UNIVERSITÀ DEGLI STUDI ROMA TRE

DIPARTIMENTO DI INGEGNERIA
Scuola Dottorale in in Ingegneria Informatica

XXXII Ciclo

Instance-level attribute alignment for heterogeneous product sources

Dottorando

Federico Piai

Relatore

Prof. Paolo Merialdo

Gruppo di Big Data and Databases

April 2020

Acknowledgements

Vorrei ringraziare innanzitutto la mia famiglia, che ha costruito gran parte di me, la mia sicurezza sempre e l'ultimo baluardo nei momenti più tristi quando ho l'impressione di essere solo al mondo.

Mamma per tutto quello che ha fatto per me e per il grande affetto che mi ha sempre dato. Papà, per tutto quello che mi ha insegnato e per la curiosità che mi ha infuso. Daniele, per tutti i momenti passati insieme, le nostre grandi chiacchierate, riuscire a capirci sempre nei nostri voli pindarici, l'impressione di essere i nostri più grandi amici.

Annamaria, che mi sta sempre accanto nonostante le mie stranezze, con la sua gentilezza sensibilità ed entusiasmo, per tutti i momenti insieme e per tutti i progetti che affronteremo e le nostre imperfezioni con cui ci confronteremo.

Ringrazio Paolo il mio supervisore che mi ha seguito in tutti questi anni, che ha saputo creare un gruppo coeso e competente, gestire tanti progetti e cogliere ogni opportunità per farci crescere professionalmente.

Un grazie a Donatella per la sua disponibilità e capacità di ascoltare e capire i dubbi e dare suggerimenti sempre efficaci.

Grazie a Divesh e Paolo Atzeni, senza le loro idee e competenze e la capacità di comunicarle in maniera semplice questo percorso non sarebbe stato lo stesso.

Grazie a tutti i membri del Minilab, che sono stati un'ancora di salvezza per me. A livello professionale riuscendo a "sbloccarmi" con il loro lavoro. Ma anche a livello personale: in un momento molto difficile di questo percorso, trovare tante persone con cui confrontarmi, con cui confidarmi e con cui passare anche tanti momenti spensierati insieme è stato molto bello, e ha creato anche dei bei legami che dureranno oltre questa

avventura. Grazie ad Andrea De Angelis per la sua pazienza, la sua concretezza e le belle chiaccherate che ci siamo fatti, grazie a Vincenzo che stimo molto e con cui abbiamo scambiato tante idee, grazie a Maurizio, Alessio, Jerin, Antonio, Vincenzo Martello e tutti quanti.

Grazie e Valerio, Andrea Rossi, Elena, Matteo Cannaviccio, Antonio Maccioni, Tommaso, Eleonora ed Heba che hanno condiviso con me questo percorso come colleghi e mi hanno sempre aiutato quando ne avevo bisogno.

Grazie all'"avvocato" Matteo Amadei, che c'è sempre stato anche se non abbiamo condiviso tutto il percorso insieme, una sicurezza con la sua allegria e serenità e con le sue storie Instagram!

Grazie a tutti gli amici che ci sono sempre stati in questo periodo, a Lorenzo e Alice con la loro spontaneità contagiosa con cui ho condiviso un bel periodo della mia vita, a Silvia e Leonardo per tutti i momenti insieme e le nostre confidenze reciproche, a Francesca che mi mancherà, a Giulio, Davide, Andrea, Livio, Marco, Vins e tutti i "Burini", ad Alessandro Quaglio, Alessandro Marani, Luca e Michele Principi e Salvatore, a cui vorrei dire che anche se le strade ci portano a vederci meno spesso, rimangono sempre dei pilastri per me.

Abstract

This thesis focuses on Big Data integration, a foundational area in data management research. We describe in particular the integration of product specifications from multiple sources of data, with the final goal of building a complete and reliable product graph.

Exploiting multiple data sources has the advantage to provide information about rare and niche products and uncommon properties, and having enough redundancy to solve potential conflicts. On the other hand, it involves several challenges due to the heterogeneity of Web sources.

We described a complete pipeline for product data integration, involving Web extraction and integration steps, which, unlike traditional approaches, performs the record linkage step (group specifications by product) before attribute alignment step (group attributes with equivalent semantics and define mappings). Indeed, record linkage in product context is simplified by the presence of general product identifiers, while attribute alignment is a very complex task due to presence of a lot of properties about a product, some rarer and some more common, with many different representations. We provided an extensive analysis of the state of the art on these two tasks.

We formulated a novel problem of computing attribute alignment at the instance level. Traditional schema-level alignment methods, which critically rely on local homogeneity within a source, are unable to effectively solve this problem due to the significant heterogeneity exhibited by product specifications, both across and within sources. We take advantage of the opportunities arising from the richness and redundancy of information across sources, and propose an iterative solution, called RaF-AIA, that consists

of three key steps: (*i*) First, it uses a Bayesian model to analyze overlapping information across sources to match the most locally homogeneous attributes; (*ii*) Second, inspired by NLP techniques, it uses a tagging approach to create (virtual) homogeneous attributes from tagged portions of heterogeneous attribute values; (*iii*) Third, it makes creative use of classical alignment techniques based on matching of attribute names and domains.

We developed a publicly available benchmark (Alaska Benchmark) for the tasks of attribute alignment and record linkage, which we also used to run experiments for evaluating the RaF-AiA approach, demonstrating its effectiveness and efficiency, and its superiority over alternative approaches adapted from the literature.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data integration is a foundational area in database research related to the existence of multiple sources of data [1, 2]. It has various facets, in terms of goals (for example, providing a unified access to multiple sources or building new "integrated" repositories with data coming from several sources), in terms of problems (for example, source selection, schema alignment, entity resolution and data fusion), and in terms of challenges (for example, dealing with various forms of heterogeneity and with issues in the quality of data and schema). Many interesting contributions have been made over the years, but the a lot of challenging issues still need to be solved. The recent, significant interest in Big Data has brought new issues into this arena, related not only to the size of data sources, but also to other aspects, such as their number, heterogeneity and dynamicity, each of which adds significant difficulties and complexity [3].

1.1 Integrating Product Specifications from Multiple Web Sources

In Big Data integration, even within a single domain, there are many Web sources with many different characteristics and specific challenges, which has led to significant research focusing on specific or families (e.g., products or sports), or categories (e.g., camera, or football) thereof [4, 5]. This focus enables meaningful solutions to important

practical problems and to develop methods that could be useful in related domains as well.

A major application area in this respect is that of eCommerce, where retailers are interested in providing detailed and trustable information about a diversity of products, and this information usually comes from multiple sources and, to a greater extent, Web sites from which data is extracted, with the goal of creating comprehensive catalogues [6, 7, 8, 9].

However, despite all the work on data integration, integrating product specifications from multiple sources is still not a solved problem and raises novel and intriguing research challenges [10, 7] and therefore deserves specific attention.

Given the large and increasing number of sources that provide data about product specifications and the velocity as well as the variety with which such data are available, this domain represents a challenging scenario for developing and evaluating big data integration solutions.

Integrating the data offered by pages describing product specifications to create a comprehensive, unified description of each product represents a fundamental step to enable valuable applications, such as, question answering, price comparison and data driven market analysis.

To address the issue of integrating product specification from Web sources, we extend the general approach for big data integration defined in [3] to an end-to-end pipeline that decomposes the problem into different tasks from source and data discovery, to extraction, data linkage, attribute alignment and data fusion [10].

1.1.1 End-to-End Data Integration Pipeline

Our end-to-end data integration pipeline for product specification pages is depicted in Figure 1.1. The pipeline is presented as a sequence of tasks, but different configurations can be defined depending on the application goals. Every task adopts methods and techniques from databases, information retrieval and machine learning. Every task produces output to feed the successive task in the pipeline, but intermediate results

could find other compelling application scenarios as well. To this end, we advocate that the pipeline must include an empirical evaluation benchmark for every task.

In our vision, the information need is expressed by an input set of sample pages. We observe that products are typically organized in *categories*, and hence we expect that the sample pages refer to products from the categories of interest. Our approach is inspired by the Open Information Extraction [11] paradigm: the schema for the target data is not specified in advance, and the categories of the target products do not refer to a predefined product taxonomy, but they are rather inferred from data in the product pages of the input sample and in the product pages that are gathered from the Web along a progressive and iterative process.

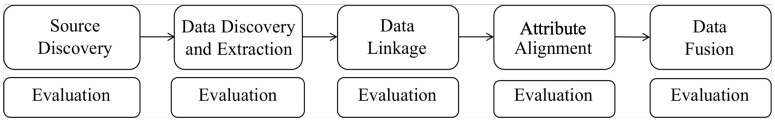

Figure 1.1: End-to-end big data integration pipeline for product specifications.

Source discovery aims at efficiently finding and crawling product websites in order to gather pages that refer to the products of interest. One might believe that discovering product websites is a minor task, as a relatively small number of *head sources* can offer enough data for most of the products. For example, amazon.com already provides data about an impressive number of products. However, valuable information is actually published by an enormous number of *tail sources* [4, 12], i.e., sources that each provide a small number of product entities. These tail sources are important because they improve coverage. They can often offer *tail entities*, i.e., products that are present in a small number of sources, as well as *tail attributes*, i.e., product properties that are present in a small number of entities. Also, tail sources ofter refer to *tail categories*, i.e., small niche categories of products. Finally, *tail sources* contribute to information diversity, as they provide values that depend on the local source, such as, product reviews and price. Source discovery also deals with efficiently crawling the discovered websites towards pages containing products of the categories of interest, without visiting unproductive regions.

Data discovery and extraction has the objective of processing the pages harvested in the previous task in order to locate and extract product attribute names and their values. As we mentioned above, we do not rely on a predefined schema, but rather extract attributes bottom-up, with the goal of discovering not just *head attributes*, but also *tail attributes* that cannot always be described in advance.

Data linkage seeks to cluster pages from different sources that refer to the same products. It is worth observing that in the traditional data integration pipeline, schema alignment is performed before record linkage [13]. Unfortunately, with a very large number of sources, such a traditional approach becomes infeasible because of the huge variety and heterogeneity among attributes. As we shall discuss later, we propose to perform data linkage before schema alignment as we can take advantage of the opportunity that products are named entities, and hence a product specification page usually publishes the product identifier.

Attribute alignment addresses the challenge of semantic ambiguity and aims to reconcile the attributes offered by different sources, that is, to understand which attributes have the same meaning and which ones do not, as well as identify value transformations to normalize different representations of the same attribute values. Since we do not rely on a global schema given in advance, correspondences among attributes are established bottom-up leveraging the results of the previous data extraction and data linkage phases.

Data fusion tackles the issue of reconciling conflicting values that may occur for attributes from different sources. Data fusion aims at evaluating the trustworthiness of data, deciding the true value for each data item, and the accuracy of the sources. To address these challenges, data fusion techniques rely on data redundancy, which further motivates the need to process many sources.

For simplicity of presentation, we have described our approach as a linear pipeline, where tasks are performed in sequence and independent of one another. However, there

might be feedback loops between the tasks, as intermediate results can indeed influence the performance and the behavior of the preceding tasks and of the end to end solution.

1.1.2 Challenges and Opportunities

The web scale raises intriguing *challenges* for all the tasks of our pipeline due to its volume, variety, velocity, and veracity [13].

- The *volume* of data refers not only to the large number of products but, more importantly, to the number of sources. As we have discussed, to achieve coverage and diversity, we need to process a very large number of sources across the entire web, not just a small number of pre-selected sources.
- The *variety* of data is directly influenced by the number of sources and arises at many different levels, affecting every task of our pipeline. At the product category level, websites, especially the head ones, organize products according to such a vast plethora of categorization strategies that makes it very difficult, if not impossible, to reconcile them into a unified taxonomy. At the product description level, heterogeneities are related to attributes and values, which are published according to different granularities (e.g., physical dimensions in one field vs three separate fields for width, length, height), formats (e.g., centimeters vs inches) and representations (e.g., the color of a product as a feature vs distinct products for different colors).
- The *Velocity* of data involves the rate of appearance and disappearance of pages in sources as well as the rate of appearance and disappearance of web sources. Also, while some attributes (such as technical and physical features) are quite stable over time, the contents of the individual pages can change daily, for example for prices and reviews.
- The *Veracity* of data deals with honest mistakes that can occur in web pages, but also with deceits, that is, deliberate attempts to confuse or cheat (e.g., providing imprecise or erroneous product characteristics).

Our approach to address these challenges aims at taking advantage of the *opportunity* that products are named entities, and hence a product specification page usually publishes the product identifier. Web sources that deliver product specification pages publish product identifiers mainly for economic reasons: websites need to expose the product identifiers to let them be indexed by shopping agents and available to customers who search products for comparing prices or consulting specifications. Large e-commerce marketplaces strongly encourage sellers and retailers to publish product identifiers,[1] as they improve efficiency both for the internal management of data and for the exchange of data with search engines like Google and Bing.

The presence of identifiers allows us to drive the pipeline from source discovery to data integration by leveraging the opportunity of *redundancy of information at the global level*, and the *homogeneity of information at the local level*.

- At the global level, we observe that head (popular) products are present in several head (large) sources as well as in many tail (small) sources. Therefore, we expect that identifiers of head products are spread across many sources. Further, many head products in a category will often co-occur in multiple sources.

- At the local level, we observe that the structure of information, within each source, is more regular. Hence, we expect the product specification presented in a given page is published according to the same structure for every page in the same source.

Redundancy as a Friend Figure 1.2 illustrates the key intuitions underlying our approach [10] from source discovery to data integration to meet the goal of *effectively* and *efficiently* dealing with head and tail sources, hence including all pages of head and tail entities. Starting from known head entities in head sources, we take advantage of homogeneity of information at the local level to extract product specifications and identifiers for tail entities in head sources (even head sources offer many tail entities).

[1] eBay, Amazon, Google Shop explicitly require sellers to publish the id for many product categories. For example, see eBay's rules:
http://for-business.ebay.com/product-identifiers-what-they-are-and-why-they-are-important.

Then, we exploit the presence of head entities across sources: searching head identifiers, we discover tail sources (even tail sources offer a few head entities). Again, we exploit homogeneity of information at the local level to extract identifiers and specifications for tail entities in tail sources.

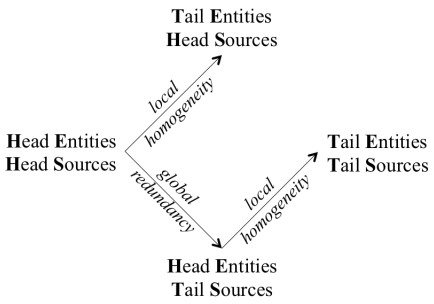

Figure 1.2: Our approach.

1.2 Contributions

This thesis has been developed in framework of the above pipeline. In particular we concentrated our efforts to study solutions for the *Attribute Alignment* task. As we detail in the following, this is a challenging task: unfortunately the local homogeneity of the sources mainly refers to the physical structures of the pages, while the heterogeneity of data occurs not only across sources, but even within sources. To tackle to problem, we have therefore developed techniques to perform *attribute alignment at the instance level*, in contrast to traditional approaches that work at the schema level.

Another important contribution of our work has been the development of a *benchmark* for big data integration. As we observed that many traditional solutions for data integration does not apply on multiple Web sources, we built a benchmark aimed to drive research in this field.

Instance Level Attribute Alignment Despite all the work on data integration, integrating product specifications from multiple sources is challenging and raises novel and intriguing research problems and therefore deserves specific attention. Data ex-

tracted from Web sources (or even coming from some existing databases) can be seen as organized in sets of key-value pairs (or, in other terms, in sparse, heterogeneous tables), and a major issue is the reconciliation of attributes (that is, attributes, in database terms), usually called attribute alignment in this framework.

Traditional approaches to attribute alignment consider heterogeneity across sources, but they rely on *local homogeneity*; that is, they assume that each source adopts homogeneous semantics and homogeneous representations of data [14, 15]. In the product domain, because of volume and variety of the product specifications, the local homogeneity assumption is not valid, and sources exhibit many forms of instance and schema level heterogeneity across sources as well as intra-source, i.e. within the same source.

We follow the data integration pipeline proposed in [12], which performs record linkage before attribute alignment. The rationale is that while attribute alignment is unfeasible because of the degree of heterogeneity of the sources, record linkage can be computed by exploiting specific properties of the domain, such as the presence of product identifiers [12], and the linkage among the instances can be exploited to

Our proposal addresses the issue of attribute alignment in such a challenging and intriguing setting. Working with data from heterogeneous sources that violate the local homogeneity assumption requires to overcome traditional solutions, which aim at aligning attributes at the schema level. We propose a solution to compute attribute alignment at the instance level: our goal is to group individual keys that are associated with values describing the same product property.

According to our end-to-end data integration pipeline, we rely on record linkage information. The rationale is that while attribute alignment is unfeasible because of the degree of heterogeneity of the sources, record linkage can be computed by exploiting specific properties of the domain, such as the presence of product identifiers, and the linkage among the instances can be exploited to align the attributes.

Indeed our approach leverages the opportunities offered by the redundancy of information across sources. We propose an iterative process that first, based on a Bayesian model analyses the overlapping information across sources to align the most homo-

geneous attributes. Then, adopting a tagging approach, inspired to NLP techniques, identifies and aligns attributes that suffer local and global heterogeneity.

We have described our approach in a paper which is currently under submission [16].

Building a Benchmark for Big Data Integration In an experimental evaluation performed between Sept 2014 and Feb 2015, the Dexter focused crawler [17] was trained to gather product pages from 10 coarse categories: camera, cutlery, headphone, monitor, notebook, shoes, software, sunglasses, toilet accessories, televisions. The crawler discovered 3.5k websites, for a total of 1.9M pages. Each website contributed to provide pages for the different categories, and pages were grouped into $7,145$ clusters, corresponding to the local categories exposed by the websites (on average every websites has 2 local categories). [2] We compared the contents of our dataset with pages in Common Crawl,[3] an open repository of web crawl data. About 68% of the sources discovered by Dexter were not present in Common Crawl. Only 20% of our sources contained fewer pages than the same sources in Common Crawl, and a very small fraction of the pages in these sources were product pages: on a sample set of 12 websites where Common Crawl presented more pages than in our dataset, we evaluated that only 0.8% of the pages were product pages.

These results suggested us the critical need for the community to build a suitable benchmark product dataset to conduct big data research. To this end our experience gave raise to the Alaska benchmark, a big data integration benchmark, which has been used in the DI2KG workshop at ACM KDD 2019. We are currently preparing a publication that describes the benchmark [18].

1.3 Roadmap

The thesis is organized as follows. Chapter 2 illustrates the main component of the big data integration pipeline for product specifications. Chapter 3 discusses related work

[2] The dataset is publicly available on-line at https://github.com/disheng/DEXTER, and represents an extension of the dataset presented in [17].

[3] http://commoncrawl.org/

and state-of-the-art for attribute alignment. Chapter 4 describes the solution that we have designed to address the instance level attribute alignment issue. Chapter 5 presents the dataset and the ground truth that we have built to evaluate our techniques. The results of this activity has given rise to the Alaska benchmark for big data integration. Chapter 6 presents the results of the experimental evaluation that we have conducted for our instance level attribute alignment technique on an enhanced version of the dataset of the Alaska benchmark. The thesis concludes with a final discussion on the work and with a description of future work.

Chapter 2

The Product Specifications Integration Pipeline

This Chapter provides a deeper overview of the main components of the pipeline introduced in the previous chapter for addressing the issues of integrating product specifications from Web sources. Several components were developed in previous work, but we have re-engineered them for interoperability and effectiveness purposes.

2.1 Source Discovery

We consider a focused crawler, Dexter [17], to discover and crawl product websites offering product pages for the input categories. The crawler iterates over three phases: *source finding*, *source crawling*, *identifier extraction*.

2.1.1 Source Finding

Our target websites are sparsely distributed on the web. To efficiently find them without visiting unproductive regions on the web, we consider two different strategies, *search* and *backlink*, whose results can be suitably combined. *Search* consists of querying a search engine with a set of seed identifiers of products; we expect that the search engine results contain pages from websites that publish information about these products.

Backlink exploits services that provide inbound links, i.e., pages containing the set of links pointing to a given page; we rely on these services to find hubs, i.e., pages aggregating links to product websites.

These strategies allow us to discover many sources without penalizing recall, with Search focused more on head sources, Backlink including also tail sources. However, they often return also many non-relevant sites. To improve precision, we filter out results that are unlikely to represent a product website.

Search takes advantage of the redundancy of product information: searching the identifiers of known products on a search engine, we expect to find pages of the same products in many different sites. Identifiers to trigger the search engine are extracted from the input set of pages by means of suitable wrappers, or leveraging microdata annotations, such as schema.org, in case these are used in the pages.

Observe that the search engine can also return pages from websites that are not useful for our goals, such as pages from a web forum or pages from news websites. To efficiently select relevant websites, we search for multiple products and rank websites based on the number of pages from the same website that are present in the results. The rationale is that it is unlikely that a website that does not contain product specification pages appears frequently on the results of multiple queries with different product identifiers. Based on the ranking, we select the top-k websites.

It is worth observing that this strategy can penalize the recall of tail sources whose pages do not appear with a sufficient number of occurrences in the search engine results. This limitation can be further exacerbated by costs and limitations of the search engine APIs, which usually impose limits on the number of results per query, and on the number of queries in an interval of time. This is a significant issue if the goal is to collect an unbounded number of product websites.

Backlink aims to deal with the above limitations by adopting an alternate solution that is not dependent on the search engine restrictions. The idea is to find hubs, that is, pages that list links to many product websites. To this end we rely on online services

that return the inbound links to a given input page.[1] Our approach is to search for pages that contain pages containing inbound links to the websites returned by Search. We then consider the domain of the links contained in these pages as candidate product websites. Similarly as for search engine results, backlinks can also lead to non-relevant hubs, subsequently leading to non-relevant websites. For example, sometimes they return generic hubs that point to multiple websites like popular websites in a country.

To select the most promising websites without penalizing the recall, we adopt an iterative approach to compute a ranking of the websites contained in the hubs returned by a seed set of product websites. As non-relevant hubs are less likely to point to many relevant websites, we score hubs based on the number of relevant websites they point to, and similarly we score the relevance of websites, based on the number of hubs that point to them. Based on this intuition, we rank websites and hubs and select the top-k websites.

Filtering Results The collection of websites discovered by Search and Backlink can contain many spurious results. In order to select our target websites, we adopt a simple machine learning approach, training a classifier to recognize if a website is a product website. The features that we have considered include all the anchor texts of the links in the home page.

2.1.2 Source Crawling

Source discovery also deals with crawling the discovered websites towards the product pages of the target categories. Also in this case, efficiency is a challenging objective, as websites often contain a huge number of pages, and only a small fraction may represent pages of interest.

We adopt a crawling strategy inspired by [19], which focused on web forums. Our approach builds on the assumption that, similar to forum websites, product websites have an internal structure consisting of one or more *entry pages* to the product content for a given category, followed by possibly paginated *index pages* that offer links to

[1] These services are used to support search engine optimization (SEO).

product pages. Therefore, our approach first looks for the entry pages related to the target category, then seeks for the index pages.

To discover the entry pages for a given category, the crawler relies on a classifier trained to predict links that are likely to point to entry pages. The classifier, which is trained with the links to entry pages of the target category in the websites of the input set of sample pages, analyses the anchors of links that are present in the home page and in target product pages (those returned by the search engine in the source discovery phase).

A similar approach, based on machine learning, has been adopted also to detect index pages. In this case, the classifier is trained to predict links that lead to product pages. A page is considered an index page if it contains a collection of links that point to product pages and that share uniform HTML format layout and presentation properties.

2.1.3 Identifier Extraction

The last phase of source discovery has the objective of extracting product identifiers from the pages collected by the crawler. These identifiers will be used to launch new searches for discovering other sources.

To extract the product identifiers that can feed a new search, we exploit the local homogeneity of web sources. For each discovered source, we use the set of known identifiers to automatically annotate the pages collected by the crawler; then, for each annotation that exactly matches a textual leaf node of the DOM tree associated with the page, we infer an extraction rule (e.g. an XPath expression). To this end, we use the technique proposed by Dalvi *et al.* [20] to infer the extraction rules given noisy training data, that is, pages annotated by possibly erroneous labels. Because of the local structural homogeneity of the sources, we expect that the application of these rules returns the identifiers that are present in the pages.

2.2 Data Discovery and Extraction

We now describe strategies to automatically extract product descriptions from the collections of product pages harvested by the previous task.

We have concentrated our efforts on specifications, in the form of pairs of attribute name and value, and on the product identifier. Future investigations will target our attention to extract and process also price and product reviews. We now describe our approach to extract specifications, and in Section 2.3 we illustrate our technique to extract product identifiers associated with the product on a page.

Automatically extracting specifications in many websites for different categories of products is challenging. Each website adopts a local template to generate the product specification pages. Then, to accurately extract the data, one should generate a specific wrapper for every website. A solution to this problem could be that of applying an automatic wrapper inference technique, such as Roadrunner [21]. However such a solution exhibits two major drawbacks: first, it has limited performance with irregular templates; second, it extracts a lot of meaningless data, since it considers every item that is not part of the template as data to extract. Another approach is to develop wrappers leveraging domain knowledge, as in [5]. However, as we already discussed, this would limit the opportunity of discovering *tail attributes*; also, with a large number of product categories this approach would require significant effort because of the heterogeneity of specifications across categories.

We have adopted a solution that represents a good trade-off between the above approaches. On the one hand, we exploit publication practices that occur globally in many product websites. On the other hand, we leverage the local homogeneity exhibited in large websites, which build pages according to a bunch of local templates.

Our solution splits the problem in two separate phases: *data discovery*, i.e., detecting the portion of product pages that contain the specifications, and *data extraction*, i.e., extraction of the specification, as attribute name and value pairs.

2.2.1 Data Discovery

We have observed that although specifications can be contained in different HTML structures, they are primarily found within HTML table and list elements. By manually inspecting 301 pages from a great variety of products, we have noticed that 62% of the specifications were inside an HTML table element, and 31% were inside an HTML list element; the remaining 7% were in other miscellaneous HTML elements.

Web pages may contain many tables and lists; however, we have noticed that, independent of product category and of the site, product pages exhibit structural characteristics that can be exploited to determine if a table or a list contains the product specifications (as opposed to being used, e.g., only for formatting purposes). Therefore, we have trained a classifier to detect tables and lists that contain product specifications. The original Dexter systems adopted a Naive Bayes classifier, considering features dealing with table and list contents, such as, statistics about number of links, number of tokens in the leaf nodes, depth of the leaf nodes [17]. We have reimplemented the classifier as part of a system, called Carbonara, that relies on a fully connected neural network, as we describe in Chapter 5.4.

2.2.2 Data Extraction

In order to extract attribute name and value pairs, we have considered two strategies.

The first strategy adopts a simple heuristic based on the observation that the structure of the fragments containing the specifications in tables and lists are very homogeneous, even across sources. By inspecting these tables and lists, we determined that attribute name-value pairs of the specification are often contained in the html row element and that the first and second text elements of each row represent the attribute name and value, respectively. A similar heuristic is applied for the elements of lists classified as specifications.

The second strategy, which is applied on tables, uses the same technique adopted to extract the identifiers, described in Section 2.1.3. In this case, attribute names and values from the input sample pages are used to annotate the pages of the discovered

sources. From the annotations we infer extraction rules. To obtain a more effective approach, we generalize these extraction rules to work on all the rows of the table, thus extracting all the attributes that it offers, including those that are not in the input set.

