product taxonomy matching algorithms in which we integrated the methods from Chapter 3 and the ones used throughout this Thesis.

To compare different methods for product taxonomy matching, we created a training dataset from product information that was crawled from the Semantic Web. We found that labels derived from shared instances have an overall low quality and are insufficient for reliable testing and, therefore, we manually annotated a subset of this training set to create a product taxonomy matching gold standard. The result is a set of class-label pairs from different e-commerce platforms that are labelled as either *equal*, *contains*, *contained-in*, or *disjoint*.

We described the methods that we test on this gold standard in Chapter 5. This includes three major types of methods. The first group consists of static methods that only use the information given in the two class-labels that should be labelled. The algorithms in the second group are also static in the sense that they do not benefit from training data, but they include external corpora like WordNet to enhance

their results. The last group consists of machine learning models that are trained on our training set and use the two class-labels as an input to make their predictions.

Chapter 6 dealt with the results of our experiments. For each method, we provided the precision, recall, and F1-score per positive label and the corresponding confusion matrix.

In Chapter 7 we analysed those results and tried to identify explanations for the phenomena we observed in the experiments. In doing so, we presented individual examples for class-label pairs that are either hard or easy to predict for a given algorithm and provide recommendations for cases where a certain algorithm may prove useful. Overall, we noticed that product taxonomy matching is a tough problem and no method provides fully satisfying results. Especially the added complexity of more advanced models is often not justified by a significant improvement over the baseline methods.

The implementations we have used in order to replicate the results of relevant contributions on a real-world dataset follow the description of the authors as close as possible and we used open source packages where available. Nevertheless, we had to add small adjustments to cater to our requirement of providing semantic annotations, i.e., detection of equality or if one class is more general than another. Only our machine learning models and S-Match [14] provide this functionality out of the box. However, those adjustments do not explain the overall bad performance of the models under consideration.

Thus, as a part of this Thesis, we have identified two fields that deserve additional attention.

First, all models across the literature require that the two taxonomies under consideration use similar concepts to label their products. We agree that this is a necessary requirement to detect equality or another semantic relationship, but we also found classes that share a set of products, but use orthogonal classification methods. An example may be a classification either by sport (Sports $>$ Golf $>$ Polo Shirts) or by brand (Sports $>$ Under Armour $>$ Polo Shirts). The two class-labels would certainly share some products, but do not fit any of the conventional semantic labels. Those partial overlaps can not be handled by any model that we are aware of. We detected those in our dataset based on shared instances, but it would be interesting to classify them based on the class-label alone.

The second area is the semantic labelling of class-label pairs that we also covered in this Thesis. Most existing methods focus on finding the best match for a given class-label in a target taxonomy. While this certainly has a use case for catalog integration, the more generic solution could also enhance product search tasks and possibly more topics.

In conclusion, we have seen that product taxonomy matching is a relevant problem with room for improvement. On the one hand, we provided insights into the

specific problem one faces with product taxonomies and, on the other hand, assist further research with the gold standard we created.

# Bibliography


[1] Scrapy. `https://scrapy.org`. Accessed: 01.05.2020.

[2] Steven S Aanen, Lennart J Nederstigt, Damir Vandić, and Flavius Frăsincar. Schema-an algorithm for automated product taxonomy mapping in e-commerce. In *Extended Semantic Web Conference*, pages 300–314. Springer, 2012.

[3] Steven S Aanen, Damir Vandic, and Flavius Frasincar. Automated product taxonomy mapping in an e-commerce environment. *Expert Systems with Applications*, 42(3):1298–1313, 2015.

[4] Rakesh Agrawal and Ramakrishnan Srikant. On integrating catalogs. In *Proceedings of the 10th international conference on World Wide Web*, pages 603–612. ACM, 2001.

[5] Zharko Aleksovski, Michel Klein, Warner Ten Kate, and Frank Van Harmelen. Matching unstructured vocabularies using a background ontology. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 182–197. Springer, 2006.

[6] Heiko Angermann and Naeem Ramzan. *Taxonomy Matching Using Background Knowledge*. Springer.

[7] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908. Acm, 2005.