Also the architecture of this component have been revised in the Carbonara system described in Chapter 5.4.

2.3 Data Linkage

Our approach to data linkage for product specification pages exploits the opportunity that product pages usually contain a product identifier. However locating the product identifiers in product pages at Web scale is quite challenging:

- It is not easy to locate, within the HTML of the product specification pages, the string that represents the identifier; some sources adopt microdata markups (such as schema.org), but their diffusion is limited [22]. Usually identifiers consist of a single token that, for some categories of products, follow specific patterns. But at Web scale, it is not possible to generalize these patterns (as done, for instance, in [23], which concentrated on a handful of sources), because of the large variety of patterns. Similarly, it is not possible to focus only on specific patterns, e.g., those associated with popular standards (as done, for instance, in [24]) because of the skewed distribution of the patterns. To give a concrete example, we have analyzed the identifiers extracted from the subset of pages annotated with microdata markups in the collection of sources discovered by our pipeline. We observed 930 different patterns for $33,281$ values, with the most frequent pattern (a sequence of 8 digits) matching less than 23% of values; the most frequent pattern associated to a standard was GTIN-13, with frequency 3%.

- Product pages usually contain identifiers not only for the main product presented in the page, but also for related products (such as suggested products, products bought by other users, sponsored products).

- Some identifiers may only be local (i.e., only within a source), not useful for linkage across sources. Local identifiers from different sources may conflict; similarly, conflicts may occur among global identifiers of products from different categories. Hence, different product pages associated with the same identifier are not guaranteed to refer to the same product.

To overcome these challenges, we leverage the redundancy of information that occurs at the global level and the regularities of the sources and uniqueness of information that occur at the individual source level [25]. Our approach to data linkage for product pages consists of an iterative phase to extract and filter identifiers, and a conclusive phase to resolve conflicting identifiers and generate the linkages.

2.3.1 Identifiers Extraction and Filtering

We start from a *seed set* of high quality product identifiers, which are used as keywords for searching among the discovered sources product pages that might refer to these products.

Next, for every retrieved product page, we infer an extraction rule for every HTML region containing an occurrence of the searched identifiers. To infer the extraction rule we use again the technique based on noisy annotations. However, here we consider as worth extracting also regions that contain the identifiers, not only those that exactly match the identifiers. In fact, for the purpose of linkage we are interested to extract the identifier that corresponds to the product on the page, and in many sources this is in a textual node, together with other tokens.

Based on the assumption of the local regularities of the sources, the extraction rules are used to obtain regions containing identifiers from all the other product pages in the same source. From every region returned by the rules, we have to select the token that represents a candidate identifier for the primary product of the corresponding page.

Since we cannot rely on a set of predefined patterns to select identifiers among the tokens of large regions, we consider that usually a source provides at most one page for

a product. Therefore, we take the frequency of the tokens as a measure of their ability to serve as identifiers.

An immediate idea is to consider the token source frequency, that is, the token with the smallest number of occurrences within the source. However, considering every source separately does not work well when searching for global identifiers because many sources, especially tail sources, contain tokens that are locally rare, but do not represent global identifiers. For example, consider a small source where there is just one laptop with a touchscreen: if the keyword `touchscreen` is used along with the description of the product, it would be very rare at the source level, and thus it could be erroneously regarded as a good candidate identifier. Even if these tokens might appear as very selective at local level, they are much more frequent if considered globally, especially in the head sources. Continuing the above example, `touchscreen` is a pretty frequent token at the global level.

Therefore, we consider the token collection frequency, defined as the total number of occurrences of a token in the whole collection. In this way, we take into account both local and global characteristics. It is worth noting that because of this property we can perform data linkage only once we have gathered a large number of sources.

The above extraction and selection processes may produce incorrect identifiers, for many reasons: a wrong selection of the candidate identifiers; a region returned by an inaccurate extractor; the presence of regions containing identifiers that do not refer to the main product of the page (e.g., suggested products). To improve the precision, we consider the duality between good extractors and correct identifiers: an extractor is good if the identifiers of its regions are correct; similarly, an identifier is correct if it comes from a region returned by a good extractor.

The selected identifiers are then iteratively used to trigger another search on the source dataset.

2.3.2 Resolution of Conflicting Identifiers

Due to the variety of information across sources, and the presence of local and global identifiers, at the conclusion of the iterations different products could share the same identifier across sources. To improve the accuracy of linkage, we identify these conflicting identifiers by considering that every source consists of a homogeneous set of products: although the criteria that define the uniformity of the product categories are local, not global, with a large enough dataset it is likely that many pairs of products co-occur in many sources because they adopt similar (even if they are not identical) categories. Then, we consider identifiers that co-occur with multiple identifiers in many sources more reliable for the purpose of linking.

2.4 Attribute Alignment

One of the main focus of this thesis is the development of techniques to perform attribute alignment for our product domain. As we shall describe deeper along the thesis, the main difficulties to face are due to the heterogeneity, at the schema and at the instance level, that occur not only across sources, as in the traditional database setting, but also intra-source, that is within specifications provided within a source.

To give a concrete example of the heterogeneity at the schema level, consider the dataset collected using the Dexter crawler. The specifications extracted from these sources contain more than $86k$ distinct attribute names (after normalization by lowercasing and removal of non alpha-numeric characters). Most of the attribute names (about $85k$) are present in less than 3% of the sources, while only 80 attribute names occur in 10% of the sources, with the most popular attribute name occurring in just 38% sources.

Chapter 3

State of the art

In this Chapter we will present a survey of approaches for different step of Data Integration tasks. For each of them, we will provide further details on existing product domain specific techniques.

In Section 3.1 we will discuss on attribute alignment, the main focus of this thesis. We will also define new challenges that attribute alignment for product specifications presents, and what makes existing technique in the state of the art unfit to deal with them.

The other sections describe existing state-of-the-art techniques for related tasks of data integration: Section 3.2 provides Web crawling and extraction techniques, Section 3.3 Record Linkage approaches, Section 3.4 Data fusion approaches.

3.1 Attribute alignment

Attribute alignment (or *schema alignment*) is a field that has been widely studied in last years, especially for what concerns relational databases, regarding the reconciliation of attributes from different sources of data.

It relies on the concept of *schema*, a description of all entities and correspondent attributes provided by a given source (for instance, tables and correspondent columns for relation databases, entity types and predicates for knowledge graphs).

It is a fundamental step of data integration processes, enabling the possibility, along with record linkage step, to define *transformations* of queries and answers from a data source to another, or to obtain a unified view of different sources of data. It can also be useful for other tasks, such as detecting similarities and differences between several sources of data.

Different works provide different definitions of schema alignment, according to the context. Intuitively we can say that its goal is to detect attributes from different sources with the same meaning (often referred as *schema matching)*, and which transformations are needed to harmonise the representation of data in different equivalent attributes (*schema mapping*) [3].

In real-world datasets, however, it is not always possible to detect one-to-one equivalence relationships between attributes, because of differences in *granularities* of attributes or, more generally, in the way the concepts expressed are converted into attributes. For example:

1. An attribute *full name* in a source may be split in two attributes *name* and *surname* in another

2. An attribute *energy efficiency classes* may be absent in another source, but can be derived from an attribute *consumption in kWh*, if the product is known.

3. An attribute *professors* for a dataset of universities, may be derived from attributes *associate professors* and *full professors* from other sources.

Indeed, in a more general way, schema alignment can be defined as a process that aims at finding these kind of *relationships* between attributes of different schemas, clearly characterizing them in order to enable further integration steps [15][1].

[1]Some proposals adopt a slightly different approach, first defining a *mediated schema* between two or more existing sources, that captures all possible aspects (or the ones needed) of the domain of interest that is provided by the sources. The *schema matching* will then consist in finding matches between the attributes of the sources and the mediated schema, while the *schema mapping* will detect relationships and define transformations needed to convert the matched source attributes to the mediated schema attributes [3]

Traditional approaches to attribute alignment are able to deal with *schematic heterogeneity across sources* [26] [27], such as differences in attribute names, in domain format, in granularity. However, they usually rely on *local homogeneity*; that is, they assume that each source adopts homogeneous semantics and homogeneous representations of data [15] [14]. We will see in next Chapter 4 that this assumption is not always valid, especially in some specific Big Data context such as product domain, due to volume and variety of product specifications.

In the following we present relevant works for the main categories of schema alignment approaches. First, we discuss traditional schema alignment techniques. Then, we introduce Universal Schema, a method that can be considered in between schema alignment and relation extraction, and that concerns specifically alignment of knowledge graphs, along with facts extracted from text. Finally, we discuss about a specific technique for alignment of product specifications, explaining its similarities and differences with our setting.

3.1.1 Traditional schema alignment

The reconciliation of databases has always been an important task in many industries for many goals, such as integrating legacy data or creating a unified database for information spread in several enterprise departments.

Traditional schema alignment works have been focused on aligning pairs or few groups of sources, working on different types of structured sources of data, such as relational databases, ontologies, object-oriented or XML.

Classification and main techniques for schema alignment Classical approaches are not usually built as fully automated algorithms *per-se*, instead there are many software programs that offer a wide variety of schema alignment techniques (we can cite as example Coma++ [28] or Harmony component of OpenII [29]). Typically, some manual effort is asked, such as selecting the more adapted techniques to adopt, or filtering all the possible matches suggested by the system, or injecting some training

data.

Existing works have classified most of existing technique, and built a taxonomy that is still valid today [15].

Schema-alignment techniques can be classified in: (*i*) schema-based: they exploit attribute names and other schema meta-data to detect matches, or (*ii*) instance-based: they compare the specific values provided for different attributes to detect matches.

Typical schema-based approaches exploit different features, as follows.

1. Similarities between attribute names, including stemmers, analysis of canonical representation or other pre-processing techniques.

2. Thesauri of synonyms or hypernyms (or word embedding techniques like FastText or Word2Vec) using a general-purpose dictionary, a domain-dictionary or a user-provided one. Fuzzy dictionaries (providing a score for each pair of similar attributes) are more flexible to potential ambiguities.

3. Similarities between attribute descriptions, if available, using more sophisticated techniques (NLP, Named Entity resolution, text mining with bag of words).

4. Analysis of *constraints* of a schema, such as primary and foreign keys.

Schema-based approaches are not very efficient in the product-domain context in which we work. In fact, usually the product specification sources do not provide any schema: it can be built a-posteriori just as the union of the attribute names found in its specifications. In consequence: (*i*) no metadata or attribute descriptions are available; (*ii*) because of the *local heterogeneity* problem, the attribute name is not necessarily an indicator of the information provided by its value; (*iii*) the structure, at least in the context in which we work, is flat: we have a single entity type, the products, and a series of attributes on it. If this can in a way simplify the analysis, on the other hand it means we cannot exploit schema constraints or information about attribute structures.

Instance-based approaches use partly similar techniques and partly specific:

1. Similarities between attribute domains, with techniques similar to schema-level, such as stemming, canonical representation, synonyms, NLP, bag of words analysis, keyword and Named Entity Extraction, Tf-Idf.

2. *Constraint-based characterization*, i.e. characterization of a domain such as numerical value ranges, averages, number of characters maximum values, average sizes, and extraction of recurrent patterns.

Instance-based approach may be more adapted to the problem of schema alignment on product domain, but they are still a partial solution because of the *local heterogeneity*. In general, attribute values do not follow a fixed pattern, with every specification using potentially a different representation (for instance, an attribute *RAM* can provide the information under different formats: “8 GB", “8", “8 Gigabytes" or even “8 GB internal memory"). Moreover, there are cases of *homonym attributes*, i.e. attributes with the same name but different meaning in the same source, which would make domain comparison less significant.

Matches that have already been found (or that are manually selected by users) can be re-used in other context. This concern individual mappings, but also whole schemas or part of them (such as, attributes describing addresses).

Different techniques are usually used together to provide better results. There are different ways to combine them:

1. using machine learning techniques, with matches as features,

2. asking user to make a choice: either before alignment, asking which matcher apply to which portions of the schemas, or after, asking him to confirm or delete certain matches,

3. using Hybrid matcher: instead of providing all matches at the end, and deciding which of them to keep, it is possible to build matchers that take a decision on every single match based on different criteria, combined with threshold on scores.

A well-known approach that exploits hybrid matchers is Rondo [30], based on the concept of *similarity propagation*. Rondo designs schemas as graphs, and iteratively tries to align nodes of the graph via hybrid matchers. The basic idea is to spread similarity from aligned nodes to the adjacent neighbors through propagation coefficients. The iteration stops when a fix point is met, after which no further improvements are possible.

Kang *et al.* propose a schema matching solution for sources with opaque column names which leverages data values [31].This is an instance-based approach, intended to be an extension to existing matching approaches based on simple techniques, like name and domain comparison. They propose to take into account the relations and functional dependencies between attributes in a schema.

Let S_1 and S_2 denote two sources, the intuition is the following: support that (i) an existing technique matched two attributes $A \in S_1$ and $B \in S_2 A$, (ii) attributes A, X in S_1 are related by a functional dependency (for instance, product name and model ID, or product name and brand), (iii), analogously, B, Y in S_2 are related *by a similar dependency*, then X and Y are matched. Concretely, their way to detect *similar dependency* is to measure mutual information of values between each possible pair of attributes in a schema, and then compare graphs between the two schemas.

Using mutual information they can be agnostic to the specific function that connects the two attributes, but they state that other techniques could work on detecting dependencies on specific functions, thus being less general but more precise (they call the two approaches, respectively, *uninterpreted* and *interpreted* matching).

This technique has the advantage to be robust to differences in representation, provided at least one attribute could be matched by standard techniques. On the other hand, it strongly relies on **local homogeneity** of sources, to get sufficient evidences of dependencies between attributes of the same source.

3.1.2 Universal Schema

Universal Schema [32, 33] works on Knowledge graph sources, containing information in form of triples subject-predicate-object, and aims at reconciling predicates between these sources.

This technique creates embeddings for each predicate exploiting known facts. With such embeddings, Universal Schema *learns* latent relations (they call them "asymmetric implicature") between each predicate, and it is able to *augment* each input Knowledge graph, discovering new facts with their own predicates but with subjects and objects coming from other sources. For instance, if a source contains a predicate *works at*, and another source contains a triple *<Mark Zuckerberg, CEO At, Facebook>*, this approach could learn a sort of implication between those predicates, and add to the first source the fact *<Mark Zuckerberg, works at, Facebook>*.

In this way, predicated are never *reconciled* into a mediated schema, but in a way each knowledge present for a predicate found in a source would be transferred to all other similar predicates. Indeed, this technique does not explicitly align predicates between different sources, and do not even define clear relationships between them, claiming that in a general-purpose situation in which they work, they would inherently be ill-defined, having ambiguities, problematic boundary cases, and incompleteness.

Universal Schema is also able to exploit *surface patterns* detected from text: triples composed by pairs of entities detected from text with, for instance, a named entity recognition technique, and the portion of text between them, used as an identifier of the relation (the predicate). A group of surface patterns detected in a corpus of text can be used as additional source.

This approach is indeed much more flexible and could be in principle adapted to product specifications integration systems. However, in practice, an important obstacle is that it requires the identification and disambiguation of all pairs subject-object in the schem. In our case the specific product identifier can be considered the *subject* of a triple, while the attribute value can be considered the *object*. The last one however is complex to disambiguate, as each pair of sources and potentially each specification pairs may

have different ways to represent the same concept, and sometimes the entities are hidden in descriptive attribute (e.g.: $< battery, Li-Ion\ Duracell\ battery,\ modelNIBL4 >$). Additional work could be done to do this adaption, which are outside the scope of this thesis.

3.1.3 Attribute alignment for products specifications

A work by Nguyen *et al.* aims specifically at aligning attributes from product specifications [9].

It was developed by Microsoft for increasing its product catalogue. They use this comprehensive catalog as a provided input, and they try to match attributes of specifications of new products from external vendors to this catalog.

For this purpose, they compare domain of attributes with different similarity measures, in different context (domain for specific category or vendor, or full domains). The similarity measures are then used to train a classifier to detect potential matches, using attributes with the same name as training data.

Notice that an existing catalogue, complete and without *local heterogeneity* is needed for this work to provide good results, otherwise the domain similarity would not be efficient.

Having a complete catalogue, they do not aim at discovering new attributes, but just to detect those with an equivalent in the catalogue. Thanks to this restricted perimeter, they are more flexible to *local heterogeneity* in the external sources, simply because, if an attribute does not match any of existing (because it is not homogeneous, or it has too much errors, or it has a different format then the catalogue) they simply *drop it*, claiming that their goal is not to be complete but to enrich their catalogue, with the best and more reliable specifications that they find on the Web.

This approach is clearly a good compromise in their setting, but it is not adapted if our goal is to integrate data from different sources without a single reliable catalogue, and with the aim of retrieving as much information as possible, including rare or specific attributes.

3.2 Web crawling and Web data extraction

A lot of efforts have been done towards extraction of data from the Web, which is a continuously evolving field. There are, though, efforts to classify the most important ones and underline their main characteristics [34] [35].

An extensive presentation of this field is outside the scope of this thesis. In this Section we will present some of the techniques that are most related to the product specifications domain, and more generic tools for extraction of semi-structured data.

Diadem is a Web crawler and extractor that is able to extract data from websites of specific domains[5]. For each domain, it needs an initial list of websites, a domain model and a few specific rules for extraction. The domain model may include main mandatory attributes for entities, with a definition of data types and patterns. Diadem is however able to learn new rules and new attribute domains during extraction.

Diadem is also able to navigate in Deep Web pages, fulfilling forms, and analyzing the actual visual rendering of pages. Its main output are records extracted from tables and list of entities in particular HTML pages.

Dexter is a solution to detect and extract product specifications from eCommerce websites [17]. It has been exploited, along with other techniques, as the input data for our attribute alignment work.

Dexter needs as input a small seed of product identifiers, eCommerce websites, product pages, and product specifications extracted from those pages. As a first step, it crawls the Web to detect new product pages and new websites, with different techniques:

1. identification of *hubs*, i.e. websites that link to one of the already known sites. The algorithm then follows all other links present in hubs to discover new websites;

2. queries via search engine, using known product identifiers as terms. In this way it discovers new product pages, from results of research, and new web Sources;

3. identification of category entry page: starting from each source homepage, all links are followed, and category entry pages are detected with a classifier based on the text of the link;

4. product pages and index pages (pages with links to product pages) are detected following links inside pages, and using several heuristics and classifiers based on the structure of the pages (using product pages already found) and the path inside the source;

5. product identifiers are extracted from product pages;

6. all the former steps are relaunched iteratively, until no new product pages or websites are found.

Once product pages are detected, Dexter tries to extract product specifications from each of them.

1. For each seed product specification, Dexter computes features like number of rows, average length and number of tokens, and uses them to build a training set of a classifier. Negative examples are detected extracting other web tables and web lists from seed product pages.

2. For each web tables and bulleted list found on a page, Dexter runs the classifier to detect which are real product pages.

3. Each row of a table or list is considered a pair $< attribute, value >$ of a product specifications. Standard separators like colon or tabs are used to separate attribute names from values.

Weir addresses the problem of data extraction and integration contextually [36].

It relies on a generative model for Web sources, defining an abstract perfect source of information, from which each source takes a part of the information, adding its own format, putting in its DOM structure and potentially introducing error.

The goal is so to "reverse engineer" this sequence of actions, defining, for what concerns data extraction, a list of all possible extraction that can be made from the DOM of each page, and iteratively choose the best one with a probabilistic approach exploiting redundancy of information.

3.3 Record Linkage

Record linkage is another fundamental step in Data Integration systems, whose goal is to decide which records of two or more data sources refer to the same entity.

Unlike schema alignment, it concerns real-world entities rather than abstract concepts: in this scenario, it is rarely necessary to define complex relations between records, and it is generally fine to just use 1-to-1 equivalence relations[2].

For this reason, and for the fact that each record inside a source present generally the same structure, *record linkage* approaches are usually not *domain-specific* (especially for sources already aligned). There are, though, some exceptions: products domain, among others, have a universal identifier (generally provided with specifications) that allows linking records without the need to analyze its attributes. In this way record linkage can be done before schema alignment.

Many approaches have been proposed in the state of the art. Christen *et al.* proposed in this sense an extensive survey [37]. In this Section we will briefly describe a few representative approaches of the state of the art in this task.

Magellan is a new kind of *Entity Matching* (EM) system. It represents the state of the art for techniques based on *machine learning* (specifically non-neural approaches) addressing Record Linkage [38].

The Magellan approach is composed of two stages: (*i*)*development* and (*ii*) *production*.

In the development stage Magellan selects, on a sample of the dataset, candidate linkage records via a blocking technique[3], then asks user to manually validate these candidates. The validated pairs will be used as training data for a classifier, that will be used to determine linkages in production step.

Magellan allows user to customize every step of its workflow:

[2]Some complex situations may arise, like a *camera kit* record in one source, which may be presented in separated records for camera body and lens in another source, or a series of camera variants with different colors, which may be presented as a single record in another source

[3]simple heuristics to create *blocks* of linkage candidates

1. Pre-processing steps (such as cleaning, extract and transform)

2. Blocking technique, with pre-defined tool that exploits similarity between domains or token similarities

3. Match classifier: many machine-learning tools (non-neural) are available, such as Random Forest, SVM, Naive Bayes, Linear and Logistic Regression

4. Feature extraction. For this purpose Magellan has a utility for automatically inferring what features could be useful based on an analysis of the fields available among the instances of the dataset: it provides 23 string similarity measures, such as Hamming Distance, Levenshtein Distance and Smith-Waterman.

DeepMatcher is an approach that exploits state-of-the-art deep learning models for entity matching. It has a modular design so it is easily customizable [39].

DeepMatcher uses labeled tuple pairs and trains a neural network to perform matching, i.e., to predict match / non-match labels. Figure 3.1 shows examples of training data in input. The trained network can then be used to obtain labels for unlabeled tuple pairs.

The schemas of the input sources must be aligned. However, it is also possible to match textual data, using a schema with a single attribute.

Label	Left Product Title	Left Manufacturer	Left Price	Right Product Title	Right Manufacturer	Right Price
Match	turbocad mac 3d for mac	imsi	249.99	imsi design llc turbocad mac 3d		204.67
Non-Match	adobe indesign cs3 upgrade [mac]	adobe	199.0	adobe photoshop cs3 for mac upgrade	adobe	189.99
Match	photoshop elements 4 mac retail eng 1u	adobe	145.39	adobe photoshop elements 4.0 mac		79.99

Figure 3.1: Example of labeled tuple pairs

Figure 3.2 shows the pipeline of DeepMatcher for linking an entity pair, composed of three main customizable modules:

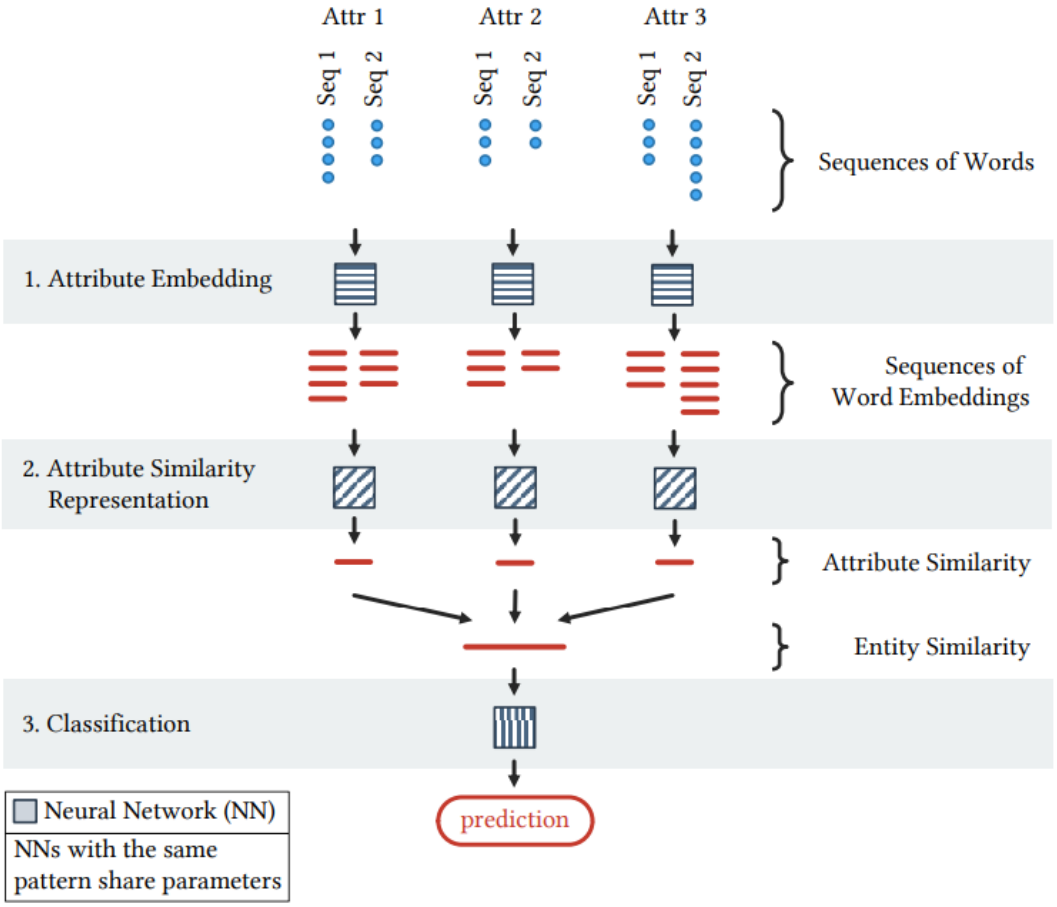

Figure 3.2: Architecture template of DeepMatcher

1. The *Attribute Embedding Module* converts values of attributes of entity pairs into word embedding vectors

2. The *Similarity Representation Module* automatically learns from the provided embeddings, a vector representation that captures the similarity of two entities.

3. The *Classification Module* uses the similarity representation vectors as features for a classifier that determines id the two entities should be linked.

3.3.1 A record linkage technique for product specifications

Qiu *et al.* proposed a technique to link product specifications from different product sources, that exploits the universal products identifier that most sources provide [25].

This solution takes as input the HTML pages of Web sources providing product specifications, and a *seed set* of high quality identifiers. It associates pages to identifier with the following steps:

1. search for known ids in pages of each source, and tries to infer an extractor for every html region in which IDs were found;
2. dilter out bad identifiers using an iterative approach, computing scores for quality of extractors and reliability of IDs;
3. use the extractor to detect new identifiers;
4. iteratively repeat steps 1-3 until no more evidences could be found.

After these steps some pages may still have multiple identifiers associated. To solve these conflicts, the algorithm *clusters* pages according to different similarity measures, and, for each page, exclude detected identifiers that are mostly associated to pages in external clusters.

3.4 Data fusion and error detection

Data fusion aims at resolving conflicting data coming from different sources, deciding which one is correct and which is wrong. Data fusion step must be executed after schema alignment and record linkage have been completed.

Dong *et al.* proposed an approach to detect the correct object for a series of facts (such as the director of a particular movie, or the winner of a prize in a specific year) given the answer retrieved from a series of sources (it presumes the data extraction phase already done) [40].

It exploits a Bayesian E-M probabilistic model, trying to detect at the same time the correct value (thus the trueness of answers provided by sources), the reliability of each source, and potential copiers, i.e. sources that completely or partly takes an information from other sources, regardless of their correctness.

Knowledge Vault proposes a Knowledge graph augmentation system [41]. First of all, data is extracted from Web content with different existing techniques: extraction from text, from HTML DOM, from Web tables and from manual HTML annotations (provided directly in Webpages using some pre-defined ontologies). Afterwards, a supervised classifier detects the reliability of each of the fact extracted, exploiting data present in the existing knowledge graph.

The training data are built looking at data extracted that were already present in KG, using the principle of *local closed world assumption*: a fact (s, p, o) present in the KG is true, a fact (s, p, o) absent from KG, but for which there exist one or more corresponding facts (s, p, o') with the same predicate and other objects, is considered as false. All other facts are indefinite. A fact can also be evaluated even if it is indefinite, using inference from the Knowledge graph (e.g: if a source states that Obama was born in Kenya, and there is no data about the place of birth of Obama in the KG, the system can predict this data as false, considering that an american President must have been born in USA).

This work covers different aspects of Web data exploitation, from extraction to integration, but its main contribution is the fusion of data, so it is defined in this Section.

Another work from Dong *et al.* has the same goal than Knowledge Vault, and uses the same closed-world assumption, but proposes different techniques mostly based on a Bayesian approach, easily horizontally scalable and adaptable to Map-Reduce framework [42].

Knowledge-based Trust is an extension to Knowledge vault approach, that determine the correctness of facts using a probabilistic approach, computing as latent variables the reliability of sources and correctness of extractors[43]. The model is also capable

to determine the granularity at which each source must be evaluated (a source may be generally good, but bad for a particular kind of data, for instance).

Once errors are detected, it may be useful to understand their cause and potentially fix them. A work from Wang *et al.* takes as input a seed of facts extracted and the information on which are correct and which not [44].

With a Bayesian approach and a search algorithm it tries to group errors according to some features (e.g: all errors from a given source that concerns football players). Features include subject, predicate and object instances, and their types, source and sections of this source (e.g.: a specific extracted table). Further manual investigation can be done on each detected group of errors to understand their cause.

Chapter 4

Instance Level Attribute Alignment

As we discussed in Chapter 2, despite all the work on data integration, integrating product specifications from multiple sources is still not a solved problem and raises novel and intriguing research challenges [10, 7] and therefore deserves specific attention. Data extracted from such sources can be seen as organized in sets of attribute name-value pairs, and a major issue is the reconciliation of attributes, usually called *attribute alignment* in this framework.

4.1 A new challenge: *local heterogeneity*

To give a concrete idea of the challenges that arise in this setting, consider the dataset of the data integration benchmark (that we introduce in Chapter 5), which contains about 30k product specifications described as sets of attribute name-value pairs from 24 real Web sources. The DI2KG camera dataset shows significant heterogeneity in specifications, both across and within sources, as it is the case that each individual source is in turn fed by autonomous data providers, such as individual sellers (for sites that offer used items, like eBay) or third party merchants (for large sites, like Amazon) [7]. This makes it infeasible to use traditional approaches to attribute alignment, which consider heterogeneity across sources, but they rely on *local homogeneity*, that is, they assume that each source adopts homogeneous semantics and homogeneous representations of

data [14, 15]. Let us comment on the major forms of heterogeneity in the DI2KG dataset.

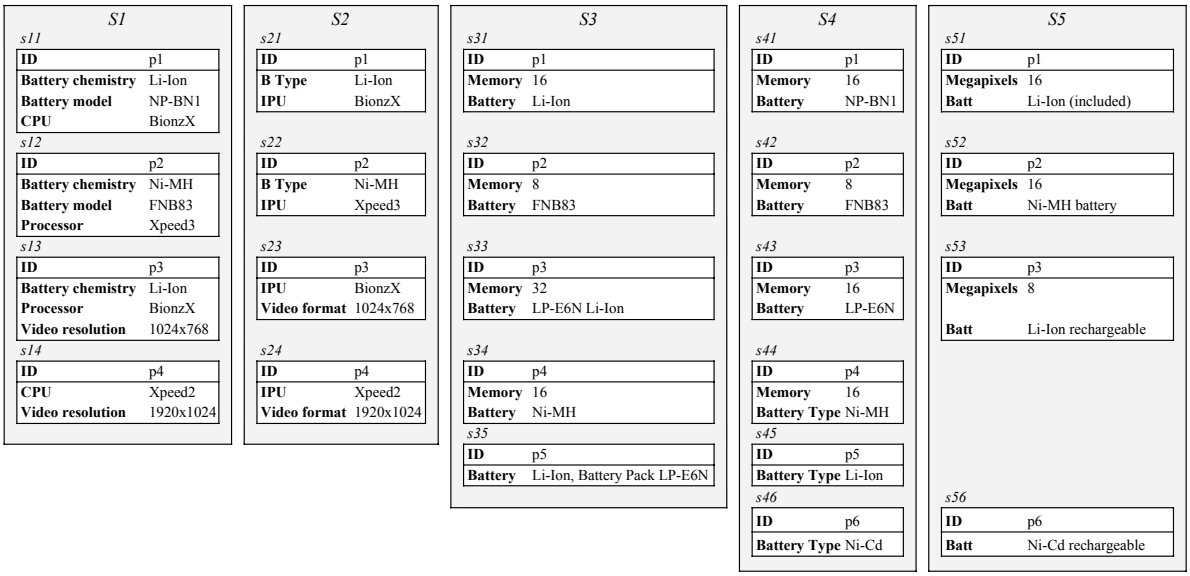

Figure 4.1: Running example: product specifications from the DI2KG dataset.

Attribute sparsity. The total number of attribute name-value pairs in the DI2KG dataset is 528,186; on average, each specification contains about 18 pairs. However, the dataset has 4,631 distinct attribute names across sources (6,560 counting duplicate attribute names across sources): on average, 273 attribute names per source. In every source, each attribute name is used on average only by 10% of the specifications (or instances).

Synonyms. Attribute sparsity is not only due to tail (i.e., rare and specific) attributes, but also to synonyms, i.e., attributes with different names but same semantics across different instances, even within the same source. Figure 4.1 shows a (tiny) portion of the actual specifications of four cameras (white boxes) that may be found in five sources (grey boxes) of the dataset. Observe attribute names `CPU` and `Processor` in source S_1: they have the same meaning, yet some specifications in the same source use `CPU` and others `Processor`. In addition, attribute synonyms also appear across different sources: attribute name `IPU` in source S_2 has the same meaning as `CPU` and `Processor` in source S_1.

Homonyms. Another challenge is due to the presence of attributes with same name but different semantics for different instances, even in the same source. In Figure 4.1, observe the `Battery` attribute in source S_3: for some instances the value refers to the battery chemistry (instance s_{31}, in the example), for others it refers to the battery model (s_{32}), or even provides both properties (s_{33}). Homonyms occur also across sources: attribute name `Battery` in source S_3 has a different meaning from the homonym attribute in source S_4.

Representation heterogeneity. Sources offer several attributes with the same name and meaning but values that use different representations across instances – even from the same source – possibly including pieces of free text to aid human users. In our example, consider the `Batt` attribute in source S_5 whose values are adorned by descriptive words (such as "included" and "rechargeable").

Mixed Attributes. There are attributes whose values are compositions of pieces of information that elsewhere – even in the same source – are represented by means of several attributes. In our example, consider the `Battery` attribute for the specifications s_{33} and s_{35}, whose values include both the battery type and the battery chemistry, without any underlying pattern.

It is worth observing that the above forms of heterogeneity, which are present both across sources and within the same source, can occur in combination. For example, the `Battery` attribute in source S_3 is a mixed attribute and assumes different semantics (homonym) within S_3.

4.2 From *Schema level* to *Instance level* Attribute alignment

This work addresses the important problem of attribute alignment in such a challenging and intriguing setting, where the major issue is the violation of the local homogeneity assumption in the many ways discussed earlier. This requires going beyond traditional

solutions, which aim at aligning attributes at the schema level. We propose a solution to compute attribute alignment at the *instance level*.

In a traditional setting, within a source, attributes with same name have the same semantics and representation homogeneity. Thus, a *schema* can be defined for each source, as the set of its attribute names and types of its values, and the problem of attribute alignment can be defined as grouping attribute names with the same semantics between source schemas (at the *schema-level*). In our setting, we cannot align attributes at the schema-level, as attribute name in a source does not identify the semantics of the attribute for all the instances. Instead, we are compelled to perform our alignment at the *instance level*: we propose a solution that groups semantically equivalent attribute name-value pairs.

Figure 4.2 illustrates a representation of part of the output of our instance level attribute alignment for the data in Figure 4.1, limited to specifications related to product p1 and p3. For each product, we report the alignment of attribute instances across sources. Also, attributes with the same semantics across products are grouped (they refer to the same property). It is worth observing that such alignment allows a data integration process to provide the provenance of the values for all the attributes of each instance.

We follow the data integration pipeline proposed in [25], which performs record linkage before attribute alignment. The rationale is that while attribute alignment is infeasible at the beginning of the pipeline because of the heterogeneity of the sources, record linkage can be computed by exploiting specific properties of the domain, like the presence of product identifiers [45, 25], and the linkage among the instances can be subsequently exploited to align the attributes.

Our approach has been developed in the framework of RaF (Redundancy as Friend), an ongoing research project that addresses the issue of integrating product specifications from multiple heterogeneous sources [10, 25]. We introduce a solution, RaF-AIA (Attribute Instance Alignment) to the problem of aligning attributes at the instance level that aims at leveraging opportunities which derive from the richness of information

across sources:

- Although local homogeneity is not always satisfied within a source (as discussed earlier), many attributes in a source tend to be locally homogeneous, identifying and taking advantage of such attributes in attribute alignment across sources is a useful opportunity;
- Some sources distinguish, by means of different names, attributes that elsewhere are presented with homonym names (e.g., source S_1 provides battery chemistry and battery model by means of two attributes with distinct names). This can allow homonyms to be distinguished;
- Common meaningful values (e.g., "Li-Ion") appear in multiple specifications, for different products, within and across sources, despite the variety of heterogeneity. This redundancy can be exploited;
- Given the number of sources, there is a lot of redundancy of information across them, despite the attribute sparsity. Again, redundancy across sources can be used as evidence of semantic equivalence.

We propose an iterative approach consisting of three key steps to solve the instance level attribute alignment problem by taking advantage of the above opportunities offered by the redundancy and variety of information across sources. The first step is based on a Bayesian model that analyses the overlapping information across sources to match the most *locally* homogeneous attributes. The second step is based on a tagging approach, inspired by NLP techniques, to create groups of *virtual* homogeneous attributes from tagged portions of values of heterogeneous attributes. Matching and tagging are iterated, as they produce complementary evidence that reinforce one each other. A final step processes the results of the iterations and aligns *globally* homogeneous attributes that the previous steps were not able to merge otherwise (for example for lack of linkage, or for differences in representation of values).

ID	p1	
property1	**s11.Battery chemistry**	Li-Ion
	s21.B Type	Li-Ion
	s31.Battery	Li-Ion
	s51.Batt	Li-Ion (included)
property2	**s11.Battery model**	NP-BN1
	s41.Battery	NP-BN1
property3	**s11.CPU**	BionzX
	s21.IPU	BionzX
property4	**s31.Memory**	16
	s41.Memory	16

ID	p3	
property1	**s13.Battery chemistry**	Li-Ion
	s33.Battery	LP-E6N Li-Ion
	s53.Batt	Li-Ion rechargeable
property2	**s33.Battery**	LP-E6N Li-Ion
	s43.Battery	LP-E6N
property3	**s13.Processor**	BionzX
	s23.IPU	BionzX
property4	**s33.Memory**	32
	s43.Memory	16
property5	**s13.Video resolution**	1024x768
	s23.Video format	1024x768

Figure 4.2: Running example: attribute alignment at the instance level for the specifications in Figure 4.1 related to products p1 and p3.

4.3 Overview

We consider a set of sources **S** that provide information about products in a vertical domain of interest (e.g., camera). Each source might be the result of an extraction process from a website (e.g., the output of a Web scraper), or a formatted file returned by an API (e.g., a json file). We assume each source $S \in \mathbf{S}$ to be a set of *records*, each of which is indeed a "product *specification*" and provides the *properties* of a product as

a set of attribute name-value pairs. In each specification, an attribute name appears at most once, and values are strings.

We assume, and take advantage of, the availability of a *good quality sample* of record linkage information, which identify different specifications (records) that refer to the same product.[1] Such a good quality sample could be obtained using human validation (e.g., based on crowdsourcing) of a sample of the results of an automated method such as [10, 25]. It is worth noting this good quality sample does not need to be perfect (i.e., a small amount of noise in the linkage result is acceptable) nor does it need to be complete (i.e., a random sample suffices). To model record linkage information, every record has an *identifier*, a special attribute name, denoted `ID`, that identifies the product. Specifications that have the same `ID` value refer to the same product.[2] When two (ore more) specifications refer to the same product, i.e., they have the same `ID` value, we say they are *linked*.

Let $s = \{\langle \texttt{ID} : p\rangle, \langle A_1 : v_1\rangle, \ldots, \langle A_n : v_n\rangle\}$ be a product specification in a source $S \in \mathbf{S}$. Given a pair $\langle A_i : v_i\rangle$ in s, we use $s.A_i$ to refer to the value v_i.

We say that A is a *source attribute* in S, denoted $S.A$, if there is at least a record in S with a pair $\langle A : v\rangle$, that is if the attribute name A appears in at least one specification in the source. We call this an *occurrence* of source attribute A, or more generally an *attribute instance*. The set of all source attributes of a source S is the *schema* of that source.

4.3.1 Problem Definition

Given a set of sources, $\mathbf{S}$, our goal is to align the attributes of all the specifications in $\mathbf{S}$: we aim to group together attribute instances coming from different specifications that provide information dealing with the same product property.

It is worth noting that the definition of what is a product property might be subjective. We follow a data-driven approach, which considers properties as they emerge from

[1] A product can be associated with several specifications from different sources, but also from the same source.

[2] This means that the identifier refers to the product, not to the specification.

the data offered by the sources. For example, the presence of distinct source attributes that describe battery chemistry and battery model (in source S_1) suggests that these are distinct properties of the domain. Conversely, some attributes provide values that might ideally be considered as a composition of multiple properties; but if these properties do not emerge in any specification as distinct attributes, we do not consider them separately. For example, consider the `Video resolution` attribute in $S1$ in our running example: it could ideally represent two properties (horizontal and vertical resolution), but if they are never presented by two distinct attributes, they are not (and should not be) considered separately.

In this context, we want to solve the *Instance Level Attribute Alignment Problem*, which can be informally described as follows: given a set of sources **S**, our goal is to create a set of clusters, such that each cluster contains attribute instances from the specifications in **S** that refer to the same property.

As we discussed in Section 4.2, this is a different goal than traditional schema-level approaches, whose goal in our context would be to create clusters of *source attributes* referring to the same property. In our context, a single source attribute could provide different properties in different occurrences, so a higher level of granularity is needed. We will discuss again about this difference in Section 5.4, when we will talk about construction of ground truth for different data integration tasks in context of Alaska Benchmark.

Figure 4.2 shows a representation of the result of instance level attribute alignment for the sources in the running example of Figure 4.1. It is worth noting that since specifications can include mixed attributes, which correspond to composition of properties that in other specifications are provided by distinct attributes, clusters can overlap. For example, the `Battery` attribute of s_{33} belongs to the cluster that corresponds to the chemistry of the battery as well as the one that corresponds to the model.

4.3.2 Our Approach

To address the instance level attribute alignment problem, we exploit the opportunities that arise from the richness of information across sources. Figure 4.3 illustrates our approach.

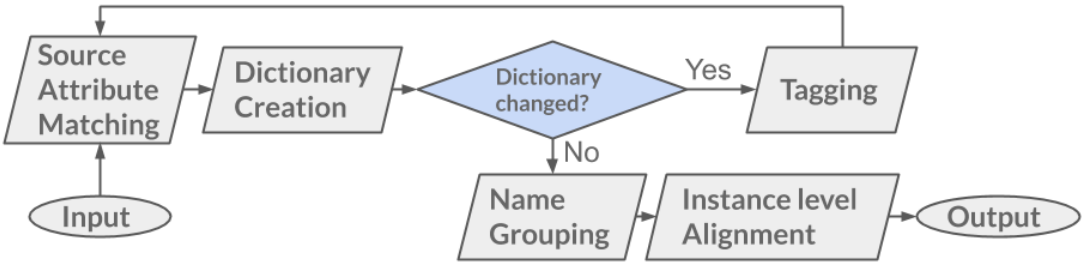

Figure 4.3: The RaF-AIA approach to instance level attribute alignment.

First, we exploit local homogeneity and overlapping data across sources: we develop a Bayesian analysis that leverages the linkage sample to compute clusters of source attributes that share instance attributes with matching values.

Source attributes grouped in the same cluster are supposed to have the same semantics, as it is unlikely that their linked instances matched by chance. We then exploit common meaningful values from these clusters to address the issues related to the various forms of heterogeneity that occur in the sources. Our approach builds a *dictionary* of values for every non-singleton cluster created by the Bayesian matching step. Then, we use dictionary terms to *tag* the values of the dataset: any sequence of tokens that matches with a term of the constructed dictionary is tagged with a label identifying the cluster associated with such a dictionary term. Elements tagged with a given label may represent values of attributes with the same semantics of the attributes in the cluster associated with such a label. We extract these values to create *virtual* attributes that are added to the original dataset.

Such an augmented version of the dataset is submitted again to the Bayesian analysis: the virtual attributes introduced by the tagging step can give rise to new matches that were not possible to identify previously because of heterogeneity of the attributes.

As source attribute matching can produce new clusters (and then new dictionaries), and new virtual attributes can allow new matching, the two steps are repeated until the dictionaries do not change anymore.

After the end of iterations, clusters having source attributes with common names are selected as candidates for merging, provided that these attributes are not isolated attributes tagged during the tagging steps (i.e. probably not locally homogeneous). If the two clusters have comparable domains, they are merged. This final step exploits redundancy of data, as the domain of clusters is now more reliable than domains of single attributes, and can match attributes that did not match previously, typically for lack of linkage data, too many noisy values or too generic attribute values.

At the final stage, non-isolated clusters contain only locally homogeneous source attributes, original or virtual. Instance-level clusters come directly from these clusters, just putting together all occurrences of each source attribute.

4.4 Source attribute matching

In this section, we describe the first step of our approach, which aims at identifying groups of locally homogeneous source attributes with equivalent semantics. Our approach leverages the linkage sample and redundancy of data among sources. Algorithm 1 presents the pseudo-code.

Let $\mathbf{A}$ be the set of all the source attributes in $\mathbf{S}$. We build a weighted graph, $G(\mathbf{A}, \mathbf{E})$, whose nodes are given by the set of source attributes $\mathbf{A}$, edges correspond to candidate matches between source attributes. The weight of each edge represents the similarity of the connected source attributes which is computed according to a scoring function, denoted SIM-SCORE, based on a Bayesian analysis that considers the values of the two attributes in the linked sample. Section 4.4.1 explains the computation of the similarity score.

To reduce the number of edges, we consider only pairs of source attributes from different sources that share at least two values in the linked sample, or one value,

Algorithm 1: Source Attribute Matching

Input : a set **S** of sources
Output: a set **C** of source attribute clusters

Construct a graph $G(V, E)$ as follows: $V = \mathbf{A}$, E contains a weighted edge $e_{A,B}$, with $weight(e) = \text{SIM-SCORE}(A, B)$, between a pair of source attributes A and B, $A \in S_i$, $B \in S_j$, $S_i \neq S_j$, if A and B share a value in at least one linked specification and $\text{SIM-SCORE}(A, B) \geqslant 0.5$;
$\mathbf{C} \leftarrow \{\{A\}, A \in V\}$ //start with singleton clusters;
for *each edge* $e_{A,B} \in E$ *ordered by descending weight* **do**
    $c_A \leftarrow c \in \mathbf{C} : A \in c$;
    $c_B \leftarrow c \in \mathbf{C} : B \in c$;
    if $\nexists s : X \in s, Y \in s, X \in c_A, Y \in c_B$ **then**
        $\mathbf{C} \leftarrow (\mathbf{C} \backslash \{c_A, c_B\}) \cup \{c_A \cup c_B\}$ // merge c_A, c_B;
    end
end
return C;

provided that is it present in at most 10% of attributes in the dataset.[3]

The clustering algorithm is based on the edge weights (i.e., probability of match between pairs of clusters), and exploits the assumption that no specification can contain two distinct attributes with the same semantics, that is, attributes are *distinguishable* within each specification. The clustering algorithm initially builds singleton clusters for each source attribute. Then, edges are sorted by descending weight. For each edge with enough match probability (that is, weight greater than 0.5), if the union of the two clusters connected by the edge does not violate the distinguishable attributes assumption (that is, it does not contain attributes belonging to the same specification), the two clusters are merged.

4.4.1 Similarity Score

The similarity score between two source attributes is computed by means of a probabilistic Bayesian approach that leverages the linkage sample. The intuition that we follow is that if two source attributes share the same values for several instances, it is

[3] In practice, we observed that this simple heuristic filtered out just attributes pairs whose only common value was *yes*.

likely that they represent the same property. The approach aims to be tolerant to the presence of noise in the values as well as in the linkage.

Given two source attributes $A \in S_i$ and $B \in S_j$, $S_i \neq S_j$ let L_{AB} be the set of pairwise values of A and B in the linked specifications of S_i and S_j for which both A and B provide non-null values. Formally:

$$L_{AB} = \{\langle s.A, t.B\rangle : s \in S_i, t \in S_j, s.\texttt{ID} = t.\texttt{ID}, s.A \neq null, t.B \neq null)\}$$

Our goal is to determine $P(A \equiv B|L_{AB})$, that is, the probability that attributes A and B are equivalent (that is, they have the same meaning), given the pairs of the values of attributes A and B in the linked specifications.

By applying Bayes' theorem:

$$P(A \equiv B|L_{AB}) = \frac{P(L_{AB}|A{\equiv}B)P(A{\equiv}B)}{P(L_{AB}|A{\equiv}B)P(A{\equiv}B)+P(L_{AB}|A{\not\equiv}B)(1-P(A{\equiv}B))} \quad (4.1)$$

We first illustrate the computation of the prior probability, $P(A \equiv B)$, then we develop the posteriors, $P(L_{AB}|\cdot)$.

4.4.1.1 Prior probability

Given a pair of source attributes A and B, we estimate the prior probability $P(A \equiv B)$ by heuristically considering the similarity of their domains, denoted $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively. In particular, we take into account (i) the similarity of the whole domains, and (ii) the similarity of the domains restricted to the values shared by the linked specifications.

We compute the similarity of the domains according to the Chekanovsky-Sørensen index [46, 47], a variant of the Jaccard index that considers the distribution of values in the sets. Also, we weight the contribution of each value by its frequency in the set of source attributes; the rationale is that sharing values that are very frequent in the dataset is less informative than sharing rare values. Formally:

$$sim(A, B) = \sum_{v \in \mathcal{D}_A \cap \mathcal{D}_B} 2\frac{min(f(v,\mathcal{D}_A),f(v,\mathcal{D}_B))}{|\{Y \in \mathbf{A}: v \in \mathcal{D}_Y\}|} \quad (4.2)$$

where $f(v, \cdot)$ is the frequency of v in $\cdot$, and the denominator $|\{Y \in \mathbf{A} : v \in \mathcal{D}_Y\}|$ is the number of source attributes in which the value v appears.[4]

We use the same approach to compute the similarity of the domains restricted to the values from the linked specification (we call this *linked domain*):

$$sim_L(A, B) = \sum_{\langle v,v\rangle \in L_{AB}} 2\frac{min(f(v,L_A),f(v,L_B))}{|\{Y \in \mathbf{A}: v \in \mathcal{D}_Y\}|} \tag{4.3}$$

where L_A (and similarly L_B) is the set of values of A in L_{AB}, i.e., $L_A = \{s.A : \langle s.A, t.B\rangle \in L_{AB}\}$.

In order to compute the prior probability $P(A \equiv B)$ we rely mostly on values that are provided by linked specifications. However, if there are few elements in linkage, values could be shared by chance, and thus we should give more weight to the similarity between the entire domains. To weight the two components we consider their sizes; however, since they can differ by many orders of magnitude, we use their logarithm, as follows:

$$P(A \equiv B) = \frac{\alpha\ sim(A,B)+\beta\ sim_L(A,B)}{\alpha+\beta} \tag{4.4}$$

where $\alpha = log(\frac{max(|\mathcal{D}_A|,|\mathcal{D}_B|)}{|L_{AB}|})$ and $\beta = log(|L_{AB}|)$. Note that if $|L_{AB}| = 1$ (there is just one pair of linked specifications) β equals 0, and then we consider only the similarity of the domain. Conversely, increasing $|L_{AB}|$ reduces the weight of α and thus gives more importance to the values coming from the linked specifications.

Example 1. Consider the source attributes $S3$.`Memory` and $S4$.`Memory` from our running example in Figure 4.1. For the sake of readability, we use A and B to denote $S3$.`Memory` and $S4$.`Memory`, respectively.

Table 4.1 reports the frequencies of the values in the domains associated with the two attributes (f_A and f_B); their frequencies in the domains restricted to the linked specifications (f_{L_A} and f_{L_B}); the frequencies of the values in $A \cup B$ ($f_{A\cup B}$); the number of source attributes in which v occurs ($occ(v) = |\{S.Y, S \in \mathbf{S} : v \in \mathcal{D}_{S.Y}\}|$).

[4]The numerator is 2 because by construction we are considering values that appear in the intersection of two sources.

v	f_A	f_B	f_{L_A}	f_{L_B}	$f_{A\cup B}$	$occ(v)$
16	0.5	0.66	0.33	0.66	0.57	3
8	0.25	0.33	0.33	0.33	0.29	3
32	0.25	0	0.33	0	0.14	1

Table 4.1: Domain of attributes $S3$.`Memory` (denoted A) and $S4$.`Memory` (denoted B).

By applying Equation 4.2 and 4.3, the similarity of domains and the similarity of linked domains are follows:

$$sim(A,B) = 2\frac{min(0.5,0.66)}{3} + 2\frac{min(0.25,0.33)}{3} = 0.5$$
$$sim_L(A,B) = 2\frac{min(0.66,0.33)}{3} + 2\frac{min(0.33,0.33)}{3} = 0.44$$

To compute the prior we need the weights α and β:[5]

$$\alpha = log(\frac{max(4,3)}{3}) = 0.42; \qquad \beta = log(|L_{AB}|) = log(3) = 1.58$$

Finally, from Equation 4.4:

$$P(A \equiv B) = \frac{0.5 * 0.42 + 0.44 * 1.58}{0.42 + 1.58} \approx 0.45$$

In a similar way we can compute the prior for attributes $S4$.`Memory` and $S5$.`Megapixels`. Let B and C denote $S4$.`Memory` and $S5$.`Megapixels`, respectively.

$$P(B \equiv C) = \frac{0.66 * 0 + 0.66 * 1.58}{0 + 1.58} = 0.66$$

These two attributes have very similar domains, thus a high prior, despite providing different data. Example 2 will show how the contribution of the posteriors adjust the estimation of the probability of alignment.

4.4.1.2 Posterior probability

We now need to compute, given the domain of attributes, the posterior probabilities, i.e., the probability to observe the values provided for each specific product in linkage,

[5]For the logarithms we use base 2 (the choice of base has no impact on the result).

both under the equivalence hypothesis (positive posterior, $P(L_{AB}|A \equiv B)$) and the null-hypothesis (negative posterior, $P(L_{AB}|A \not\equiv B)$).