[8] Paolo Avesani, Fausto Giunchiglia, and Mikalai Yatskevich. A large scale taxonomy mapping evaluation. In *International Semantic Web Conference*, pages 67–81. Springer, 2005.

[9] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[10] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[11] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[12] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In *Handbook on ontologies*, pages 385–403. Springer, 2004.

[13] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 2 edition, 2013.

[14] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. Semantic schema matching. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 347–365. Springer, 2005.

[15] Robert Isele, Jürgen Umbrich, Chris Bizer, and Andreas Harth. LDSpider: An open-source crawling framework for the web of linked data. In *Proceedings of 9th International Semantic Web Conference (ISWC 2010) Posters and Demos*, 2010.

[16] Aidan Hogan José-Miguel Herrera and Tobias Käfer. Btc-2019: The 2019 billion triple challenge dataset. In *Proceedings of the 18th International Semantic Web Conference (ISWC)*. Auckland, New Zealand, 2019.

[17] Prodromos Kolyvakis, Alexandros Kalousis, and Dimitris Kiritsis. Deepalignment: Unsupervised ontology matching with refined word vectors. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 787–798, 2018.

[18] James A. Larson, Shamkant B. Navathe, and Ramez Elmasri. A theory of attributed equivalence in databases with application to schema integration. *IEEE Transactions on software engineering*, 15(4):449–463, 1989.

[19] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26, 1986.

[20] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[21] Robert Meusel, Petar Petrovski, and Christian Bizer. The webdatacommons microdata, rdfa and microformat dataset series. In *International Semantic Web Conference*, pages 277–292. Springer, 2014.

[22] Robert Meusel, Anna Primpeli, Christian Meilicke, Heiko Paulheim, and Christian Bizer. Exploiting microdata annotations to consistently categorize product offers at web scale. In *International Conference on Electronic Commerce and Web Technologies*, pages 83–99. Springer, 2015.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[24] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[25] Lennart Nederstigt, Damir Vandic, and Flavius Frasincar. A lexical approach for taxonomy mapping. *Journal of Web Engineering*, 15(1-2):084–109, 2016.

[26] Natalya F Noy and Mark A Musen. The prompt suite: interactive tools for ontology merging and mapping. *International journal of human-computer studies*, 59(6):983–1024, 2003.

[27] Panagiotis Papadimitriou, Panayiotis Tsaparas, Ariel Fuxman, and Lise Getoor. Taci: Taxonomy-aware catalog integration. *IEEE Transactions on Knowledge and Data Engineering*, 25(7):1643–1655, 2012.

[28] Sangun Park and Wooju Kim. Ontology mapping between heterogeneous product taxonomies in an electronic commerce environment. *International Journal of Electronic Commerce*, 12(2):69–87, 2007.

[29] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[30] Petar Petrovski, Anna Primpeli, Robert Meusel, and Christian Bizer. The wdc gold standards for product feature extraction and product matching. In

*International Conference on Electronic Commerce and Web Technologies*, pages 73–86. Springer, 2016.

[31] Jan Portisch and Heiko Paulheim. Alod2vec matcher. *OM@ ISWC*, 2288:132–137, 2018.

[32] Anna Primpeli, Ralph Peeters, and Christian Bizer. The wdc training dataset and gold standard for large-scale product matching. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 381–386, 2019.

[33] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[34] Marta Sabou, Mathieu d'Aquin, and Enrico Motta. Exploring the semantic web as background knowledge for ontology matching. In *Journal on data semantics XI*, pages 156–190. Springer, 2008.

[35] Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, 2017.

[36] Damir Vandic, Jan-Willem Van Dam, and Flavius Frasincar. Faceted product search powered by the semantic web. *Decision Support Systems*, 53(3):425–437, 2012.

[37] Yuanzhe Zhang, Xuepeng Wang, Siwei Lai, Shizhu He, Kang Liu, Jun Zhao, and Xueqiang Lv. Ontology matching with word embeddings. In *Chinese computational linguistics and natural language processing based on naturally annotated big data*, pages 34–45. Springer, 2014.

[38] Ziqi Zhang and Monica Paramita. Product classification using microdata annotations. In *International Semantic Web Conference*, pages 716–732. Springer, 2019.

## Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Master-/Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 03.05.2020 Unterschrift