We can model the set of the observed value pairs L_{AB} as a set of independent events, one for each linked pair of specifications.

$$P(L_{AB}|\cdot) = \prod_{<s.A,t.B>_i \in L_{AB}} P(s.A, t.B|\cdot) \tag{4.5}$$

Negative posterior. Under the null-hypothesis, the two attributes are not aligned and we can assume that they are independent:

$$P(s.A, t.B|A \not\equiv B) = P(s.A|A \not\equiv B)P(t.B|A \not\equiv B) \tag{4.6}$$

We model the value provided by each attribute as the outcome of a random variable, where the probability of each value is estimated with its frequency in the domain of the corresponding attribute. Therefore:

$$P(s.A, t.B|A \not\equiv B) = f(s.A, \mathcal{D}_A)f(t.B, \mathcal{D}_B) \tag{4.7}$$

Positive posterior. Under the equivalence hypothesis, the two attributes represent the same *real-world* property, which we model by means of a random variable X.

In order to compute $P(s.A, t.B|A \equiv B)$, we need to distinguish two cases: the values provided by the two attributes are either (i) different or (ii) equal. Intuitively, we expect that they are equal (we are under the equivalence hypothesis), unless one of the two values (or both) is wrong.

Let us consider first the case in which $s.A$ and $t.B$ are different. Either only one of them provides the actual value of X, or they both make errors, and none provides the actual value of X:

$$\begin{aligned} &P(s.A = v_1, t.B = v_2, v_1 \neq v_2|A \equiv B) = \\ &\quad P(X = s.A = v_1, t.B = v_2, v_1 \neq v_2|A \equiv B) + \\ &\quad P(s.A = v_1, X = t.B = v_2, v_1 \neq v_2|A \equiv B) + \\ &\quad P(s.A = v_1, t.B = v_2, X \notin \{v_1, v_2\}|A \equiv B) \end{aligned} \tag{4.8}$$

By applying the conditional probability definition:

$$\begin{aligned} &P(s.A = v_1, t.B = v_2, v_1 \neq v_2 | A \equiv B) = \\ &\quad P(X = v_1 | A \equiv B) P(s.A = v_1 | X = v_1, A \equiv B) \\ &\quad\quad P(t.B = v_2 | X = v_1, A \equiv B) + \\ &\quad P(X = v_2 | A \equiv B) P(s.B = v_1 | X = v_2, A \equiv B) \\ &\quad\quad P(t.A = v_2 | X = v_2, A \equiv B) + \\ &\quad P(s.A = v_1 | X \notin \{v_1, v_2\}, A \equiv B) \\ &\quad\quad P(t.B = v_2 | X \notin \{v_1, v_2\}, A \equiv B) \\ &\quad\quad\quad P(X \notin \{v_1, v_2\} | A \equiv B) \end{aligned} \tag{4.9}$$

Since we are under the hypothesis that the attributes A and B are equivalent, the union of their domains represents an approximation of the domain of X. Then, we can estimate $P(X = v | A \equiv B)$ considering its frequency in $\mathcal{D}_A \cup \mathcal{D}_B$:

$$P(X = v | A \equiv B) = f(v, \mathcal{D}_A \cup \mathcal{D}_B) \tag{4.10}$$

To compute $P(s.A = v' | X = v, A \equiv B)$ we need to distinguish whether $t.A$ is correct (it equals the value of the random variable X) or wrong.

An attribute can provide a wrong value because of an error by the source or because of a linkage error. In our model, we can consider linkage errors as a special case of source errors (the source provides a wrong association of the identifier with the specification). Therefore, we assume that each attribute has the same error probability ϵ for every observation and that, in case of error, the attribute provides a random value from those of the domain of the property, which is estimated by $\mathcal{D}_A \cup \mathcal{D}_B$:

$$P(s.A = v' | A \equiv B, X = v) = \begin{cases} 1 - \epsilon + \epsilon f(v, \mathcal{D}_A \cup \mathcal{D}_B) & [v' = v] \\ \epsilon f(v, \mathcal{D}_A \cup \mathcal{D}_B) & [v' \neq v] \end{cases} \tag{4.11}$$

Notice the $+\epsilon f(v, \mathcal{D}_A \cup \mathcal{D}_B)$ term in the first row: the random value provided in case of error may be the correct value by chance. This assumption models the fact that frequent values are less likely to be mistaken.

By replacing Equations 4.10 and 4.11 in Equation 4.9:

$$P(s.A = v_1, t.B = v_2, v_1 \neq v_2 | A \equiv B) = \\ \epsilon(2-\epsilon) f(v_1, \mathcal{D}_A \cup \mathcal{D}_B) f(v_2, \mathcal{D}_A \cup \mathcal{D}_B) \tag{4.12}$$

Notice that, if $\epsilon = 0$ (perfect sources on these attributes), the above probability equals 0, i.e., equivalent attributes from two linked specifications cannot provide different values.

Let us now consider the case in which $s.A$ equals $t.B$. Either they are correctly providing the actual value of X (the sources are not making errors), or both of them are wrong and they are assuming the same value by chance.

$$P(s.A = t.B = v | A \equiv B) = \\ P(s.A = t.B = X = v | A \equiv B) + \\ P(s.A = t.B = v, X \neq v | A \equiv B) \tag{4.13}$$

By the conditional probability definition, the probability of the first term is given by the probability that the correct value for the property described by the attributes is v multiplied by the probabilities that A and B are correctly providing v, that is: $P(X = v | A \equiv B) P(s.A = v | X = v, A \equiv B) P(t.B = v | X = v, A \equiv B)$. Similarly, the second term results: $P(X \neq v | A \equiv B) P(s.A = v | X \neq v, A \equiv B) P(t.B = v | X \neq v, A \equiv B)$.

By applying Equations 4.10 and 4.11, we obtain:

$$P(s.A = t.B = v | A \equiv B) = \\ f(v, \mathcal{D}_A \cup \mathcal{D}_B)(1 - \epsilon(2-\epsilon)(1 - f(v, \mathcal{D}_A \cup \mathcal{D}_B)) \tag{4.14}$$

Notice that if the sources are perfect ($\epsilon = 0$), the above probability equals $f(v, \mathcal{D}_A \cup \mathcal{D}_B)$, i.e., it corresponds to the frequency of the value (which is estimated by the union of the domains). Observe that negative posterior is a special case of positive, i.e., if we apply same model we get the same results.

Example 2. In Example 1 we computed prior probability for source attribute pairs $S3$.`Memory`, $S4$.`Memory` and $S4$.`Memory`, $S5$.`Megapixels` from our running example in Figure 4.1. Now we develop the computation for posterior probability and final matching score. Table 4.2 shows the conditioned probabilities (positive and negative) between

these pairs of attributes, using the Equations 4.7 and 4.14, and the domain frequencies.

$S3.$ `Memory`	$S4.$ `Memory`	Positive posterior	Negative posterior
16	16	$0.57 * (1 - 0.1(2 - 0.1)(1 - 0.57)) =$ **0.52**	$0.5 * 0.66 =$ **0.33**
16	16	**0.52** (as above)	**0.33** (as above)
8	8	$0.29 * (1 - 0.1(2 - 0.1)(1 - 0.29) =$ **0.25**	$0.25 * 0.33 =$ **0.08**
$S4.$ `Memory`	$S5.$ `Mpixels`	Positive posterior	Negative posterior
16	16	$0.66 * (1 - 0.1(2 - 0.1)(1 - 0.66)) =$ **0.62**	$0.66 * 0.66 =$ **0.44**
8	16	$0.33 * 0.66 * 0.1 * (2 - 0.1) =$ **0.041**	$0.33 * 0.66 =$ **0.22**
16	8	$0.66 * 0.33 * 0.1 * (2 - 0.1) =$ **0.041**	$0.66 * 0.33 =$ **0.22**

Table 4.2: Posterior values for Example 2.

We can now compute the final score, using Equations 4.5, and 4.1. We omit the subscript in L_{AB} for simplicity.

$$P(L|S3.\texttt{Memory} \equiv S4.\texttt{Memory}) = 0.52 * 0.52 * 0.25 = 0.068$$

$$P(L|S3.\texttt{Memory} \neq S4.\texttt{Memory}) = 0.33 * 0.33 * 0.08 = 0.0087$$

$$P(S3.\texttt{Memory} \equiv S4.\texttt{Memory}|L) = \frac{0.068 * 0.45}{0.068 * 0.45 + 0.0087 * 0.55} \approx \mathbf{0.86}$$

$$P(L|S4.\texttt{Memory} \equiv S5.\texttt{Mpixels}) = 0.62 * 0.041 * 0.041 = 0.0010$$

$$P(L|S4.\texttt{Memory} \neq S5.\texttt{Mpixels}) = 0.44 * 0.22 * 0.22 = 0.0021$$

$$P(S4.\texttt{Memory} \equiv S5.\texttt{Mpixels}|L) = \frac{0.0010 * 0.66}{0.0010 * 0.66 + 0.0021 * 0.33} \approx \mathbf{0.48}$$

4.4.2 Approximate Match

In order to be more tolerant in comparison, values are pre-processed as follows: *(i)* tokens are split at each number-letter and uppercase-lowercase transition *(ii)* sequences of non-alphanumeric character are replaced with a single whitespace, except for commas and dots in numeric sequences *(iii)* accents and diacritics are removed *(iv)* uppercase letters are converted to lowercase *(v)* values are converted to an set of tokens (e.g.: "12MP frontCamera-11.5MP rearCamera" $\rightarrow \{12, MP, front, camera, 11.5, rear\}$).

While computing the similarity score (Section 4.4.1) we consider two values as equivalent if Jaccard similarity between their tokens is greater than a given threshold.[6]

4.5 Instance Level Alignment

We exploit the results of the source attribute matching step in order to identify forms of heterogeneity among the attributes and extract values that can give rise to more locally homogeneous attributes.

Our approach relies on the assumption that the source attribute matching step produces clusters of attributes that are mostly homogeneous: it is unlikely that linked instances share similar values on the same attributes by chance across different sources. We consider the union of the domains of the source attributes grouped by each (non-singleton) cluster as a *dictionary* of values for the product property represented by the attributes in the cluster. We associate each cluster with a label, and we tag every value that contains a term from a given dictionary with the label of the corresponding cluster. If many values from a given source attribute are tagged by terms of a given dictionary, it is plausible that they have the same semantics of the source attributes grouped in the cluster corresponding to such a dictionary. Then, we extract the tagged strings of these values and create a *virtual source attribute*.

It is worth observing that even if virtual source attributes contain terms of a given dictionary, we cannot conclude that they are semantically equivalent to the attributes

[6] We have set the threshold at 0.9.

in the cluster from which the dictionary has been generated: they may have similar domains but different meanings. However, we can rely on the solid Bayesian analysis of the source attribute matching phase to validate the equivalence between a virtual attribute and the existing clusters. Interestingly, linked instances that did not match because of heterogeneous, mixed values or noisy values, once the values have been extracted in a virtual attribute may match, creating larger clusters.

The above observations motivate our iterative approach: virtual attributes can produce larger clusters, larger clusters can improve dictionaries giving rise to new virtual attributes.

4.5.1 Tagging and Virtual Attributes Extraction

The first phase for the extraction of virtual attributes consists in creating dictionaries of values for the clusters of source attributes obtained by the Bayesian analysis. We associate each non-singleton cluster $c \in \{c \in \mathbf{C}, |c| > 1\}$ with a *dictionary*, denoted $\mathcal{D}_c$, containing the values of its attributes: $\mathcal{D}_c = \bigcup_{A \in c} \mathcal{D}_A$. For efficiency, we exclude very long values, as they usually correspond to noisy or mixed attributes. Also, we filter out values that are present in many clusters, as they do not characterize the domain. Typically these values have generic meanings (such as, "Yes", "No", "Not available") that can apply to many attributes, even with completely different semantics. Specifically, we drop values with more than 25 characters, and values that are present in more than 10% of the clusters.

The dictionaries associated with the clusters are used to tag all the values in the dataset. We associate a label l_c with each cluster c, and we tag with l_c every string (sequence of tokens) contained in any attribute value $\mathtt{A} \notin c$ that matches a term in $\mathcal{D}_c$. If a source attribute `A` contains at least two values that have been tagged with the same label l_c, we extract the tagged strings and use them as values for a new *virtual attribute*, whose name is denoted `#A#c`.

It is worth observing that a given value could be tagged by many labels (because terms from different dictionaries match with different portions of the value). Whenever

a tagged string is contained by another tagged string, we consider more reliable the match with the larger term, and hence we drop the label of the smaller tagged string.

Some virtual attributes could be very similar (possibly identical) to the source attribute from which they have been extracted. Such virtual attributes are useless, as they would play the same role as the originating attribute. Therefore we do not consider virtual attributes that are too similar to the originating source attribute. In particular, we eliminate virtual attributes whose domain has Jaccard similarity with the domain of the original attribute higher than 80%.

Example 1. In our running example of Figure 4.4 after the matching step we have the following non-singleton clusters:

$$c_1 = \{S1.\texttt{Battery chemistry}, S2.\texttt{B type}\}$$
$$c_2 = \{S1.\texttt{Battery model}, S4.\texttt{Battery}\}$$
$$c_3 = \{S1.\texttt{CPU}, S2.\texttt{IPU}, S1.\texttt{Processor}\}$$
$$c_4 = \{S3.\texttt{Memory}, S4.\texttt{Memory}\}$$
$$c_5 = \{S1.\texttt{Video Resolution}, S2.\texttt{Video Format}\}$$

whose dictionaries result as follows:

$$\mathcal{D}_{c_1} = \{\text{Li-Ion, Ni-MH}\}, \mathcal{D}_{c_2} = \{\text{NP-BN1, FNB83}\}$$
$$\mathcal{D}_{c_3} = \{\text{BionzX, Xpeed3, Xpeed2}\}$$
$$\mathcal{D}_{c_4} = \{\text{8, 16, 32}\}, \mathcal{D}_{c_5} = \{\text{1024x768,1920x2014}\}$$

Figure 4.4 shows tagged values and the virtual attributes created accordingly.

In source attribute S_3.`Battery` several values were tagged by terms in $\mathcal{D}_{c_1}$ and $\mathcal{D}_{c_2}$, leading to the creation of virtual attributes `#Battery#c1` and `#Battery#c2`, and hence identifying the local homonym of the source attribute S_3.`Battery`. Tagging values of S_5.`Batt`, which suffers from representation heterogeneity, with terms in $\mathcal{D}_{c_1}$ allows the creation of the virtual attributes `#Batt#c1`, which are cleansed from uninformative pieces of text. No virtual attribute is created with elements of cluster c_4 tagged in

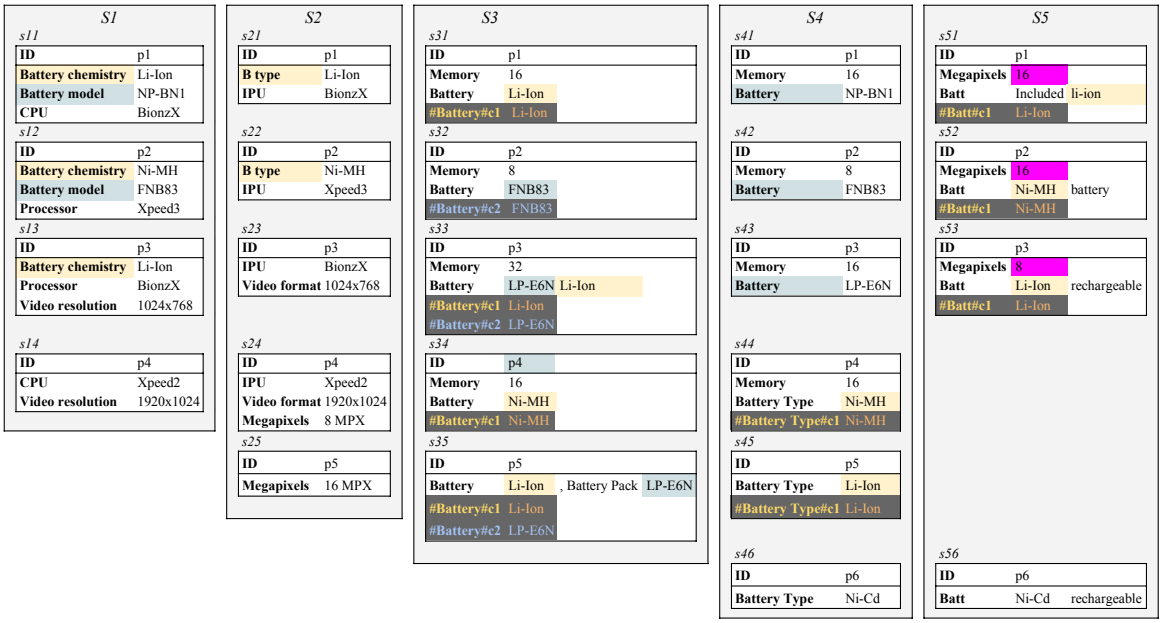

Figure 4.4: Results of tagging and virtual attribute extraction.

S_5.`Megapixels`, as it is too similar to the original attribute (the domains are identical). Notice attribute S_4.`Battery Type`: it was not part of cluster c_1, as it does not have enough linkage for matching. However, its values in s_{44} and s_{45} were tagged with terms from $\mathcal{D}_{c_1}$, giving rise to the virtual attribute `#Battery Type#c1`.

4.5.2 Iterating Matching and Tagging

The source attribute matching and the virtual attribute extraction steps are launched iteratively, as one provides new evidence that can be exploited by the other. Algorithm 2 reports the pseudo-code of the iterative step, which takes as input a set of sources $\mathbf{S}$, and produces as output (i) a set of sources $\mathbf{S}'$ augmented with virtual attributes and (ii) a set of clusters $\mathbf{C}$ of its source attributes (where each cluster can include also virtual attributes).

Initially (line 1) the matching step is launched and produces as output a set $\mathbf{C}$ of source attribute clusters. This clustering along with the full dataset is provided as input to the dictionary creation step (line 2), which returns a dictionary for each cluster.

Then the iteration starts (line 3). The dictionaries are exploited by the tagging step that produces a new version of the dataset, $\mathbf{S}'$, introducing virtual attributes, as described in Section 4.5.1. The matching algorithm processes $\mathbf{S}'$ from which it generates (from scratch) new clusters (line 5) and the associated dictionaries (line 7). These steps are repeated until the dictionaries do not change anymore:[7] as no additional evidence arises the iteration converges.

It is important to observe that the tagging step (line 4) is always done on the original version of dataset, $\mathbf{S}$, which does not include any virtual attribute, but using the latest version of the dictionaries, which are created after each matching step. In this way, at every iteration the dictionaries accumulate knowledge about the domain of each cluster; the tagging step takes advantage of the enhanced dictionaries to tag more values, giving rise to more accurate virtual attributes, which trigger new attribute matches.

Algorithm 2: Matching and tagging iteration

Input : a set $\mathbf{S}$ of sources
Output: a set $\mathbf{S}'$ of sources augmented by virtual attributes, a set $\mathbf{C}$ of source attribute clusters (computed over $\mathbf{S}'$)

1 $\mathbf{C} \leftarrow sourceAttributeMatching(\mathbf{S})$;
2 $\mathbf{D} \leftarrow \{\mathcal{D}_c, c \in \mathbf{C}, |c| > 1\}$;
3 **do**
4 $\quad \mathbf{S}' \leftarrow tagging(\mathbf{D}, \mathbf{S})$ //adds virtual attributes to $\mathbf{S}'$;
5 $\quad \mathbf{C} \leftarrow sourceAttributeMatching(\mathbf{S}')$;
6 $\quad \mathbf{D}_{prev} \leftarrow \mathbf{D}$;
7 $\quad \mathbf{D} \leftarrow \{\mathcal{D}_c, c \in \mathbf{C}, |c| > 1\}$;
8 **while** $\mathbf{D} \neq \mathbf{D}_{prev}$;
9 **return** $\mathbf{C}$, $\mathbf{S}'$;

Example 2. To illustrate the interaction between tagging and matching let us continue Example 1, where we showed the results of the first tagging step, which created the virtual attributes shown in Figure 4.4. The successive matching step (which is inside the iteration) produces clusters c_3, c_4 and c_5 identical to the previous call, while clusters

[7] Ratio of values removed or added to each cluster domain must be less than 10%.

c_1 and c_2 change, as the virtual attributes allow new matches:

$$c_1 = \{S1.\texttt{Battery Chemistry}, S2.\texttt{B type},$$
$$S3.\texttt{\#Battery\#c1}, S4.\texttt{Battery Type},$$
$$S4.\texttt{\#Battery Type\#c1}, S5.\texttt{\#Batt\#c1}\}$$
$$c_2 = \{S1.\texttt{Battery Model}, S3.\texttt{\#Battery\#c2}, S4.\texttt{Battery}\}$$

Notice the attribute $S4.\texttt{Battery Type}$, which remained isolated in the previous matching step because of lack of linkage (s_{35} and s_{45} are in linkage, but the values of $s_{35}.\texttt{Battery}$ and $s_{45}.\texttt{Battery Type}$ do not match, because of the mixed values in $s_{35}.\texttt{Battery}$). $S4.\texttt{Battery Type}$ has now enough linkage and matching with the virtual attribute $S3.\texttt{\#Battery\#c1}$, and therefore it is has been included in the same cluster.

It is important to observe that the presence of $S4.\texttt{Battery Type}$ contributes to improve the dictionary associated with c_1 with an additional value, "Ni-Cd". This has a positive impact in the successive tagging step (which occurs in the next iteration) as the dictionary enhanced with such a value allows the creation of a virtual attribute, $S5.\texttt{\#Batt\#c1}$, that includes also the tagged value ("Ni-Cd") extracted from $s_{56}.\texttt{Batt}$. With the new dictionary, attribute $s4.\texttt{\#Battery\#c1}$ is not created anymore, as it would be identical to $s4.\texttt{Battery}$.

4.5.3 Instance-Level Clustering

Before building the final instance-level clusters, we perform astep which aims at merging the clusters obtained by the iterations based on features related to *global* homogeneity that occur among the sources.

We observe that the clusters obtained by the iterative approach contain attributes with reduced heterogeneity. Then, we can additionally exploit classical schema matching strategies, based on attribute names and domain similarity, in order to detect matches that could not be found before, because of lack of linkage, or due to strong forms of representational heterogeneity that prevent attribute matching.

Cluster pairs that have at least a pair of source attributes with the same name are selected as candidates for merging. Even a singleton cluster can be selected as one element of the pair (or both), provided that no virtual attribute has been generated from it. Indeed, if a source attribute did not match with any other, and portions of its values gave rise to one or more virtual attributes, this is a strong sign that the attribute could be locally heterogeneous: we do not want it to be merged to other clusters in its original form. Also, clusters cannot be merged if they include source attributes that have at least one pair of instances in the same specification.

For each cluster in a candidate pair $\{c_a, c_b\}$, we compute the set of all tokens present in values of all source attributes: $\{t_a, t_b\}$. If $\frac{|T_a \cap T_b|}{min(|T_a|,|T_b|)} > 0.8$, clusters are merged. This formula is the maximum Jaccard Containment [48] with regard to each of the two sets. Using tokens allows merging of attributes with noisy values.

We are now able to group instance-level attributes sharing common concepts. As most of the complexity and heterogeneity of attributes have been isolated and removed, thanks to virtual attributes, instance-level clusters can be built directly from the clusters.

For each non-singleton cluster $c \in \mathbf{C}$, each occurrence of its non-virtual attributes is added to an instance-level cluster. For each occurrence of its virtual attributes, we add to the cluster the correspondent original attribute instance, with the original value. The rationale for this choice is that virtual attributes are just an internal tool for matching, and should not be exposed as a result of the algorithm. The goal of RaF-AIA is to detect matching between attribute instances and not extract the specific value, which is outside the scope of this work.

Example 3. Isolated attributes $S2$.`Megapixels` and $S5$.`Megapixels` are selected for merging. Their token sets are, respectively, $\{8, 16, MPX\}$ and $\{8, 16\}$. Tokens from the second attribute are all contained in first, so their similarity is 1, and the two attributes are merged in a single cluster. $S3$.`Battery` and $S4$.`Battery` are not selected because $S3$.`Battery` is an isolated source attributes, and two virtual attribute were created from it.

The algorithm now iterates over each non-singleton cluster in **C** to build a correspondent instance-level cluster. Consider the cluster c_1: the attributes $S1$.`Battery Chemistry`, $S2$.`B type` and $S4$.`Battery Type` are not virtual, so we simply add their attribute instances to the result. Attributes $S3$.`#Battery#c1` and $S5$.`#Batt#c1` are virtual, so for each of their occurrences we add to the instance-level cluster their corresponding original attribute occurrence. This corresponds to the instances in which the property they refer to occurs. Indeed, we won't add $s32$.`Battery`, containing only the model of the battery. The final instance-level cluster c_{i1} would be:

$$
\begin{aligned}
c_{i1} = \{&s11.\texttt{Battery Chemistry}, s12.\texttt{Battery Chemistry},\\
&s13.\texttt{Battery Chemistry}, s21.\texttt{B Type}, s22.\texttt{B Type},\\
&s31.\texttt{Battery}, s33.\texttt{Battery}, s34.\texttt{Battery}, s35.\texttt{Battery},\\
&s44.\texttt{Battery Type}, s45.\texttt{Battery Type},\\
&s46.\texttt{Battery Type}, s51.\texttt{Batt}, s52.\texttt{Batt}, s53.\texttt{Batt}\}
\end{aligned}
$$

Figure 4.2 shows a representation for a portions of the results (for two products): data are grouped by product `ID` (the figure shows only products $p1$ and $p3$), and a property, $property_k$, is created for each cluster c_{ik}.

Chapter 5

Dataset and Ground Truth Construction

Several tasks were needed to measure performances and effectiveness of RaF-AIA against real-world data.

- **Building an input dataset of product specifications**. This dataset must be based on real data from the Web, extracted with reliable techniques. It should cover main challenges that we can found in real-world data, including errors, unreliable and conflicting data, and heterogeneity.
- **Building a record linkage**. Most products have an associated identifier, that makes it possible to obtain a record linkage before attribute alignment. Still it remains a challenging step, as not all sources provide this identifier, and not in the same way, making it easily mistakable with identifiers of related and/or associated products [25].
- **Building a valid ground truth for instance-level alignment**. The task is quite complex and requires extensive manual efforts to align single attributes in specifications, in a big and heterogeneous dataset.

Along with the choice of a ground truth, some measures must be defined to give a numerical evaluation to the result of the alignment task.

We developed these steps in the context of the ALASKA benchmark,[1] a broader project whose purpose is the definition of a benchmark for data integration [49]. The ALASKA Benchmark has been conceived for evaluating performances of data integration and knowledge graph augmentation tasks. It is a benchmark in continuous evolution, aimed at being general and useful for the research communities that contribute to improve data integration and knowledge graph population. The benchmark focuses on the product domain, a challenging scenario for developing and evaluating big data integration solutions, as we discussed in Chapter 2.

In this Chapter we describe how we have built the dataset and the ground truth. In Chapter 6 we will present the experiments that we carried out, that are based on ALASKA dataset and ground truth but are independently conducted.

5.1 Dataset construction

The first task to build a dataset is to find sources with the kind of desired resources: in our case, websites with product pages. It corresponds to the step called *Source Discovery* in the pipeline we discussed in Chapter 1.

For this goal, we exploited the *Dexter*[2] research project [17] that provided us a good starting point. Dexter is designed to discover and extract product specifications on the web. Part of Dexter results is the discovery and selection of data sources.

Figure 5.1 shows the number of data sources discovered by Dexter and the relative number of product pages that it crawled. The volume of data emerges in a visible way, evidences about the heterogeneity of the dataset are described in Section 5.3. The pages the Figure refers to are HTML pages of products of the associated domain (or *vertical*). Dexter also provided the URLs of the crawled pages.

[1] www.di2kg.inf.uniroma3.it
[2] https://github.com/disheng/DEXTER

cat.	# sites	# pages
camera	1,107	278k
cutlery	339	103k
headphone	475	142k
monitor	941	184k
notebook	589	272k
software	419	129k
sunglass	471	182k
toilets	82	36k
tv	841	132k
all	3,005	1.46M

Figure 5.1: Dexter discovered sites and crawled pages

vertical	# sources	# pages with exctracted specs	# distinct attribute names
camera	1083	94710	38663
notebook	577	54852	14180
headphone	464	49975	7574
monitor	929	48067	14594
shoes	131	23481	2489
tv	826	21835	14959
software	372	12626	6332
toilets	81	5319	1464
sunglasses	386	4871	1137
cutlery	294	3302	1742

Figure 5.2: Dexter discovered sites and crawled pages

We chose the camera vertical, as it has the largest number of sources and web pages and we want volume and variety to be major players in our dataset. For each website of our target category, we downloaded the associated HTML pages at the URLs collected by Dexter, and an extraction tool, called *Carbonara Extractor*, was run on the pages. Carbonara produces a set of $< attribute\_name, attribute\_value >$ pairs from each input page: such a set represents the specifications of the product described in the page. The output of the tool is a JSON object and, in addition to the successful key value pairs extracted, it also contains a special attribute that represents the HTML page title.

The Carbonara extraction tool is explained in Section 5.2.

Figure 5.2 shows the results of the extractions obtained by Carbonara: reduced

numbers in this Figure with respect to those in Figure 5.1 are due to broken URLs of pages or unsuccessful extraction process by the extraction tool.

The source discovery process is followed by a data selection process. The goal for the first version of ALASKA Benchmark was to choose 25 sources. Such a goal was driven by the following criteria:

- obtaining enough amount of information from data sources in terms of specifications of products;
- avoiding copiers and clones;
- maximizing overlapping sources in terms of shared real-world entities.

Discovered data sources and the provided data (the JSON files of extraction) were filtered based on what we called *3-3-100 filtering*: it was used to reduce noise, i.e. fake attributes that came from the extraction process, and to focus on larger data sources. The 3-3-100 filtering works as follows:

1. key-value pairs with an attribute name that is not present in at least 3 pages of the same source were filtered from the extraction files they belong to (not considering the attributes "<page title>" and "__unstructured" attributes);
2. after that, extraction files with less than 3 key-value pairs were kept out;
3. finally, only data sources with at least 100 files of extraction made it through the filtering.

Figure 5.3 shows statistics about data sources after the first two steps of 3-3-100 filtering: in particular, for each source, the number of resulting extraction files ("# urls 3-3-100"), the number of distinct attribute names ("# distinct attribute names 3-3-100"), the average number of key-value pairs in extraction files ("avg attribute names 3-3-100") and the average number of unsuccessful key-value pairs extraction per extraction file ("avg unstructured values 3-3-100").

site	# urls 3-3-100	# attribute names 3-3-100	# distinct attribute names 3-3-100	avg attribute names 3-3-100	avg unstructured values 3-3-100
www.ebay.com	14274	138233	1987	9.68	8.13
www.ebay.ca	8245	86788	1478	10.53	9.15
www.alibaba.com	7972	180766	1590	22.68	0.81
www.ebay.co.uk	3614	36536	678	10.11	10.75
alibaba.com	2568	25802	594	10.05	0.92
www.ebay.in	1940	63554	1147	32.76	12.31
www.ebay.com.sg	1899	23297	734	12.27	11.04
uae.souq.com	1380	19213	81	13.92	13.94
www.ebay.ie	1098	13764	338	12.54	8.57
www.gosale.com	1002	11416	21	11.39	7.26
www.buzzillions.com	832	28748	259	34.55	23.23
search.greenvilleadvocate.com	785	8899	404	11.34	7.47
www.lelong.com.my	755	12824	791	16.99	5.96
www.priceme.co.nz	740	7485	19	10.11	0
shopping.dealtime.com	736	14254	80	19.37	0.26
www.shopping.com	706	13902	80	19.69	0.26
www.priceme.com.my	701	7021	19	10.02	0
www.productreview.com.au	690	4830	7	7	0
www.exporters.sg	686	4352	55	6.34	0
www.shopmania.in	630	33929	80	53.86	10.63
search.tryondailybulletin.com	591	6078	259	10.28	8.7

Figure 5.3: Results of 3-3-100 filtering

As a result from the 3-3-100 filtering process, 63 sources were selected. We discarded sources which were versions of another source in a different country (e.g. www.ebay.com vs www.ebay.ca), and sources which contained pages cloned from other sources. In all these cases we kept the largest source.

As the final objective is integrating data of instances which refer to the same real-world entity, one of the criteria the data source selection was based on is the amount of overlapping entities shared by the selected data sources themselves. Since this kind of data was not at our disposal, the results of [25] were extremely useful. In that paper, the authors take advantage of the opportunity that sources publish product identifiers and develop a technique to extract them.

We used the identifiers extracted by applying [25] as partial record linkage matching the instances of products from the web pages of the dataset to compute the numbers of overlapping entities, shared by the initial 63 sources, which we identified as result of the 3-3-100 filtering process.

In particular we developed a greedy approach to select a set of sources that max-

```
 1 ebay.com        7294
 2 gosale.com      224
 3 shopbot.com.au  194
 4 buzzillions.com 157
 5 price-hunt.com  127
 6 flipkart.com    123
 7 cammarkt.com    130
 8 priceme.co.nz   133
 9 mysimon.com     127
10 pricedekho.com  121
11 testseek.com    125
12 alibaba.com     113
13 shopmania.in    100
14 henrys.com      95
15 eglobalcentral.co.uk    96
16 mypriceindia.com        91
17 buy.net 78
18 everyrule.com   79
19 garricks.com.au 76
20 camerafarm.com.au       75
21 pcconnection.com        71
22 walmart.com     64
23 wallmartphoto.com       100
24 yellowpages.observer-reporter.com       56
25 search.greenvilleadvocate.com   120
26 search.tryondailybulletin.com   152
27 cambuy.com.au   52
28 wexphotographic.com     54
29 ukdigitalcameras.co.uk  117
30 ilgs.net        79
31 canon-europe.com        53
32 bdstall.com     51
33 submarino.com.br        52
34 priceme.com.my  52
35 miniprice.ca    43
36 pe.traetelo.com 43
37 pricefalls.com  42
38 bhphotovideo.com        41
39 frys.com        39
40 mini-price.ca   38
41 woolworths.co.uk        38
42 offeroftheday.co.uk     58
43 currys.co.uk    55
44 ecoustics.com   37
45 smarthome.com   33
46 capitalstores.co.uk     30
47 homestore.com.ua        28
48 fnac.com        26
49 homedepot.com   25
50 exporters.sg    24
51 uae.souq.com    24
52 amazon.com      21
53 officedepot.com 21
54 amazmart.com    20
55 awok.com        20
56 lelong.com.my   19
57 productreview.com.au    18
58 eclipsecomputers.com    17
59 assets4.tiendaclic.mx   15
60 camerahouse.com.au      15
61 etronicsexpress.com     13
62 flexshopper.com 12
63 tootoo.com      10
```

Figure 5.4: Data sources with overlapping entities

imizes the overall overlap of the products. The algorithm is inizialized by taking the largest source and adding it to the (empty) set of set of the selected sources. Then the algorithm iterates to add the source with largest overlap with the set of the selected sources. Figure 5.4 exemplifies the the algorithm. Each row reports the name of a data source (on the left) and a number (on the right). Row 1 is a special case and reports the largest source (ebay.com) and the number of instances that have that match (according to the identifier) with the all other sources. For all the rows except the first one, the number on the right corresponds to the number of instances shared with all the sources

that appear in previous lines. For example, consider line 4: buzzillions.com has 157 instances shared with ebay.com, gosale.com and shopbot.com.au. Based on the overlap estimated in this way, we selected the sources with the largest overlap.

We decided to select the first 30 sources. However, after a quality check, some sources were discarded because they provided poor data (they are mainly result pages of search engines). The final version of the dataset consists of 24 sources for a total of 29,787 web pages. The data sources of the final version of the camera dataset are listed in Table 5.1.

To extract product specifications from the pages of the dataset we have developed a specific tool, called Carbonara, which is described in the next Section.

5.2 The Carbonara Extractor

Carbonara [3] is our extraction tool, used for extracting specifications from product pages. It improves and replaces the Dexter extraction system [17] described in Chapter 2.

5.2.0.1 Finding Product Specifications

The main problem faced by Carbonara is how to distinguish between relevant and not relevant pieces of information within a web page.

We observe that the specifications of the main product of the page are in tables and lists. However quite often HTML tables and HTML lists are used for page layout purposes, such as, groups of activity buttons, little dashboards, series of links and so on. It is therefore crucial to understand if the DOM nodes corresponding to tables and lists are rendered in the page to publish specification data on the main product rather than for defining the page layout. This is a tricky issue, due to the characteristics of HTML pages from e-commerce web sites, which are plenty of navigation control bars, menus, account settings, cart information. To this end, we developed supervised machine learning classifiers, based on neural networks, to classify relevant tables and list.

[3] https://github.com/0xNaN/carbonaraextractor

Selected Data Sources
buy.net
cammarkt.com
www.alibaba.com
www.buzzillions.com
www.cambuy.com.au
www.camerafarm.com.au
www.canon-europe.com
www.ebay.com
www.eglobalcentral.co.uk
www.flipkart.com
www.garricks.com.au
www.gosale.com
www.henrys.com
www.ilgs.net
www.mypriceindia.com
www.pcconnection.com
www.pricedekho.com
www.price-hunt.com
www.priceme.co.nz
www.shopbot.com.au
www.shopmania.in
www.ukdigitalcameras.co.uk
www.walmart.com
www.wexphotographic.com

Table 5.1: Data sources in the final version of the camera dataset

5.2.0.2 Training Set Creation

We created a labeled dataset to train the classifiers. The classifiers relies on features extracted from tables and have been trained by a set of manually labeled tables and lists. The training dataset includes both positive and negative examples: tables and lists were selected by following xpath rules and then downloaded with the objective of extracting features from them.

The xpath rules were created manually for twenty web sites. Tables and lists not containing product specifications, thus negative examples for the classifiers, were selected using generic xpath rules for finding all tables and lists of a given web page and

Stem	Frequency
camera	6964
digit	4028
len	2086
video	1969
zoom	1583
product	1401
nikon	1374
canon	1372
price	1357

Table 5.2: Partial result of domain-based relevant words generation process

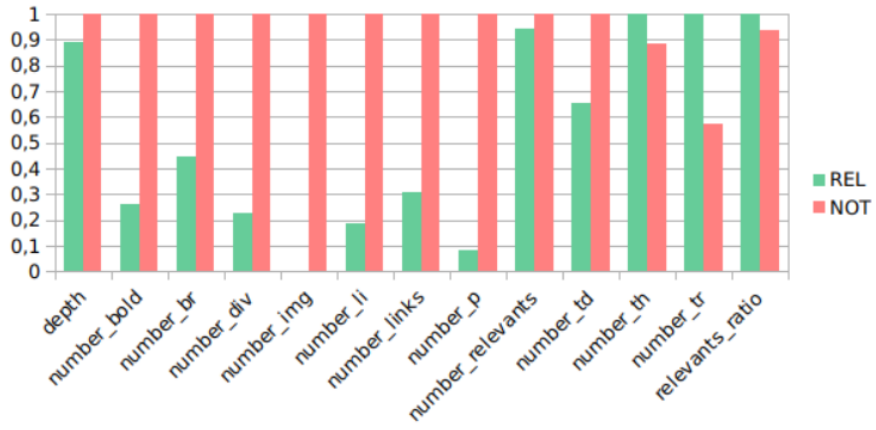

Figure 5.5: Analysis of features for HTML tables

not considering the tables and lists that were already selected by the rules built by hand for positive examples. The underlying intuition is that web pages from the same web site maintain a similar structure and, thus, xpath rules are robust enough to scale on all the pages from the same web site.

Features include both structura (html) properties and domain keywords. About 200 relevant keywords were considered. Table 5.2 shows a partial result of the relevant keywords and the associated frequency.

Figures 5.5 and 5.6 show the average values of features (normalized between 0 and 1) for positive and negative examples (relevant and not relevant tables or lists).

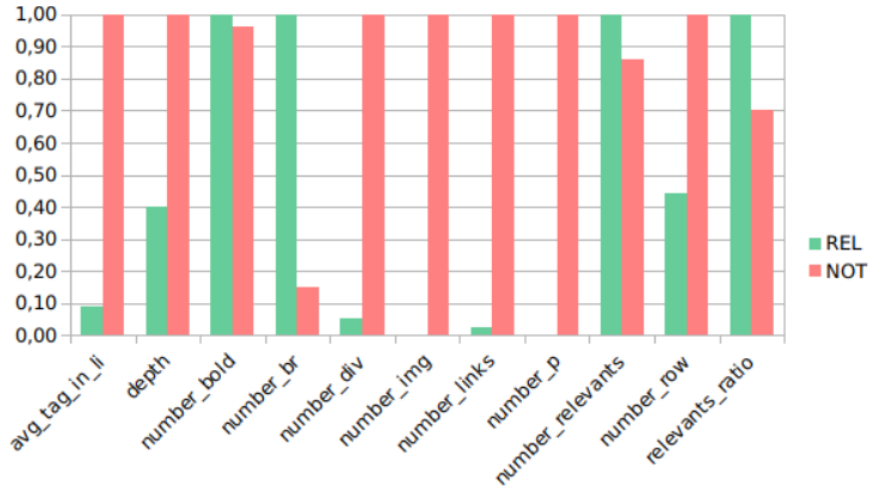

Figure 5.6: Analysis of features for HTML lists

For tables, we can easily observe the importance of some features (such as, *number_bold*, *number_br*, *number_div*, *number_img*, *number_li*, *number_links* and *number_p*) to distinguishing between positive and negative examples. An explanation is given to to the fact that tables are often used as a layout structure, that contains a large number of HTML tags as the ones the features are referring to, such as `<br>`, `<div>`, `<img>` tags, and so on. The higher value for *number_td* is due to the fact that specifications tables are "compact", i.e they have a low number of columns, because usually they just have a key-value pair for each row. The difference detected for *number_tr* is due to the fact that specifications tables typically grow in height as they bring specifications in rows. Unsuccessfully, the domain-based features, depending on relevant words, such as *number_relevants* and *relevants_ratio* are not particularly discriminating in our case: layout tables are usually huge and containing a lot of content, including relevant words too.

For lists the more discriminating features are *avg_tag_in_li*, *depth*, *number_div*, *number_img*, *number_links*, *number_p* and *number_row*, as lists are often used for menus or for layout purposes, for example for galleries, having a big number of links, images, rows filled with other tags and being very long: think about a series of sponsored

products with images of the products themselves and links leading to specification pages that are a very frequent presence in e-commerce web sites.

The training set, that was used to train the classifiers, consisted of about 1,800 training examples.

5.2.0.3 Classifiers Model and Training

Classifiers used for declaring a table o list as containing relevant information, i.e., product specifications, were developed using `Keras`. Two different classifiers were built: one for HTML tables and one for HTML lists. The two classifiers are based on a Neural Network (NN) and share the same architecture and training process, including hyper parameters, which are described in the following.

The model classifier consists of two neural network layers: the former, connected to the input, the latter is the final output layer, two probability scores of the classification, one for probability of the content being relevant and one for it being non-relevant.

A grid search analysis was conducted in order to obtain the best hyper-parameters.The values of loss, accuracy and standard deviation of accuracy were taken into consideration for choosing the best configuration. The grid search was based on configurations with different values for:

- batch size, with values in [8, 16, 32]
- number of neurons of the first layer, with values in [8, 16, 32]
- activation function of the first layer, with values in [“tanh", “sigmoid"]
- number of epochs

During the grid search, a Keras callback, called `EarlyStopping`, was used for terminating the training process when no longer necessary, based on monitoring a given quantity, in our case the value of loss. The best configuration for the training of the classifier was the following: a batch size of 32, 32 neurons in the first layer, and *tanh* as activation function for the first layer. The callback stopped each training process at

a given number of epochs, based on the variation of the value of the loss function; the average number of epochs during the grid search was used as the final number of epochs for the main training of the classifier. During the main training we obtained a loss of 0.13 and 95% accuracy.

Carbonara initializes the features extractor, processes the DOM tree and fetches all the HTML tables and lists from the web page in input and feed them to the right classifier.

The classifiers process the inputs and if the probability is equal or greater than a threshold,[4] the considered node is further analyzed to extract product specifications.

The extraction of products specification is done through the application of an exhaustive set of extracting rules: by manually checking the web pages in our data set, patterns of key-value pairs embedded in HTML code were found, and rules were manually defined in order to extract the name values pairs.

5.3 Dataset profiling

The camera dataset is a very heterogeneous (both intra-source and inter-sources) and challenging dataset. We now describe the dataset in order to emphasize its challenging features.

[4] We have empirically set the threshold value 0.8.

5.3.1 Dataset dimension

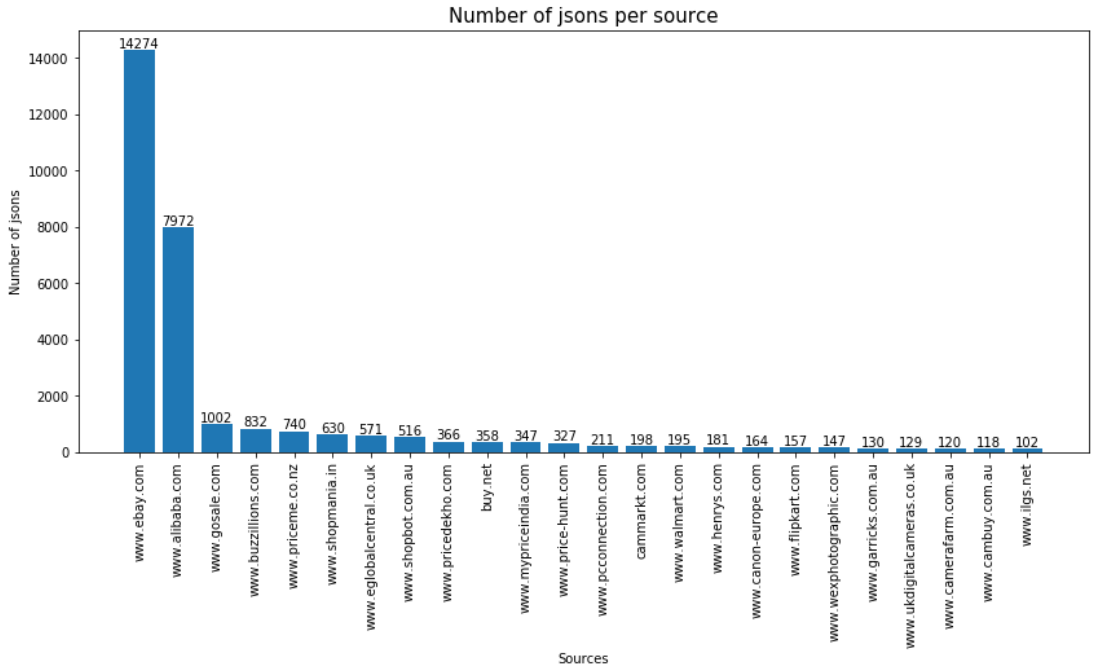

Figure 5.7: Number of JSONs (instances) for each source

Figure 5.7 shows the distribution of the 29,787 specifications extracted from the sources. We have selected two *head sources* for this dataset, that is, sources containing a large number of instances, and 22 *tail sources*, that is, sources containing few instances. Each selected sources contains at least 100 instances, as an effect of the 3-3-100 filtering described above. Figure 5.8, instead, shows the average of attributes which are contained in the specifications of each source.

These numbers give an idea of how complicated the record linkage task can be for the camera dataset, in fact we need to consider that we selected sources also trying to maximize the presence of the same instances on different sources.

Figure 5.9, instead, shows how many distinct attributes are present in each source. With this chart we can estimate how varied the schema of some sources is; for instance, ebay.com contains a large number of distinct attributes: it's not surprising if we consider that this source allows users to post ads and that users can often use different words

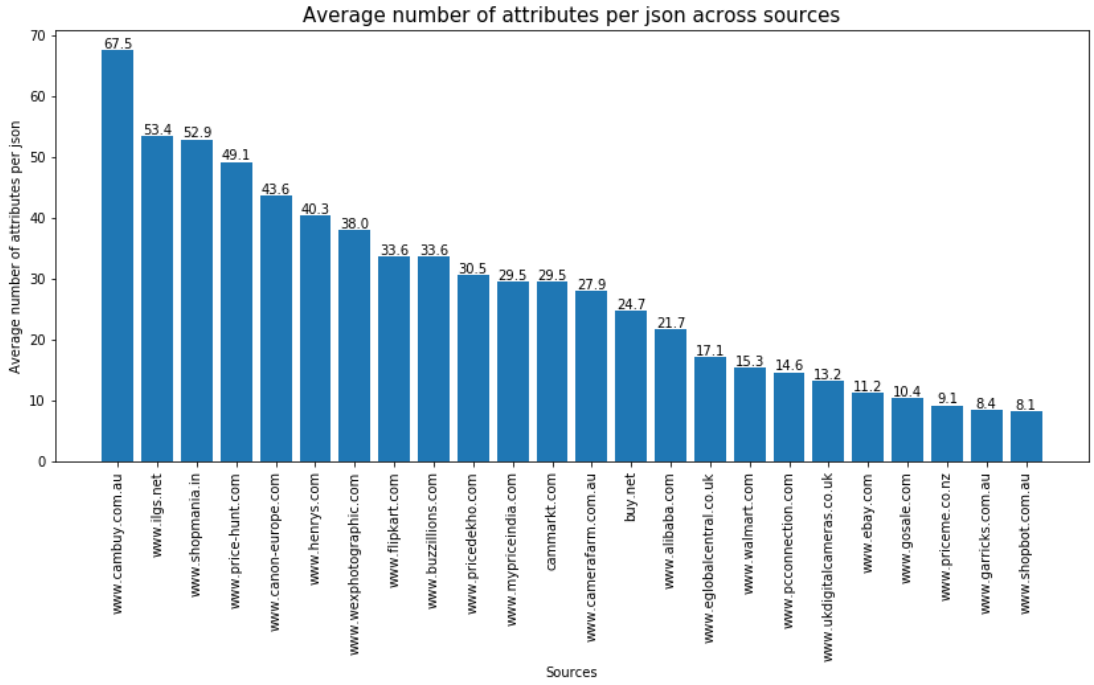

Figure 5.8: Average number of attributes across sources

to describe the same concept. However, also for other e-commerce websites we can see that the number of distinct attributes is high, although it can be assumed that there is someone in charge for data entry. The high number of attributes makes the camera dataset very interesting for the schema alignment task.

5.3.2 Schema heterogeneity

In order to measure schema heterogeneity considering only pre-integration data, we calculated the schema entropy for each source. Schema entropy is defined as follows: consider a source (e.g. www.ebay.com) with m different specifications and n distinct source attributes. Table 5.3 is a bit-wise matrix for the source we are considering: each row shows which source attributes are present in a given specification of extraction from that source. In particular, when we found the value 1 in a cell, it means that the source attribute is present in the specification we are considering, vice versa when we found the value 0 in a cell, it means that the source attribute is not present in the specification.

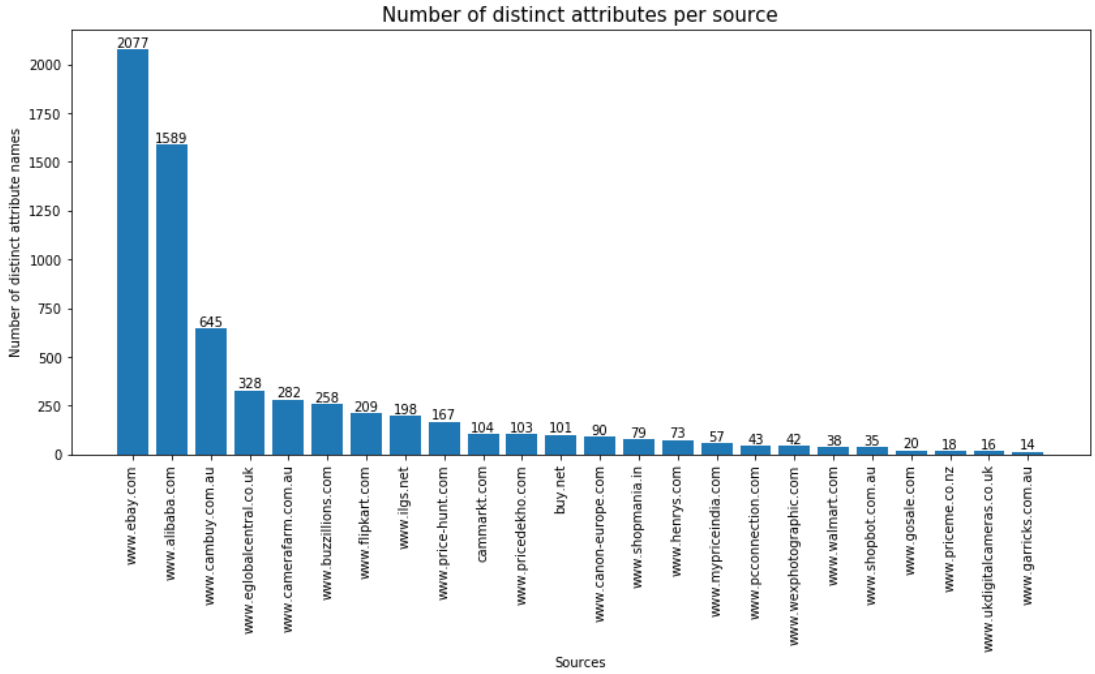

Figure 5.9: Number of distinct attributes for each source

	source_attribute_1	source_attribute_2	...	source_attribute_n
ebay.com/1.json	1	0	...	1
ebay.com/2.json	1	0	...	1
...	...	...	...	...
ebay.com/m.json	0	0	...	1

Table 5.3: Bit-wise matrix for ebay.com

To calculate entropy we use the rows of Table 5.3 as symbol and formula described in equation 5.1. Hence, if the table has all the same rows it means that the source has a fixed schema, so each specification of extraction contains the same set of attributes.

$$H(X) = -\sum_{i=1}^{m} p(x_i) * log_2(p(x_i)) \tag{5.1}$$

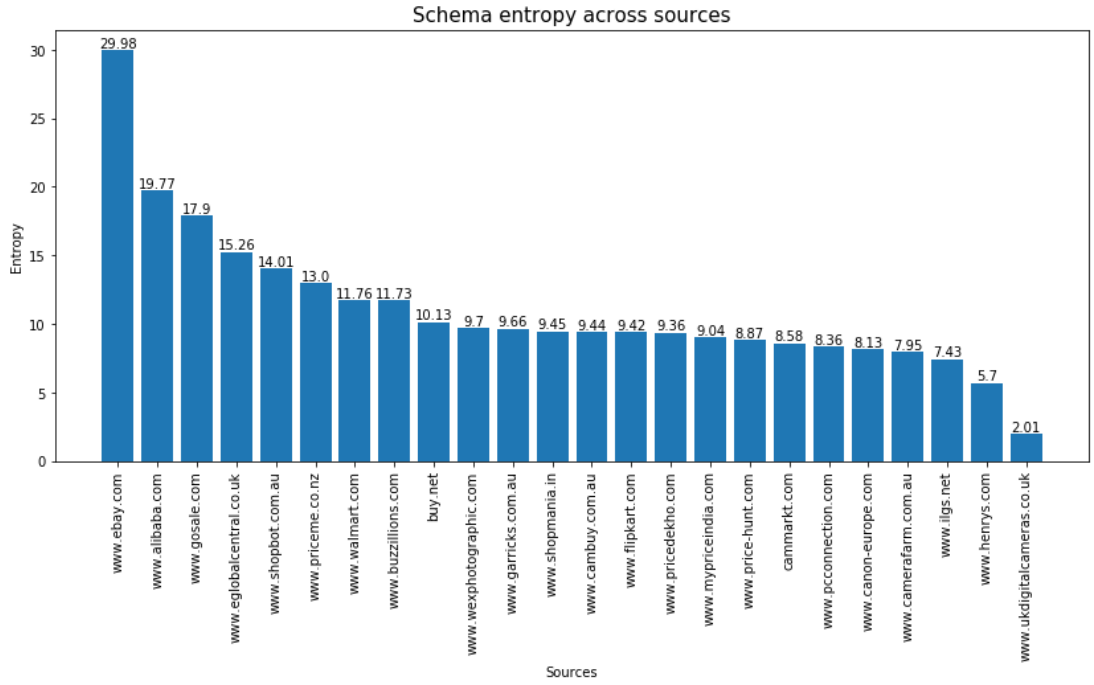

Figure 5.10: Schema entropy for each source

Figure 5.10 shows schema entropy for each source in the camera dataset. Hence, we can notice that `www.ebay.com` seems to be very heterogeneous at schema level, using a very different set of attributes in the different specifications of extraction; `www.ukdigitalcameras.co.uk`, instead, seems to be very regular.

5.3.3 Attribute values heterogeneity

In order to measure attribute values heterogeneity, we calculated the values entropy as follows: consider Table 5.4; each row contains the counter of attribute values for a specific attribute name in the dataset. For instance, in the first row, we have that Nikon is the value associated to attribute *brand* in thirty different specifications of extraction, while Canon is in twenty different specifications of extraction and so on. In that case we are not considering source attributes, so these values can come from different sources.

Attribute name	Values counter
brand	"nikon": 30, "canon": 20, ...
...	...
battery	"aaa": 27, "li-ion": 13, ...

Table 5.4: Attribute values counter for each attribute name

We compute entropy using the entire attribute value as symbol, using equation 5.1. Figure 5.11 shows the aggregate number of attribute names which fall within a certain range of entropy.

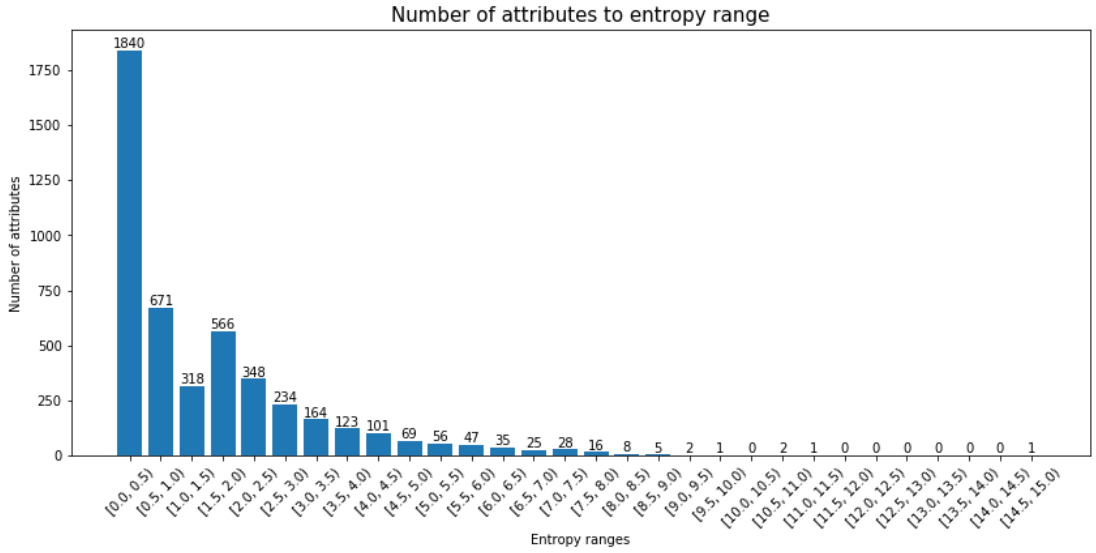

Figure 5.11: Attribute values entropy

The rightmost point, the one with higher entropy, is the $<$ *page title*$>$ attribute, while the leftmost, with lower entropy are mostly attribute names with `yes/no` possible values.

5.4 Ground truth construction

We now introduce the supported Data Integration tasks and our approaches for building the Ground Truth related to each of them.

The ALASKA Benchmark, in the current version, provides a ground truth for evaluating the following fundamental Data Integration tasks:

1. Schema Alignment
2. Record Linkage
3. Instance Level Attribute Alignment

The first two tasks are well recognized integration steps, with a great amount of research work on them, as we discussed in Chapter 1. Instance-level attribute alignment task is the specific problem that RaF-AIA addresses. As we discussed in Sections 4.2 and 4.3.1, while the Schema alignment task aims at creating clusters of equivalent attributes of the *schema* of each source (which we called *source attributes*), the Instance-Level Attribute Alignment works at a finer level, clustering individual attribute instances present in a single specification.

Notice that the Instance-level task has been defined as Knowledge graph augmentation task in last Benchmark version[5], and has a substantial difference to the problem defined by RaF-AIA: we provide to benchmark user a predefined list of "target attributes" (i.e., real world property), along with some example of associations between attribute instances and target attributes, as training data. Users must thus associate all attribute instances with these predefined target attributes, taking advantage of training data. The RaF-AIA approach creates cluster of equivalent attribute instances without needing to know which clusters must be created and which are their characteristics. However, the ground truth construction process for those two problems is the same.

We widely discussed on the heterogeneity problem in Chapter 4, both at inter-source and intra-source level, in particular for what concerns synonyms (attributes that come with different names providing the same information), homonyms (under a given attribute, values not semantically referring to that attribute may appear) and mixed attributes (the value of a given attribute may provide a list of values referring to sub-categories of the attribute, in a descriptive way).

[5] http://di2kg.inf.uniroma3.it/2019

To build a robust ground truth, we need to face semantic ambiguity, diversity in the way instances are represented, errors in terms of veracity of the provided pieces of information, but also in terms of typos, distraction errors, errors due to lack of domain knowledge; or simply absence of complete information, or intentionally omitted data.

For creating the Ground Truth, we adopted a semi-automatic approach. First, we adopted automatic “weak approaches". This was done to make sure that it is actually possible to provide a solution for the several tasks, and also to demonstrate that the tasks are challenging, since weak approaches do provide a solution, but it is usually imperfect and with many errors. In this preliminary steps of the creation process, we aimed at maximizing the recall: for Schema Alignment and Record Linkage tasks this means to create coarse-grained clusters of source attributes and instances, respectively. Then, we exploited human interventions in order to improve the precision.

5.4.1 Building the Ground Truth for Schema Alignment

The Ground Truth for the Schema Alignment task consists of a set of clusters. Each of these clusters has a name, which is the target attribute in the mediated schema the given cluster refers to, and comprehends a set of source attributes, which are mapped semantically to the target attribute. Source attributes are related to a single source; in fact, source attributes with the same name can exist across multiple data sources (for example “dimensions" can be a source attribute for www.alibaba.com and for www.flipkart.com, but they are considered as two separate source attributes). For example, the cluster of the target attribute “resolution", may contain the source attribute “megapixels" from www.garricks.com.au, “mpx" from www.shopmania.in and “pixels" from www.ukdigitalcameras.co.uk, because the values in the JSON files of extraction that come under these attribute names refer to the shared concept of resolution of a digital camera.

The initial goal, once we collected the camera dataset, was to identify 100 distinct target real-world attributes for the camera domain. The idea was to select a variety of both so-called *head* attributes and *tail* attributes. Head attributes are the most frequent

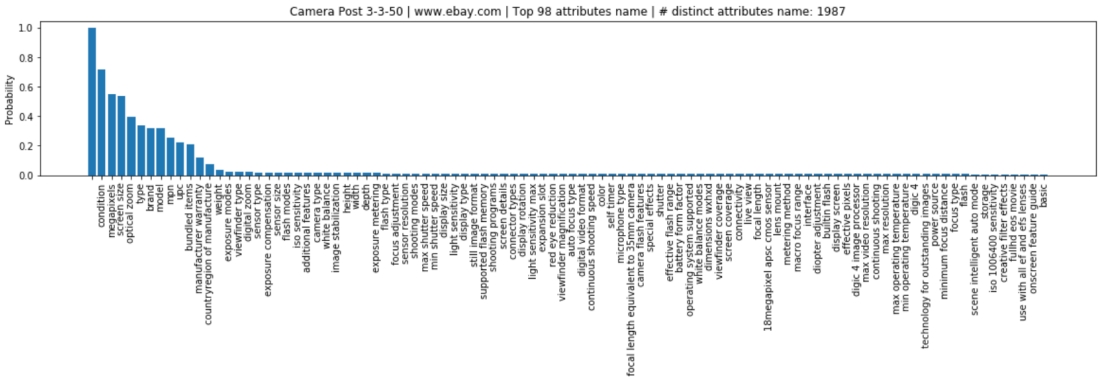

Figure 5.12: Distribution of attributes of the source www.ebay.com

attributes within a single source, while tail attributes are the less frequent ones. The inclusion of both head and tail attributes was due to the objective of equipping the Schema Alignment task with more and less challenging cases, in order to drive the algorithms producing the solutions towards a more complete approach, which considers all the possible cases, and to fuel the competition in this task.

Our first approach to the identification of the 100 attributes was a simple approach: we called it `Top50-Random50`. The approach works as follows: for each data source, select the top 50 most frequent source attributes in terms of number of ispecifications that expose that attribute, and then select another 50 source attributes in a random fashion. From the resulting selection, we wanted to maximize the number of mappings from source schemas to mediated schema, which correspond to the number of source attributes that can be aligned. The selected attributes resulting from this process would have then been analyzed by a human, who has enough domain knowledge (a taxonomist), with the goal of creating a mediated schema with clusters of source attributes. This approach wanted to produce an initial selection, as a bootstrap for real mediated schema creation. Schema alignment, when dealing with a reasonably small number of sources or when the mediated schema is forced to be with a small number of target attributes of interest, is typically done manually [50].

Figure 5.12 shows the distribution by occurrences of all the attributes that the source www.ebay.com exposes through the specification files of extraction produced by

Carbonara Extraction on its web pages. As we can see, the distribution is very skewed, relegating the majority of attributes in a so-called “long tail", which approximately starts from the attribute named “weight".

The trend of this distribution characterizes all the distributions of attributes of all the sources. This kind of distributions penalizes the `Top50-Random50` approach, because limits the probability of selecting source attributes that can be aligned, and thus possible candidates for a cluster of the desired mediated schema. This is due to the following phenomenon: consider a target attribute TA_x consisting of a set of source attributes, then consider that the majority of attributes in source schemas are tail attributes, now it is easy to evaluate that selecting a reasonable amount of source attributes for TA_x is unlikely, as these source attribute need to be picked by the `Random50` selection of `Top50-Random50` methodology. Furthermore, even consider source attributes with the same name, for example “resolution", belonging to separate data sources, they are likely to be part of the same cluster in the Schema Alignment results; what can happen is that “resolution" can be a head attribute in a very small subset of sources and, thus, is selected through the `Top50` selection, but, at the same time, “resolution" is more likely to be a tail attribute in most of the remaining sources, therefore the `Random50` selection should be enough picky to select these attributes in a random way, which is very unlikely.

Unfortunately, this kind of distributions also impacts our approach based on `Stratified Sampling`. The idea was to sample, for each source at our disposal, source attributes from subsets, produced by the partition of source attributes by frequency. The number of subsets, or better called buckets, is a parameter that needs to be defined.

Table 5.5 shows how source attributes of each source get divided in buckets for `Stratified Sampling` based on 3 buckets. The created scenario explains clearly the difficulties in selecting source attributes from each bucket in order to get a reasonable amount of source attributes that are “alignable". The situation becomes exasperated when analyzing Table 5.6, which shows the results of partitioning based on 10 buckets. The two previous approaches failed, because of the long and flat tail in the probability

Source	**[0.0, 0.3)**	**[0.3, 0.6)**	**[0.6, 1.0]**
www.ebay.com	1977	6	4
www.alibaba.com	1574	9	7
www.camerafarm.com.au	268	3	12
buy.net	77	4	21
www.pricedekho.com	49	44	11

Table 5.5: Partial example of the distribution of source attributes using Stratified Sampling with 3 buckets

Source	**size [0.0, 0.1)**	**size [0.1, 0.2)**	**size [0.2, 0.3)**	**...**	**size [0.7, 0.8)**	**size [0.8, 0.9)**	**size [0.0, 1]**
ebay.com	1973	1	3	...	1	0	3
alibaba.com	1542	10	22	...	4	1	2
camerafarm.com.au	212	51	5	...	0	10	2
buy.net	45	30	2	...	2	8	7
pricedekho.com	20	13	16	...	0	0	2

Table 5.6: Partial example of the distribution of source attributes using Stratified Sampling with 10 buckets

distributions of attributes in each source.

The third approach was designed to address this issue and provide a way to overcome the difficulties involved in gathering head and tail attributes from separate sources that need to be mapped to the same target attribute in the mediated schema for the creation of the Schema Alignment Ground Truth. This approach was based on:

- Meta-Blocking
- Jaccard Similarity
- Connected Components

Meta-Blocking usually involves the creation of a graph: in our case, the nodes are all the available source attributes, and the edges are weighted based on a similarity measure, the Jaccard similarity. In order to initialize the graph, for each source source attribute (thus, for each node), a set of tokens was created. These tokens are the union of the values that appear under the name of the considered source attribute for each JSON file of extraction in the considered source. Values were first normalized: they

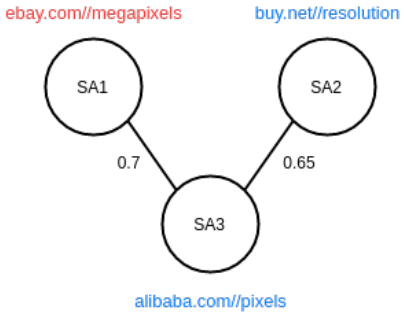

Figure 5.13: Example of nodes and weighted edges in Schema Alignment graph

were transformed to lowercase, trimmed, punctuation, symbols and stop words were removed. Subsequently, the Jaccard similarity between all pairs of source attributes was computed, also between nodes referring to the same source, as we realized by analyzing the dataset that there's heterogeneity even within the same source and that the same semantically equal attributes appear with different names. We used the standard Jaccard similarity metrics.

A weighted edge between two nodes was added only if the computed similarity resulted to be greater than a threshold of value 0.5.

The intuition can be explained in Figure 5.13. The expected result was to have *connected components* of the graph that could be clearly recognized as a cluster of a single target attribute to be considered in the mediated schema. This is a weak approach, such the ones we wanted to try for the already mentioned purposes. The actual result provided a quite good starting point for an intervention by a human. We got clusters with head and tail attributes together, so we overcame the issues of the previous approaches, and the clusters were accurate enough. The need for correction by a human was clear since the beginning, but that was expected, since we were building a Ground Truth. The human involved in the process had to split clusters into smaller ones since similarity between values of semantically different attributes led the algorithm to the creation of coarse-grained clusters, we refer to this kind of clusters as *super-clusters*. For example, one of these super-clusters was the one including source attributes with names such as “weight", “weight including batteries" and “only body weight", whose

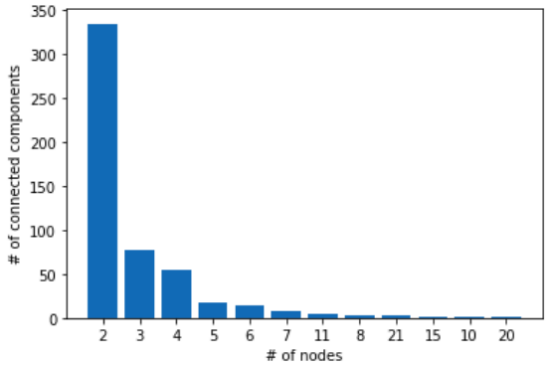

Figure 5.14: Distribution of connected components after meta-blocking for Schema Alignment

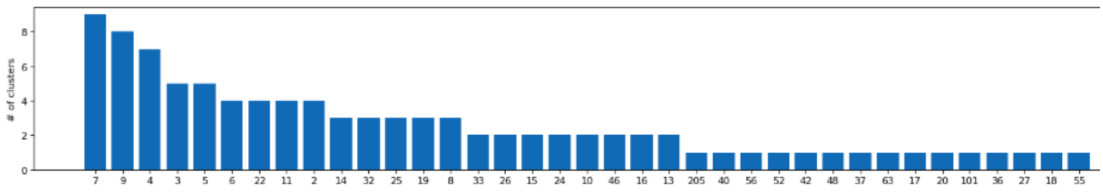

Figure 5.15: Distribution of clusters for Schema Alignment after human refinement

values are similar, but that our approach could not distinguish in terms of semantics. A significant value added by the human intervention was the process of naming the clusters, giving birth to the first target attributes names to be considered for the Schema Alignment Ground Truth.

As a result of the human-based process, we got 94 different clusters, manually refined in about 20 hours of work.

Figure 5.14 shows the distribution of connected components based on their size. As we can see, connected component made of 2 nodes are the most frequent, and the pairs computed are usually correct in terms of semantic meaning of the attributes they refer to. Other connected components are distributed in medium and large sizes; larger ones were the actual starting point for human manipulation. Figure 5.15, instead, shows the result of the human manipulation on the clusters identified by the algorithm: a significant difference in terms of sizes of clusters is noticeable. The human individual

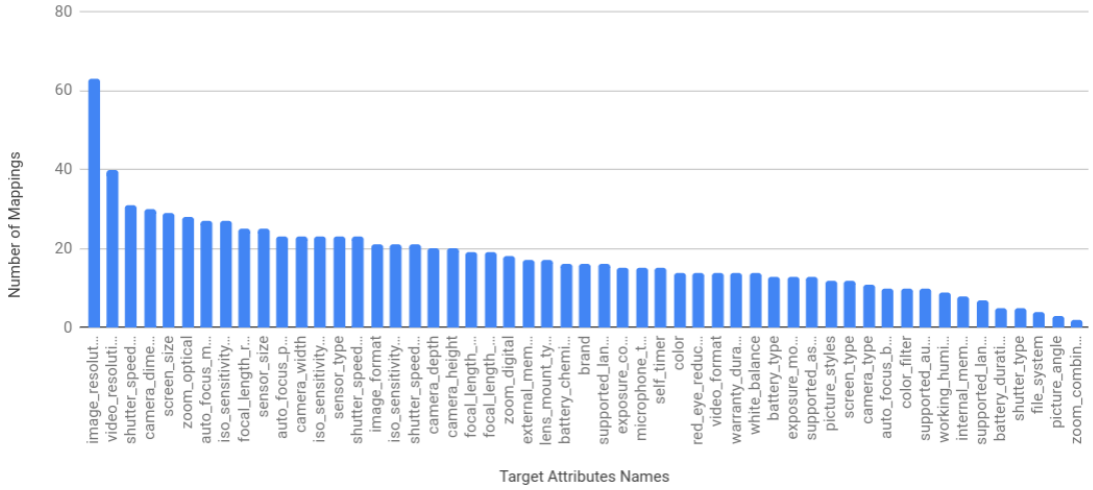

Figure 5.16: Distribution of Target Attributes over the Number of Mappings

involved in the process found it quite easy to select new nodes to add to pre-existing larger clusters or in finding smaller pre-existing clusters to enlarge with the inclusion of nodes coming as a splitting from other clusters.

The 94 clusters, which are sets of source attributes, were refined again a second time, in order to create more fine-grained clusters, with the objective of creating more and less challenging target attributes clusters. Finally, the resulting Schema Alignment Ground Truth, for the current and initial version of the ALASKA Benchmark, is made of 53 target attributes that end-users of the Benchmark need to complete from the ground up with the dataset at their disposal, using the DI2KG Data Format. The number of existing mappings from source attributes to mediated schema attributes is 943.

Figure 5.16 shows the distribution of selected target attributes in terms of number of mappings. As expected, the target attributes with larger number of mappings are the ones that appear the most as first shown specification in product pages: they include the resolution, the shutter speed, the focal length and the dimensions of a camera, which are the first specification, which an individual who wants to buy a camera would be interested in. The list of considered target attributes appears on the horizontal axis of the Figure.

cluster id	product name	# sources	# instances	hours of work	classification
1	NIKON D7000	14	131	7	HEAD
2	NIKON COOLPIX S9500	8	25	4	HEAD
4	PANASONIC FZ200	12	26	3	HEAD
3	CANON EOS REBEL T3i	3	135	4	TAIL
5	FUJIFILM S8100FD	3	7	1	TAIL
6	NIKON D50	2	46	1	TAIL

Table 5.7: Statistics about the Record Linkage Validation Set

5.5 Record Linkage Ground Truth

Record Linkage is one of the main components of Data Integration. The goal is to link the records, each representing a single instance, across different data sources and gathering them in clusters. Each cluster contains instances that refer to the same real-world entity of the considered domain. Record Linkage is one of the tasks supported by the ALASKA Benchmark and, as such, we need a Ground Truth based on our products dataset in order to evaluate the performance of proposed solutions by the end-users. Even for this Ground Truth creation process, the focus was on having as highest Recall as possible by an automated process, whose results would then be refined and corrected via human intervention in order to have 100% Precision. In the Record Linkage Ground Truth creation process this translates to producing “reasonable" clusters of instances related to a single real-world entity, with the highest number of actually matching pairs of instances when considering all the possible pairs of instances in the cluster. Before designing the automated processes, a *Validation Set* was created: this provided us the ability to estimate how good our processes were doing, so that we could concentrate on the development of the best approach. Our Validation Set includes 6 separate real-world entity clusters. The 6 entities are equally divided in head and tail entities. The Validation Set was built completely manually and consists of 370 instances selected in about 20 hours of work.

Table 5.7 shows the main statistics about the clusters belonging to the Validation Set. The creation of the Validation Set was driven by domain knowledge and no particular tool was used during the process. The Validation Set builder only had a simple search tool based on page titles of the instances and had access to related HTML web

pages and JSON files of extraction.

For the current version of the ALASKA Benchmark, the following simplifying reductions, based on the considered domain of cameras products, were adopted:

- instances of products with the same camera body, but with different included accessories, such as lenses, tripods, memory cards, and so on, were included in the same entity cluster;
- instances of products referring to the same entity camera, but with different colors, were included in the same entity cluster.

These rules were followed in the creation of both the Validation Set and the Ground Truth.

5.5.1 Graph-based Approaches

The first approaches that have been experimented for the creation of real-world entity clusters were based on the construction of a graph with weighted edges. The created graph always included the $29,787$ instances of our camera dataset as nodes, while the weights of the edges were based on the output of separate similarity measures used in each approach.

Our first graph-based approach relies on the following intuition: most frequent attributes' values in a given source should be less discriminating and less frequent attributes' values should be more discriminating. Based on this intuition, our first approach, for each JSON file of extraction, works as follows:

- clean attributes' values;
- tokenize all the values;
- compute a weight for each token;
- compute a similarity score between the set of tokens of the current file and the set of tokens of another file, for each file in the dataset.

The weight for tokens, called *inverse_frequency*, was computed, given a source S and a token W, with:

$$weight_S(W) = \frac{\frac{N}{dfW}}{max(weights(S))} \tag{5.2}$$

where N is the number of JSON files of extraction of source S, dfW is the number of JSON files of extraction of source S that contain the token W, and $max(weights(S))$ is the maximum weight in source S.

The similarity score, which represents the probability of two instances being in match, was computed, given a set of tokens $E1$ from source A and a set of token $E2$ from source B, with the following:

$$similarity\_score = \frac{\sum min(weight_A(x), weight_B(x)), \forall x \in E1 \cap E2}{|E1 \cap E2|} \tag{5.3}$$

A graph with a node for each instance in the dataset and with edges weighted based on Equation 5.3 was built. An edge was added only if the associated weight was above a given threshold. For creating clusters for real-world entities, the connected components of the graph were considered. Each connected component represents an entity, with its nodes representing the instances referring to the entity.

This first approach failed, as the results provided a giant connected component with 7561 nodes and about 20000 isolated nodes. Different threshold for weighted edges were experimented, but the effect of doing this was only the enlargement or downsizing of the giant connected component, as well as the downsizing or enlargement of the number of the isolated components.

The second approach is schema-based, as we can leverage the target attributes clusters produced semi-manually in the previous phase of Schema Alignment described in Section 5.4.1. This approach is based on the construction of a graph and follows the same considerations made for the first approach in terms of similarity measure, but only considering tokens from attributes that are mapped to shared target attributes. More precisely, the graph construction followed these steps, for each JSON file of extraction (i.e. for each instance):

Figure 5.17: Example of HTML page title of an e-commerce web page regarding a camera

- map its attributes to target attributes, based on schema mappings
- create a set of tokens from values of the attributes that had at least one mapping
- compute the similarity measure (Eq. 5.3) considering the set of tokens from other JSON files of extraction, built the same way, for each other file

The graph with as nodes all the instances of the camera dataset and as weights of the edges the similarity measure computed as described in the previous list was built and its connected components were analyzed. This approach failed in a different manner: there was no giant connected component, but almost only isolated nodes. This suggests that a refinement of the Schema Alignment is necessary or that the approach needs to be smarter and more robust.

A third kind of approach was designed. It is based on the intuition that in the title of the products HTML pages of e-commerce websites there is often everything we need for identifying the real-world entity. Figure 5.17 shows the HTML page title of a famous e-commerce, which, among other kind of products, sells also cameras. As we can observe, in the page title we can find the brand, the model, the resolution, the type and even the color of the camera the web page refers to. We can also identify other pieces of information about accessories, which for us are considered as noise, as they help only if comparing two cameras sold with the same accessories bundle. Attributes and their values were not considered for this approach, only page titles were taken into consideration.

Titles cleaning and normalization was performed by:

- deleting recurrent common patterns (every source was analyzed manually in order to find these patterns, e.g. “Buy...", “...at eBay.com", “...from CamBuy in

Sidney", and so on)

- removing unneeded spaces
- transforming to lowercase
- removing common selected words (e.g. “digital", “camera", “with", “compact", and so on)
- removing isolated characters and symbols

This was done with the objective of creating a set of bigrams from every page title of instances at our disposal. The created bigrams are overlapping by one element, i.e. the window moves forward one word per time (e.g. from the title “polaroid is426 16 megapixel" the following bigrams were created: (“polaroid",“is246"), (“is246",“16"), (“16",“megapixel")).

Table 5.8 shows the top 20 most common bigrams created from all the available page titles. We built a graph in which nodes are representative of instances and an edge (not weighted this time) exists only if the titles of the instances it connects share a selected number of bigrams. Several thresholds of this kind were experimented, but results were not enough good in terms of clusters. Again, connected components of this graphs were considered as the entity clusters. Medium-sized connected components were actually quite correct, but the goal of 100% Recall was way far, based on how the graph re-built the clusters of the Validation Set and even by manually checking the resulting clusters.

The results obtained with the approaches described in the current Section were not enough for what our objective was, hence we drastically changed methodology switching from our target of creating larger, high recall, but less accurate cluster in favor of a construction of the Record Linkage clusters from the ground up, with an approach that relies on responses from humans about pair-wise matching candidate pairs.

Bigram	Occurrences
('canon', 'eos')	2329
('canon', 'powershot')	2039
('mp', 'slr')	1785
('nikon', 'coolpix')	1501
('mp', 'black')	1329
('slr', 'black')	1328
('body', 'only')	1312
('16', 'mp')	1109
('12', 'mp')	1069
('cyber', 'shot')	918
('eos', 'rebel')	915
('sony', 'cyber')	888
('panasonic', 'lumix')	827
('shot', 'dsc')	792
('black', 'body')	782
('18', '55mm')	747
('10', 'mp')	734
('fujifilm', 'finepix')	686
('optical', 'zoom')	685
('mp', 'silver')	612

Table 5.8: Most common bigrams in page titles

5.6 Crowdsourcing Web Application

Crowdsourcing based solutions exploit the assistance received from a crowd of humans to effectively solve a wide variety of problems. Typically this kind of problems are quite easy for humans to solve, but way more difficult for automated computer algorithms. Applications of such systems are easy to be found on the web in the last 15 years [51]. Crowd-sourcing systems come with a brand new vision of problems, along with a problem formulation that is very different with respect to traditional top-down or bottom-up approaches. Humans are in the loop and the problem needs to be solved reasoning about what contributions humans can give and how to combine them. Even the evaluation of users and strategies to retain the more capable ones play a major role in this kind of systems.

It is with the help of the crowd that we addressed the issues that Record Linkage raised, which we could not overcome with automated approaches. This new solution follows a different direction with respect to the approaches described in Section 5.5. This new approach aims at building clusters of real-world entities from the ground up, leveraging pair-wise matching that are verified by the humans in the loop. This solution was inspired by the research work “Online Entity Resolution Using an Oracle" [52] and represents a simplified implementation of the proposed algorithm. A web application based on crowd-sourcing was developed and humans were involved in the pair-wise matching problem: for human, understanding if two products are actually the same product is a quite easy problem to solve.

The workflow, on the user side, is as follows: the User logs in, then starts a Record Linkage Session, which consists of a set of tasks that need to be accomplished; the tasks are the declaration of match or non-match about a pair of products. The User, in order to distinguish between products, is shown the specifications that have been extracted from the associated e-commerce web pages and their page title. The User also has the possibility of reaching the web pages of products of the current task, if in need of a more complete set of information about the products. The User then declares if the two products are in match or not and subsequently chooses if he wants to continue with another task or to end the current Linkage Session. It is important to say that the User does not need to finish all the assigned tasks before logging out.

The end goal of the Application is to create clusters of instances referring to the same real-world entity: these cluster should have no instances in common, if two clusters share an instance, it means that they need to be merged into one bigger cluster, as they refer to the same real-world entity, or it is an error in the Record Linkage process with the consequence that we no longer have 100% Precision. Entity clusters are, in fact, *disjoint sets*. Typically, when dealing with disjoint sets, two particular operations are needed: finding the unique set that contains a given element and computing the union of two sets. This data structure was named after these two operation: `UnionFind`. It manages disjoint sets of nodes, represented as rooted trees, with each node containing

one member and each tree representing one set. It makes use of two heuristics, “union by rank" and “path compression", for achieving better performances.

A Record Linkage process on the entirety of the dataset of instances would produce a partition of the instances, divided in clusters that have nothing in common. For the current version of the ALASKA Benchmark, the target was to create clusters for 100 selected real-world entities, including both head and tail entities. With head entities we refer to real-world entities whose cluster has instances from at least 10 separate data sources, while with tail entities we refer to real-world entities whose cluster has at most 10 instances instances. In order to have a preliminary idea about an entity, we developed a script, which tries to capture the camera model by searching in the associated page title through the use of regular expressions. After finding all the camera models related to all the instances, clusters of instances with the same model were created. Before creating clusters, the camera models found were cleaned and reduced to a common form. The size of the created clusters gave us an insight about the probability of the entities being a head or a tail entity. The sizes were considered as lower bounds to the actual sizes of clusters, and this gave us the opportunity to select the entities we wanted to build the cluster of. For each selected entity, a main seed was chosen, the initial instance of the cluster.

One of the core challenges of the Application is the task creation. The work at the base of the Application proposes as key points the definition of a function of pair-wise matching likelihood and a benefit function. This is due to the fact that velocity of clusters construction is a main objective. Tasks are constituted by pairs of instances that the user needs to evaluate in terms of being matching or non-matching. Hence, the Application needs a smart methodology for selecting the most promising instances to be proposed to the user in order to enlarge the cluster in construction.

Instances that need to be linked as part of a real-world entity cluster are seen as nodes of a graph, which has weighted edges between nodes; the weight of the edges represents the probability of the nodes they link together to be in match: in particular, an edge between two nodes exist if their similarity score is above a given threshold.

Stopping condition: 2 consecutive question blocks with only negative responses

VAL SET CLUSTER ID	ENTITY	CLASSIFICATION	# SOURCES VAL SET	# INSTANCES VAL SET	# BLOCKS	# QUESTIONS	RECALL
1	NIKON D7000	HEAD	14	131	19	95	0,42
2	NIKON COOLPIX S9500	HEAD	8	25	9	45	1
3	CANON EOS REBEL T3	MIDDLE	3	135	10	50	0,21
4	PANASONIC FZ200	HEAD	12	26	5	25	0,42
5	FUJIFILM S8100FD	TAIL	3	7	4	20	1
6	NIKON D50	MIDDLE	2	46	15	75	1

Stopping condition: 3 consecutive question blocks with only negative responses

VAL SET CLUSTER ID	ENTITY	CLASSIFICATION	# SOURCES VAL SET	# INSTANCES VAL SET	# BLOCKS	# QUESTIONS	RECALL
1	NIKON D7000	HEAD	14	131	41	205	1
2	NIKON COOLPIX S9500	HEAD/MIDDLE	8	25	10	50	1
3	CANON EOS REBEL T3	MIDDLE	3	135	49	245	0,91
4	PANASONIC FZ200	HEAD/MIDDLE	12	26	6	30	0,42
5	FUJIFILM S8100FD	TAIL	3	7	5	25	1
6	NIKON D50	MIDDLE	2	46	16	80	1

Stopping condition: 3 consecutive question blocks with only negative responses AND benefit less than threshold = 0.3

VAL SET CLUSTER ID	ENTITY	CLASSIFICATION	# SOURCES VAL SET	# INSTANCES VAL SET	# BLOCKS	# QUESTIONS	RECALL
1	NIKON D7000	HEAD	14	131	84	420	1
2	NIKON COOLPIX S9500	HEAD/MIDDLE	8	25	12	60	1
3	CANON EOS REBEL T3	MIDDLE	3	135	102	510	1
4	PANASONIC FZ200	HEAD/MIDDLE	12	26	37	185	1
5	FUJIFILM S8100FD	TAIL	3	7	6	30	1
6	NIKON D50	MIDDLE	2	46	20	100	1

Figure 5.18: Simulations for similarity function selection

This graph was pre-computed and injected into the Application. The weight of the edges are computed in a way similar to the approaches described in Section 5.5. By leveraging the Validation Set, we developed a similarity function, which satisfies the following requirements:

- the sub-graphs (of the graph built using the similarity scores as weights) induced by the clusters of the Validation Set are connected;
- with respect to the different tested similarity functions, by conducting a simulation in which the User hypothetically declares as matching all the proposed pairs, the clusters of the Validation Set are completely re-built from the ground up (100% Recall) before reaching the considered stopping condition (discussed later).

Figure 5.18 shows the results of the simulations run for three different stopping conditions, using the same similarity measure. The last table in the figure shows the results of the simulation based on the similarity measure that has been used in the Application, it is based on the average Jaccard Similarity between the sets of bigrams

and between the sets of tokens shared by the page titles of the instances in comparison. If the instances do not share any bigram, then only the Jaccard Similarity of the shared tokens is considered, viceversa if the instances do not share at least 3 tokens, then only the Jaccard Similarity of the shared bigrams is considered. The final similarity function is based on the similarity of bigrams and tokens of the page titles of the instances. The function `get_next_question` exploits the pre-computed graph and selects the best instances to propose to the User. The selection of an instance is based on a measure of *benefit* that the inclusion of the given instance would bring to the cluster.

The benefit is simply computed as the weighted degree of the node representing the candidate instance, when considering a sub-graph, comprehending only the given node and the nodes of the cluster currently in construction, of the total graph. When the benefit of all the candidate instances has been computed, the most promising ones are selected: actually, the Application proposes to the User a number `QUESTION_BLOCK_SIZE` (lines 28-29) of instances to be declared or not in match with the seed of the current cluster. Receiving the responses of the User is also critical, but most of the complexity is handled by the `UnionFind` data structure, described earlier. The responses, both the positive and negative ones, are stored in the database. The handling of the responses involves including the instances, which received a positive answer for being in match, in the cluster which is currently being built. New tasks regarding the current cluster are stopped being created when arriving in two different scenarios, not necessarily at the same time:

- the last question has less than `QUESTION_BLOCK_SIZE` candidate instances
- the most promising instance to be proposed has a benefit score, which is less than a given threshold, called `BENEFIT_THRESHOLD`, and the Application has received a series of consecutive negative responses longer than a threshold, called `ALL_NEGATIVE_BLOCKS_THRESHOLD`

When one of these two scenarios appear, the Application switches to the construction of another cluster and, at the same time, resets the counter of negative responses received

and resets the “colors" of the instances seen by the User during the past construction, but still not linked to any cluster, as shown in lines 41-45.

As a result from the use of the Application by the oracles, we obtained 100 real-world entity clusters. A total amount of 2846 instances were included in the correct clusters.

5.7 Iterative Record Linkage Pipeline

Record Linkage is a difficult process and complexity increases when dealing with very heterogeneous data, such as our camera dataset. We talk about Big Data Integration when referring to the ALASKA Benchmark, but, considering all the famous “V’s" of Big Data, the “V" of *Variety* is the most characterizing one in our scenario. As we have seen, entities come with different kinds of representation, attribute hide difficulties related to their semantic ambiguity in term of their name, that can mislead, and their values, very problematic to manage. The application of different approaches, described in the previous sections, led to good results, that we think have a high room for improvement. Problems, such as the giant components from the graph-based approaches, or the high ratio of wasted questions asked to the oracles through the web application (up to 80% of negative, thus wasted, responses when building certain clusters) brought us to design a system that can have multiple point of views and that can hopefully capture the salient aspects of instances when deciding if a pair is a match or not.

We designed an iterative pipeline, whose workflow is shown in Figure 5.19. This pipeline is mainly based on the use of independent components that we call “Black Boxes"; they represent the different point of views mentioned earlier. Each black box is required to provide as output the most promising pairs of instances, while taking as input a selection of seed instances, which are the representative instances of a real-world entity cluster. For the initial version of this pipeline, we thought of 3 black boxes, the first based on Deep Learning (DL) techniques, the second based on Machine Learning (ML) techniques and the last one based on techniques, that do not exploit DL or ML techniques, such as statistics or in general classic techniques (e.g. JedAI). This pipeline

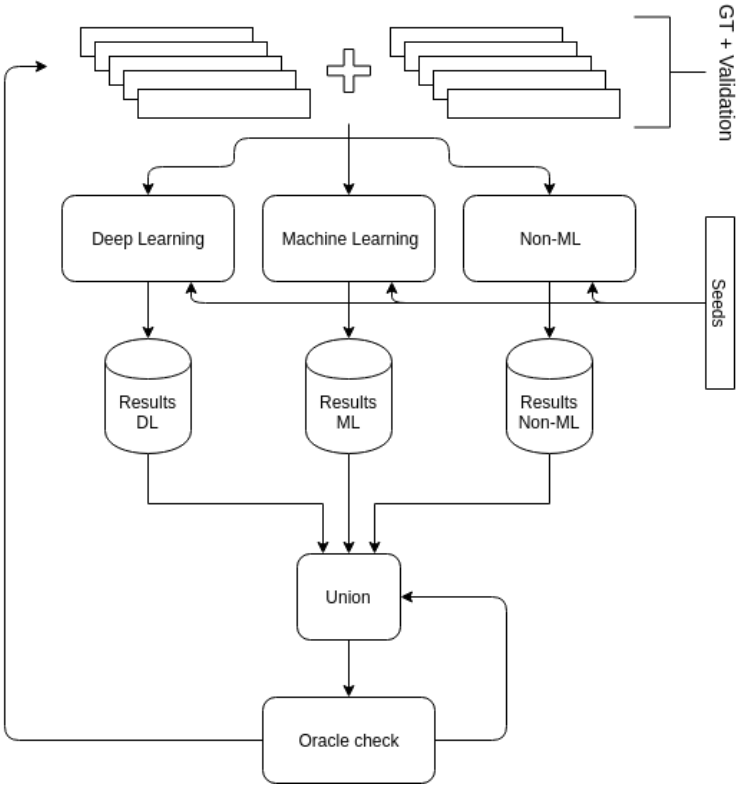

Figure 5.19: Iterative pipeline for Record Linkage

is still based on the use of oracles, as highest precision and recall as possible is still the main target. The iterative workflow is as follows:

1. black boxes are trained through the use of matching and non-matching pairs identified via the previous approaches. In particular, we select each pair of instances in the same cluster (in our Ground Truth for Record Linkage and the Validation Set created) as a matching pair, and each pair pertaining to different cluster as non-matching pair (this does not include pairs in which one or two instances do not belong to any of ground truth clusters)

2. black boxes receive as input the same seeds, representatives of the clusters that need to be completed

3. black boxes produce as output a ranked list of most promising pairs for each seed,

in which one element is the seed and the other element is an instance that is not already in a cluster

4. a component, called *Union*, merges the lists of the same seed in a smart way by combining the results (and ranks) from all the black boxes

5. oracles check the lists of most promising pairs and verify the matching ones

6. non-matching pair are provided to the *Union* component for smarter merging in the next iteration (the component will be able to not propose again a pair which has already a response by the oracles)

7. new instances of matching pairs are included in the respective clusters

When the *Oracle Check* component slows down the production of new instances to be included in clusters for each iteration, new seeds for the black boxes can be provided, or new black boxes can be included. The intuition is that, for each new iteration, the availability of new training data for the black boxes will increase the observed variety and help the black boxes themselves in the generalization of their process. This intuition was strengthened by the preliminary results obtain by a work-in-progress black box. This black box is based on the use of the Magellan framework for Record Linkage (see Section 3.3) and includes a Random Forest Classifier that declares pairs of instances, more specifically, a features vector that represent that pair, as matching or non-matching, with also a probability of them being in match. This black box was trained with matching and non-matching pairs of example of the clusters in our Record Linkage Ground Truth and Validation Set. The training was based only on 30% of the instances of the clusters, while the testing was based on pairs of instances from the rest 70% of the instances.

Figure 5.20 shows the precision, recall and f1-measure obtained on the classification of pair of instances from the clusters of our Ground Truth and Validation Set that the model never saw for its training, which was based on pair of instances belonging to the same clusters. A f1-measure of 97.07% is great and gives hopes for the results of this

Result number	% nodes for each cluster	precision	recall	f1-measure
1)	30,00%	99,52	94,42	96,90
2)	30,00%	99,21	95,23	97,18
3)	30,00%	99,43	94,91	97,12
AVG		99,39	94,85	97,07

Figure 5.20: Magellan results for Record Linkage

approach. The tests on the classifier were made in a way, that simulates the way the iterative pipeline works: the black box will be firstly trained on the current clusters, which represent a subset of the real clusters that we aim at, then it will be asked to produce matching pairs with instances it never analyzed during training.

These preliminary results are a good starting point for the actual outcome of the approach based on this iterative pipeline.

5.8 Instance-level attribute alignment Ground Truth

The alignment at instance-level, also called *tagging* task in the ALASKA Benchmark, is the most unusual one in the Big Data Integration panorama.

We expect that the participation to the challenge proposed by this task will be way less with respect to the Schema Alignment task and the Record Linkage task, which are recognized and deeply studied problems by the research communities.

In our case we are dealing with messy attributes with a lot of variety: different names for the same semantic concept within the same source and between separate sources, mixed kind of values for the same considered attribute, and even errors due to the sources themselves or the extractor used. Chapter 4 describes these *local heterogeneity* cases with more details.

The end goal is to declare, after a previous process of Record Linkage, which data source contributed with which $< attribute\_name, attribute\_value >$ pair to the values of a given target attribute when building the integrated data available for an identified real-world entity. This is a rough and extremely difficult task, as the semantics of values

are involved and the scenario in which it is immersed is a very ambiguous one, with misleading names and non-correct values to take care of.

The Ground Truth for the Tagging task of the ALASKA Benchmark was created entirely by hand.

For the initial version of the Benchmark, a simplifying restriction was applied in order to create the Ground Truth for the Tagging task, which is as follows: a key-value pair (called *provenance* in the Benchmark work), belonging to a certain JSON file of extraction, can contribute with its value only to the values of one of the target attributes that the source attribute (identified by the key of the pair and the source of the JSON file of extraction) is mapped to in our Schema Alignment Ground Truth. In particular cases, a key-value pair can also not contribute to any of the target attributes.

The process of creating this Ground Truth was done through the use of a software tool that, for each real-world entity clusters, for each instance that is included in the cluster, fetches the possible target attributes that the considered instance can contribute to and then proposes on the screen the possible choices that can be done. The user of the tool only needs to select a single target attribute that the current key-value pair refers to; by not selecting any target attribute the user declares that the considered key-value pair does not contribute to the integrated data of the entity: it is a possible scenario due to the high heterogeneity of the dataset.

Chapter 6

Experiments

In this Chapter we present the experiments that we made to demonstrate the effectiveness and robustness of RaF-AIA.

The experiments were performed on the Camera and Monitor datasets provided by Alaska Benchmark, described in Chapter 5. The Benchmark record-linkage ground truth is used as input of our algorithm.

In order to evaluate our approach, we consider precision and recall of our clusters with respect to the ground-truth clusters. As clusters can overlap, we evaluate results over *pairs of attributes* in match, i.e., pairs of attributes that share *at least* one cluster [53]. Precision is computed as usual as the fraction of correct pairs over all the pairs in match the algorithm provided, while recall is the fraction of actual pairs in match that the algorithm found. Because of the incompleteness of the ground-truth, evaluation is limited on all the pairs that occur in the ground-truth clusters: if the algorithm provides two attributes in match, and one or both are not in the ground truth, this pair is ignored and not considered as a true or false positive. However, if both the attributes are in the ground truth, but they do not belong to the same cluster (they do not form a pair), it is considered a false positive.

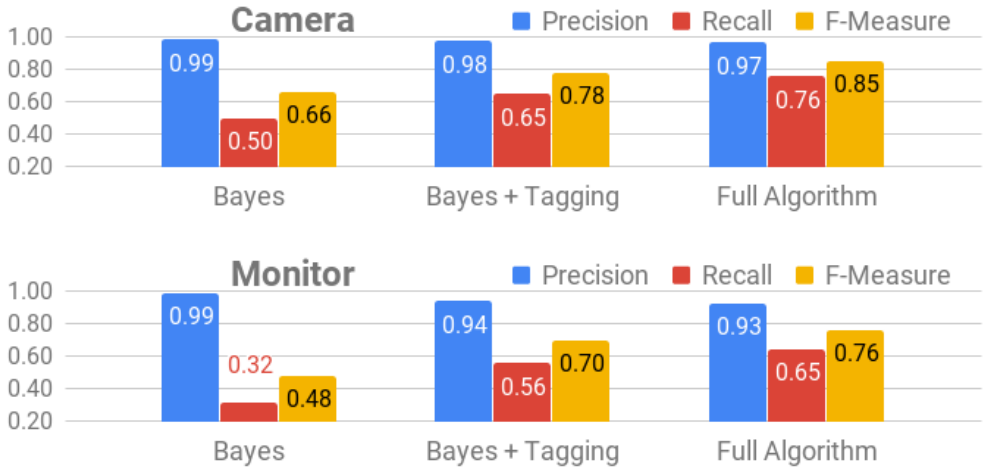

Figure 6.1: Experiments on algorithm phases

6.0.1 Evaluation of the Steps

First, we report results on the contribution of each step of the RaF-AIA approach. Figure 6.1 summarizes the main results.

The Bayesian matching step clusters together attributes that share the same values for a sample of linked specifications. The approach is strongly conservative and produces very homogeneous attribute clusters: launched alone (on the original dataset) it achieves high precision but low recall.

In combination with the tagging step, the overall approach improves the recall, thanks to the creation of virtual attributes that increment the match opportunities. It increases the recall, even if there is a small loss in precision, which is due to the possibility that some wrong virtual attributes are created, producing accidental matches.

With the final step, which merges clusters based on attribute names and domain similarity, the overall approach further improves the recall at the cost of a small loss in precision.

It is important to observe that results are similar on the two datasets (camera and monitor).

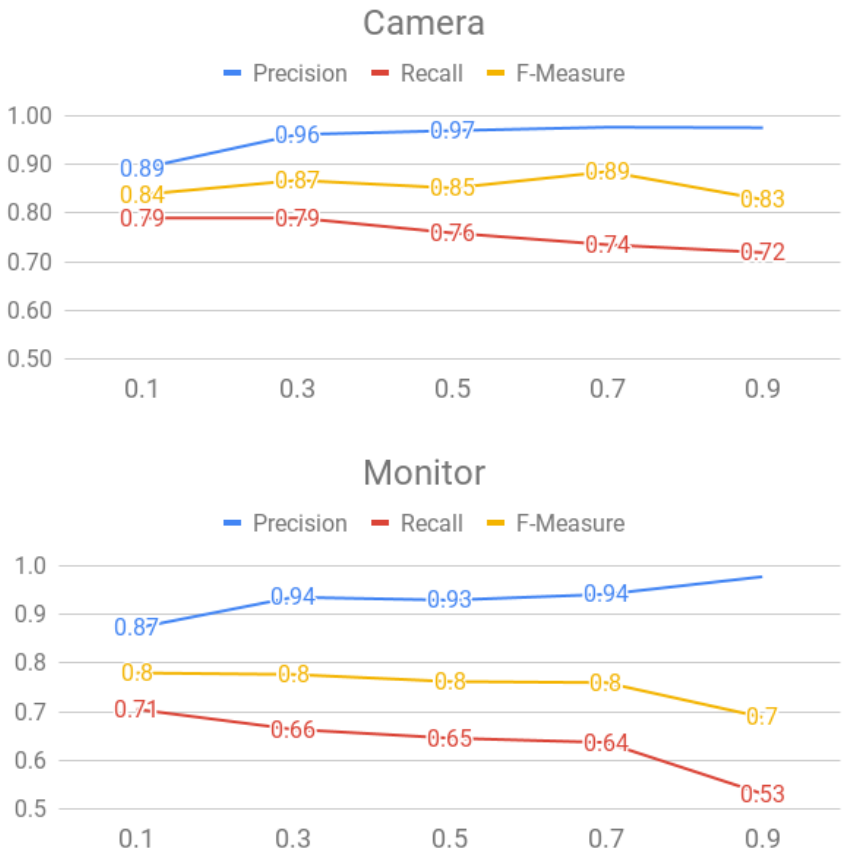

Figure 6.2: Varying the match threshold

6.0.2 Robustness to the Match Threshold

The matching algorithm depends on a threshold on the similarity score between a pair of attributes. As the similarity score refers to a probability, we consider that two attributes match is their similarity score is greater than 0.5. However, it is important to investigate the robustness of the approach with respect to this threshold. To this end, we have run experiments varying the threshold between 0.1 (matches with low probability are allowed) and 0.9 (strict matches). Figure 6.2 shows the results of RaF-AIA with different values of the matching thresholds, i.e., the minimum score above which source attribute pairs are considered in match by the Bayesian analysis.

As expected, precision increases and recall decreases as long as threshold is higher,

except for some small oscillations due to the iterative nature of the algorithm. The differences are however not so significant: in most cases the Bayesian analysis provides strong evidences of match or mismatch, making the algorithm robust to this threshold. Also, the precision never drops even with a very low threshold: indeed, some low-weighted edges, even if above threshold, may be ignored if they would break the assumption that no attributes instances in the same specification may be in the same cluster.

6.0.3 The Role of Linkage

The Bayesian analysis exploits also the linkage sample to match attributes, by comparing the values of attributes in linked instances. To evaluate the robustness of the algorithm with respect to the size of the linkage sample, we artificially removed a random part of the linkage and evaluated the performances of the approach.

Figure 6.3 reports the results of the experiment. As the percentage of linkage increases, the Bayesian matching step can rely on more evidences, thus recall tends to improve significantly, with the precision remaining generally stable. However, RaF-AIA provides good results even with few linkages available.

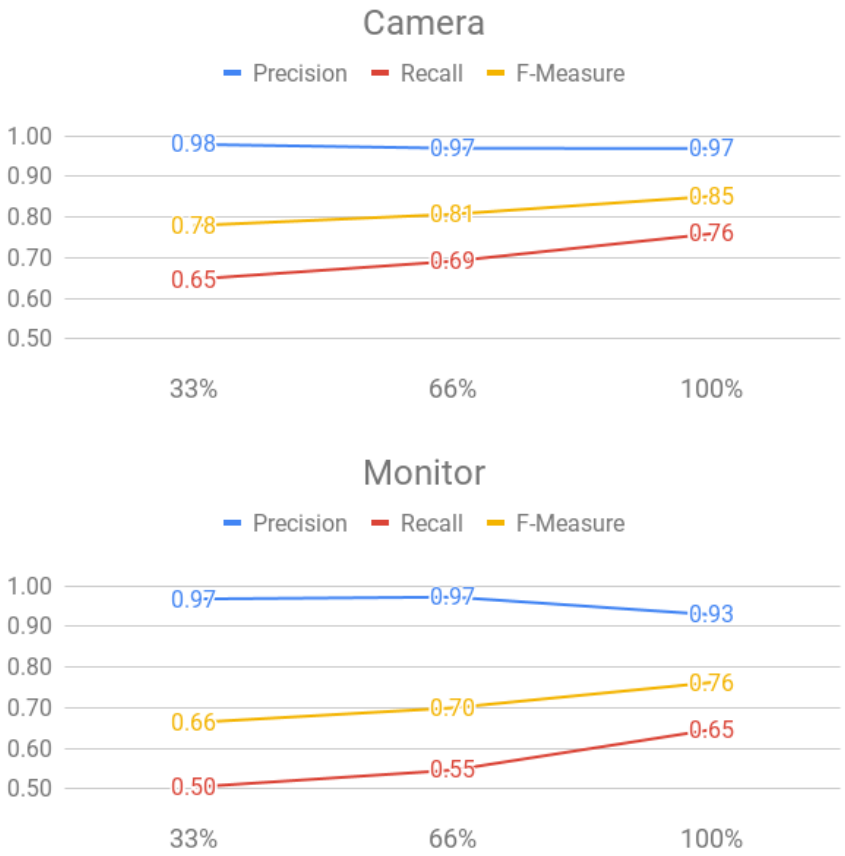

Figure 6.3: Varying the percentage of available linkage

6.0.4 Number of Sources

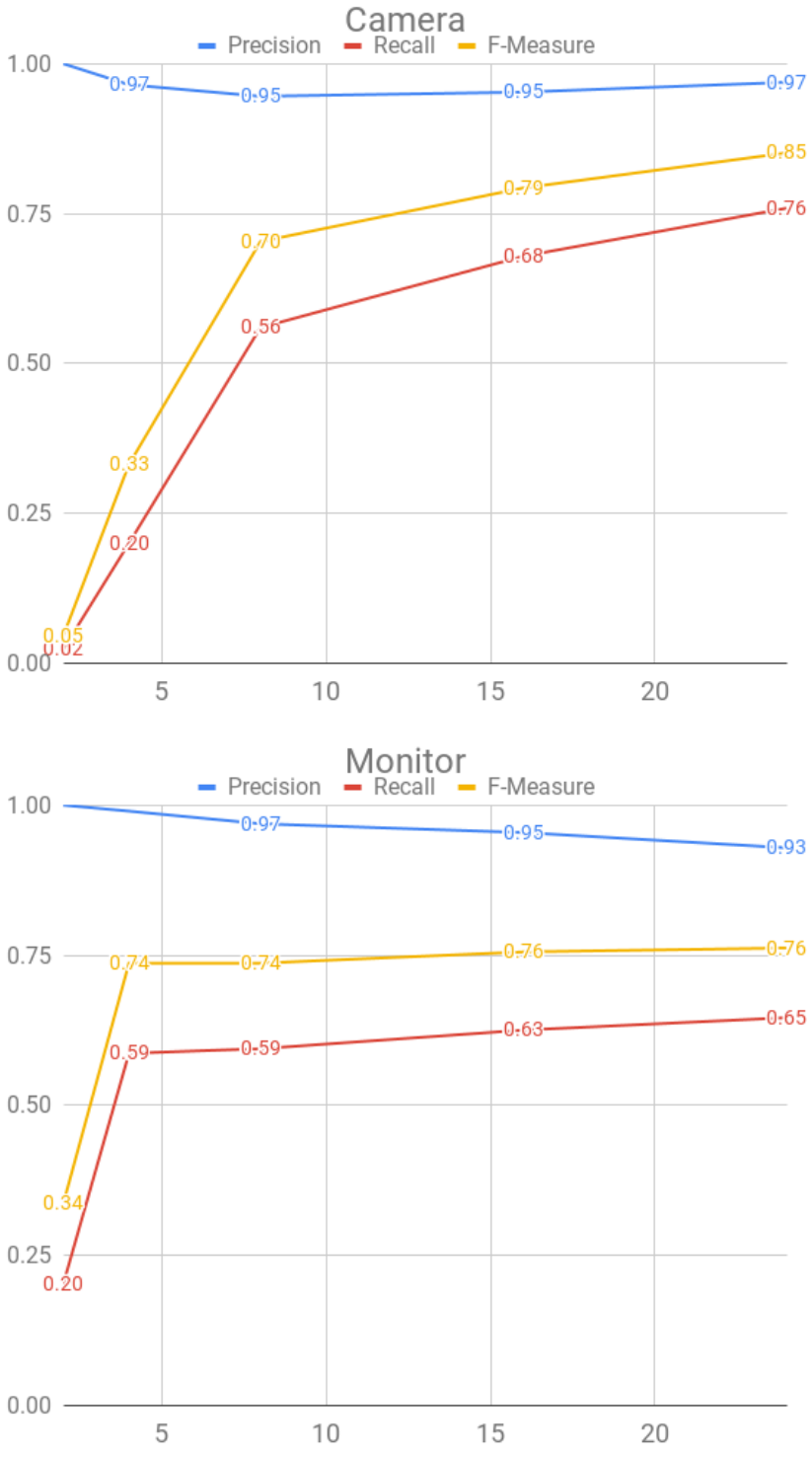

Figure 6.4: Varying the number of sources

Our approach aims at taking advantage of the redundancy of information that occurs among the sources. On the other hand, more sources might imply higher heterogeneity and noise. In order to evaluate the impact of the number of sources on the performance, we have applied the RaF-AIA approach with an increasing number of random sources. Fixed the number of sources, we have repeated the experiment 5 times (each one with a different set of randomly picked sources), and we have computed average precision and recall, removing best and worse cases, to avoid biases due to the random choice. Figure 6.4 presents the results of these experiments. We observe that more sources bring more evidences and data redundancy, thus a better recall, while the impact on precision, due to noise and heterogeneity, is low.

6.0.5 Error rate

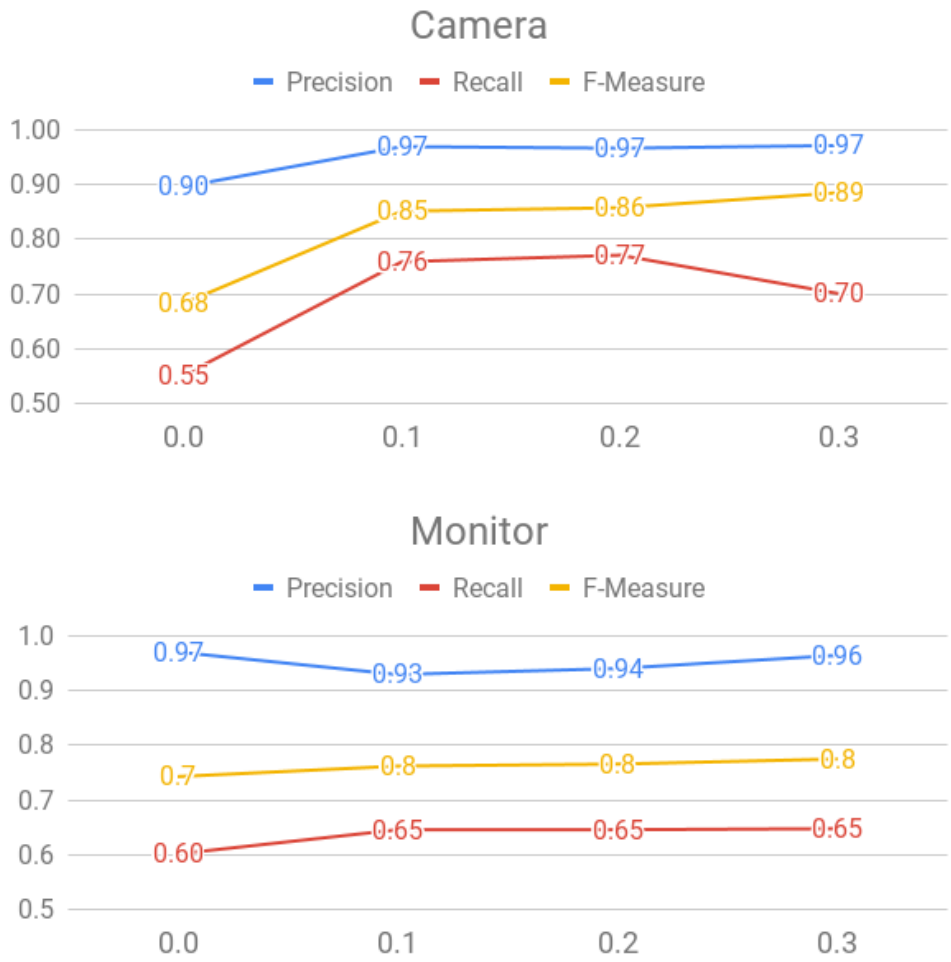

Figure 6.5: Varying error rate

The matching model takes into account that sources may provide a wrong value for a given property (or they maybe an error in the extraction steps). We assume that each attribute has the same error probability ϵ for every observation, currently set at 0.1. It is important to investigate the robustness of the approach with respect to this parameter.

Figure 6.5 shows the results of RaF-AIA varying ϵ between 0 and 0.3. High error rate tend to penalize matching between attributes with many linkage and no mismatch

(typically attributes with low cardinality), while on the opposite it favours attributes with few linkages and high cardinality (more error-prone).

In the plot we can see that, when error rate is set to zero, we have a general loss in F-Measure both for Camera and Monitor dataset. In both cases we have a smaller recall, indicating that equivalent attributes were not detected by the algorithm, probably because of some mismatch in values. Notice that if ϵ is set to zero, we consider that no errors are possible, therefore, any pair of attributes having even for just a single products two different values, are considered as non matching. Surprisingly, precision in Camera also is lower with zero-error: indeed, non-equivalent attributes with a lot of matches by chance (because of low cardinality) are favoured by this threshold and thus matched by the algorithm.

The results for higher error rate have some oscillations, but generally remain quite stable, proving that in most cases the choice of a particular ϵ does not affect results of the algorithm.

6.0.6 Comparison with Alternative Approaches

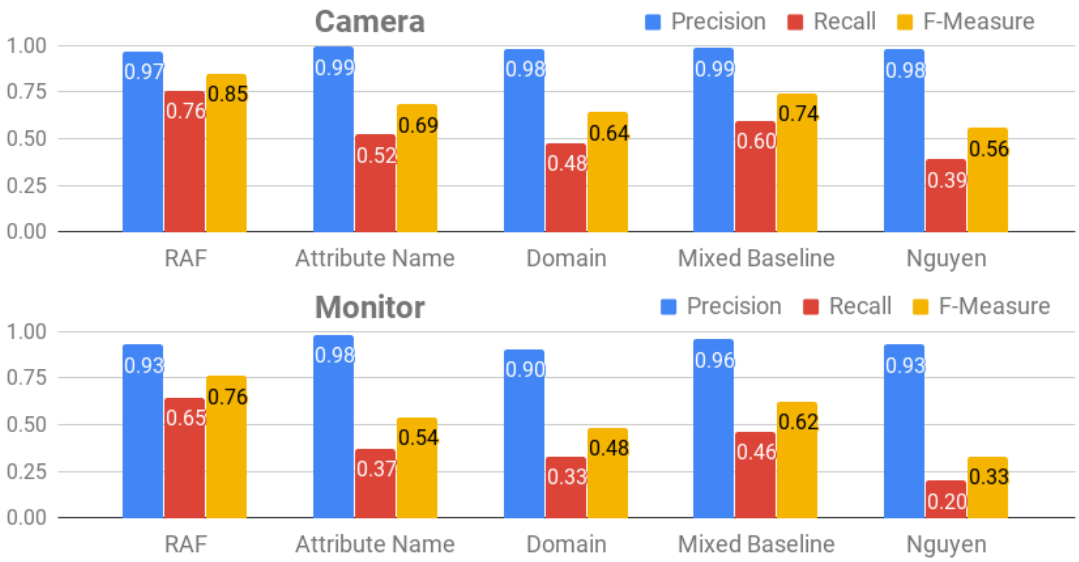

Figure 6.6: Comparison with baselines and alternative approaches.

We compared the RaF-AIA approach to some baselines and to an alternative schema matching approach for product specifications known in the literature. As baselines we consider: (*i*) a naive approach that simply groups attributes with the same name (we call this baseline Attribute Name); (*ii*) another simple approach that creates clusters based on the similarity of the domain of the source attributes, using Jaccard containment index as similarity measure (Domain); (*iii*) the last baseline combines the results of the previous ones by merging their clusters that overlap with at least one attribute (Mixed Baseline). The existing schema matching approach that we use for comparison is that developed by Nguyen *et al.* [9]. This approach (which is discussed in more details in Section 3.1.3) aims at aligning specifications from multiple sources to a reference catalog of products provided input. They match attributes from the external sources to the catalog, by means of a classifier trained to predict attributes matches. The classifier is trained by attributes with the same name, based on the assumption that they have the same semantics. To adapt their approach to our setting, we elected as *catalog* the source with most specifications in linkage, and then aligned the other sources according the approach developed by the authors.Also, we do not delete attributes that do not match with any attribute of the catalog (as in the original approach), but we add them to the catalog, so they are available for matching with further attributes.

Figure 6.6 shows the results of this comparison.[1] We observe that all the approaches achieve high precision, with the baselines performing slightly better, but RaF-AIA significantly outperforms all the competitors in recall. The approach developed by Nguyen *et al.* has the worst performance. In a heterogeneous setting with many homonym attributes, training the classifier relying on the assumption that attributes with the same name are semantically equivalent drastically compromises the accuracy of the predictions.

The results in Figure 6.6 refers to the performance of the different approaches on the overall datasets. However, it is interesting to investigate how they perform on different partitions of the datasets.

[1] In the plots, we refer to our approach simply as RAF, for brevity.

Figure 6.7 reports the F-Measure of RaF-AIA, along with Name and Domain baselines, on different partitions of the dataset.

Figure 6.7: Performance on different partitions of data

As our approach aims at exploiting the redundancy of information, it is interesting to investigate how it performs with respect to the distribution of target attributes in the sources, which is related to the degree of redundancy of a product across sources. In our example, the battery chemistry is very popular product property, as it is present (with different source attributes) in every source. However, in our setting, it is important to consider also the size of source attributes within a sources, as a source attribute with a small number of occurrences has lower chances to be aligned. To consider these aspects, we have created four distinct partitions according to the distribution of attributes across sources and within sources. In Figure 6.7, these partitions are denoted HH, TH, HT, TT: the first letter identifies a Head/Tail (i.e., popular/rare across sources) target attribute, the second letter indicates a Head/Tail (i.e., large/small) source attribute.

We can see that RaF-AIA obtains better results than the baselines on all the partitions, proving robustness and generality. However, we can highlight some significant results observing that RaF-AIA improves significantly baseline results on the HT partition. In fact, thanks to the Bayesian algorithm and to the exploitation of linkage, RaF-AIA is able to discover matches even with few evidences. On the other hand, working with head target attributes, it is more likely that, even if two source attributes

have small or no evidences for match, they will be matched *by transitivity*: they may be both matched with a common attribute, or there can be even longer paths. In this sense RaF-AIA can be considered a holistic approach, taking into account all other attributes while needing to decide if two specific attributes are in match.

RaF-AIA has good results even for homonyms, always better than alternatives. Notice that the name baseline has clearly a very bad results on this subset.

Figure 6.7 presents results also for an interesting partition of attributes, which we indicate as Homonym/Non-Homonyms. The Homonyms class represents the challenging set of attributes that assume the same name but provide values with different semantics within the same source (as attribute `Battery` in source S_3, in our running example). Also in this case we observe that RaF-AIA always outperforms the baselines, especially in the monitor dataset.

Chapter 7

Conclusions

Data integration is a continuously evolving field, due to its complexity sometimes it requires specific domains solution.

The integration of product specifications, in particular, is a major research issues, with many industries involved. One of its main goals is to allow the construction of a comprehensive product graph, an enabling technology for many applications [7].

We presented a complete pipeline for effective integration of products from Web sources. The development of each step of this pipeline is an iterative process: even if many components have been developed in previous works, new advancements exposes new challenges, new opportunities and refinements in problem definition that may require adaptations on former steps.

We addressed the issue of attribute alignment in this context. We realized that current state of the art techniques were unfit to solve our problem, because they all rely on *local homogeneity* of sources, i.e. the fact that records of the same source present data in the same format, with coherent attribute names and the same granularity, which is not true in our context.

In fact, we did not have just to solve the same problem with a different approach, but we had to deal with a *different problem* than classical Schema Alignment. Source schemas can only be built a-posteriori from data, and do not reflect the actual structure of the data. Instead, we addressed the problem of *Instance-level attribute alignment*,

i.e., alignment of attributes in every single instance.

We coped with the problem of creating an input dataset that reflected the actual challenges of this domain, and build a comprehensive and valid ground truth.

We built these input and evaluation data in the context of a broader project called ALASKA benchmark[1], whose goal is to define a general benchmark for evaluating performances of data integration and knowledge graph augmentation tasks, focusing on product domain [49].

We performed extended experiments for RaF-AIA using these data. We showed robustness and flexibility of RaF-AIA, which provides good results under different configurations, and even altering input data and using different subset of sources. We also evaluated individually different components of the algorithm, to show their contribution and the importance of each of them. We compared the results of RaF-AIA with baselines and existing approaches in the state of the art, proving that it is the more adapted tool to cope with this specific issue. We also partitioned the dataset under different criteria, and showed the performance of RaF-AIA in each specific subset, proving its ability to provide good results on different type of attributes and sources.

7.1 Future Work

Product graph construction In order to build a unified view on product specifications, in form of a product graph, there are additional tasks that we must tackle.

- Extraction of atomic values from noisy and mixed attributes values. The tagging component of RaF-AIA is currently used as an intermediate step for attribute alignment. However, it represents a first step towards the development of a data extraction technique that works at the finer level of mixed attribute values.

- Attribute mapping: in RaF-AIA, our goal was to align attributes that presented data under the same format, even if under different patterns of representation. To complete integration we should tackle differences of format, such as different

[1] www.di2kg.inf.uniroma3.it

languages or unit of measurements, aligning attributes with different formats and defining mappings between them.

Extend the Alaska Benchmark The Alaska Benchmark is now focused on integration of product specifications. We want to extend it to other domains and other data integration tasks.

Maintaining the Benchmark Product Dataset: Addressing the Velocity Challenge In March 2018, we have checked all the URLs of the pages of the Alaska dataset. We have observed that just 30% of the original pages and 37% of the original sources are still valid (we consider a source valid if it contains at least one working URL). We also performed an extraction of the product specifications. We obtained complete specification from just 20% of the pages.

These numbers clearly indicate that the velocity dimension affects all the tasks of the pipeline. Developing solutions to collect snapshots over regular time intervals and perform data integration over time can open intriguing research directions. While some activities, such as checking the appearance/disappearance of sources can be done on monthly basis, others, such as crawling websites to check appearance/disappearance of pages and changes in the pages should be performed more frequently. To this end, the development of efficient incremental solutions for source discovery and web crawling represent interesting research directions.

As our experiments emphasize, data extraction rules are brittle over time. The development of wrappers resilient to changes in the pages has always been a primary goal in data extraction research. A dataset with multiple snapshots over a long interval of time, as the one that we have advocated above, could serve as a benchmark for data extraction solutions.

Beyond Product Specifications So far we have considered the extraction of the identifiers and of the specifications. However important data that complete the product description are price and reviews. Challenging issues for the extraction of price are

to distinguish the price of the principal product in the page from the prices of other products, such as suggested product, similar products, and the actual price from discounts or list price. Reviews represent important information in many applications. An interesting problem is how to combine structured data from the specification with the unstructured data of the reviews.

Bibliography

[1] Alon Halevy, Anand Rajaraman, and Joann Ordille. Data integration: the teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, pages 9–16. VLDB Endowment, 2006.

[2] Amit P Sheth and James A Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys (CSUR)*, 22(3):183–236, 1990.

[3] Xin Luna Dong and Divesh Srivastava. Big data integration. In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 1245–1248. IEEE, 2013.

[4] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment*, 5(7):680–691, 2012.

[5] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Diadem: thousands of websites to a single database. *Proceedings of the VLDB Endowment*, 7(14):1845–1856, 2014.

[6] Xin Luna Dong. Challenges and innovations in building a product knowledge graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2869–2869. ACM, 2018.

[7] Xin Luna Dong. Building a broad knowledge graph for products. In *Proceedings*

of the 35th International Conference on Data Engineering (ICDE), pages 25–25. IEEE, 2019.

[8] Anitha Kannan, Inmar E Givoni, Rakesh Agrawal, and Ariel Fuxman. Matching unstructured product offers to structured product specifications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 404–412. ACM, 2011.

[9] Hoa Nguyen, Ariel Fuxman, Stelios Paparizos, Juliana Freire, and Rakesh Agrawal. Synthesizing products for online catalogs. *Proceedings of the VLDB Endowment*, 4(7):409–418, 2011.

[10] Luciano Barbosa, Valter Crescenzi, Xin Luna Dong, Paolo Merialdo, Federico Piai, Disheng Qiu, Yanyan Shen, and Divesh Srivastava. Big data integration for product specifications. *IEEE Data Eng. Bull.*, 41(2):71–81, 2018.

[11] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.

[12] Xin Luna Dong. How far are we from collecting the knowledge in the world? In *Keynote at 19th International Workshop on Web and Databases*. ACM, 2016.

[13] Xin Luna Dong Divesh Srivastava. *Big Data Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, March 2015.

[14] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm. *Schema Matching and Mapping*. Springer Science & Business Media, 2011.

[15] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[16] Federico Piai, Paolo Atzeni, Paolo Merialdo, and Divesh Srivastava. Instance level attribute alignment in heterogeneous sources. *Under submission*, 2019.

[17] Disheng Qiu, Luciano Barbosa, Xin Dong, Yanyan Shen, and Divesh Srivastava. Dexter: Large-scale discovery and extraction of product specifications on the web. *PVLDB*, 8:2194–2205, 2015.

[18] Valter Crescenzi, Andrea De Angelis, Xin Luna Dong, Donatella Firmani, Maurizio Mazzei, Paolo Merialdo, Federico Piai, and Divesh Srivastava. The alaska benchmark for big data integration. *In preparation*, 2019.

[19] Jingtian Jiang, Xinying Song, Nenghai Yu, and Chin-Yew Lin. Focus: learning to crawl web forums. *Knowledge and Data Engineering, IEEE Transactions on*, 25(6):1293–1306, 2013.

[20] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4(4):219–230, 2011.

[21] Valter Crescenzi and Paolo Merialdo. Wrapper inference for ambiguous web pages. *Applied Artificial Intelligence*, 22(1-2):21–52, 2008.

[22] 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.

[23] Hanna Köpcke, Andreas Thor, Stefan Thomas, and Erhard Rahm. Tailoring entity resolution for matching product offers. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 545–550. ACM, 2012.

[24] Aliaksandr Talaika, Joanna Biega, Antoine Amarilli, and Fabian M Suchanek. Ibex: harvesting entities from the web using unique identifiers. In *Proceedings of the 18th International Workshop on Web and Databases*, pages 13–19. ACM, 2015.

[25] Disheng Qiu, Luciano Barbosa, Valter Crescenzi, Paolo Merialdo, and Divesh Srivastava. Big data linkage for product specification pages. In *Proceedings of*

the 2018 International Conference on Management of Data, pages 67–81. ACM, 2018.

[26] Chenjuan Guo, Cornelia Hedeler, Norman W Paton, and Alvaro AA Fernandes. Matchbench: benchmarking schema matching algorithms for schematic correspondences. In *British National Conference on Databases*, pages 92–106. Springer, 2013.

[27] Won Kim and Jungyun Seo. Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, 24(12):12–18, 1991.

[28] 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.

[29] Peter Mork, Len Seligman, Arnon Rosenthal, Joel Korb, and Chris Wolf. The harmony integration workbench. In *Journal on Data Semantics XI*, pages 65–93. Springer, 2008.

[30] Sergey Melnik, Erhard Rahm, and Philip A Bernstein. Rondo: A programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 193–204. ACM, 2003.

[31] Jaewoo Kang and Jeffrey F Naughton. On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 205–216, 2003.

[32] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M Marlin. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 74–84, 2013.

[33] Patrick Verga, Arvind Neelakantan, and Andrew McCallum. Generalizing to unseen entities and entity pairs with row-less universal schema. *arXiv preprint arXiv:1606.05804*, 2016.

[34] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *Knowledge-based systems*, 70:301–323, 2014.

[35] Christopher Olston, Marc Najork, et al. Web crawling. *Foundations and Trends® in Information Retrieval*, 4(3):175–246, 2010.

[36] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and integration of partially overlapping web sources. *Proceedings of the VLDB Endowment*, 6(10):805–816, 2013.

[37] Peter Christen. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering*, 24(9):1537–1555, 2011.

[38] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208, 2016.

[39] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34. ACM, 2018.

[40] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Integrating conflicting data: the role of source dependence. *Proceedings of the VLDB Endowment*, 2 (1):550–561, 2009.

[41] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[42] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *arXiv preprint arXiv:1503.00302*, 2015.

[43] Xin Luna Dong, Evgeniy Gabrilovich, Kevin Murphy, Van Dang, Wilko Horn, Camillo Lugaresi, Shaohua Sun, and Wei Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *arXiv preprint arXiv:1502.03519*, 2015.

[44] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1231–1245, 2015.

[45] 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.

[46] Stephen A Bloom. Similarity indices in community studies: potential pitfalls. *Mar. Ecol. Prog. Ser*, 5(2):125–128, 1981.

[47] András Schubert. Measuring the similarity between the reference and citation distributions of journals. *Scientometrics*, 96(1):305–313, 2013.

[48] Marios Hadjieleftheriou and Divesh Srivastava. Approximate string processing. *Foundations and Trends® in Databases*, 2(4):267–402, 2011.

[49] Valter Crescenzi, Andrea De Angelis, Dong Xin Luna, Donatella Firmani, Maurizio Mazzei, Paolo Merialdo, and Divesh Srivastava. Alaska: a real-life benchmark for data integration. 2019. In preparation.

[50] Xin Luna Dong and Theodoros Rekatsinas. Data integration and machine learning: A natural synergy. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1645–1650, 2018.

[51] Anhai Doan, Raghu Ramakrishnan, and Alon Y Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.

[52] Donatella Firmani, Barna Saha, and Divesh Srivastava. Online entity resolution using an oracle. *Proceedings of the VLDB Endowment*, 9(5):384–395, 2016.

[53] Arindam Banerjee, Chase Krumpelman, Joydeep Ghosh, Sugato Basu, and Raymond J Mooney. Model-based overlapping clustering. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 532–537, 2005.

[54] Angelika Kimmig, Alex Memory, Lise Getoor, et al. A collective, probabilistic approach to schema mapping. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 921–932. IEEE, 2017.

[55] Panagiotis G Ipeirotis, Eugene Agichtein, Pranay Jain, and Luis Gravano. To search or to crawl? towards a query optimizer for text-centric tasks. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 265–276, 2006.

[56] Stefano Ortona, Giorgio Orsi, Marcello Buoncristiano, and Tim Furche. Wadar: Joint wrapper and data repair. *Proceedings of the VLDB Endowment*, 8(12): 1996–1999, 2015.

[57] Oktie Hassanzadeh, Ken Q Pu, Soheil Hassas Yeganeh, Renée J Miller, Lucian Popa, Mauricio A Hernández, and Howard Ho. Discovering linkage points over web data. *Proceedings of the VLDB Endowment*, 6(6):445–456, 2013.

[58] Donatella Firmani, Sainyam Galhotra, Barna Saha, and Divesh Srivastava. Robust entity resolution using a crowdoracle. *IEEE Data Eng. Bull.*, 41(2):91–103, 2018.

[59] George Papadakis, Georgia Koutrika, Themis Palpanas, and Wolfgang Nejdl. Meta-blocking: Taking entity resolutionto the next level. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1946–1960, 2013.

[60] Parag Agrawal, Arvind Arasu, and Raghav Kaushik. On indexing error-tolerant set containment. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 927–938, 2010.

[61] Chris Anderson and Mia Poletto Andersson. Long tail. 2004.

[62] Maurizio Lenzerini. Data integration: A theoretical perspective. *PODS*, 2002.

[63] Ibrahim Abaker Targio Hashem Zakira Inayat Waleed Kamaleldin Mahmoud Ali Muhammad Alam Muhammad Shiraz Abdullah Gani Nawsher Khan, Ibrar Yaqoob. Big data: Survey, technologies, opportunities, and challenges. *The Scientific World Journal*, 2014.

[64] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of data integration.* Elsevier, 2012.

[65] Mauricio Hernández and Salvatore Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2:9–37, 01 1998. doi: 10.1023/A:1009761603038.

[66] Luis Gravano, Panagiotis G Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, Divesh Srivastava, et al. Approximate

string joins in a database (almost) for free. In *VLDB*, volume 1, pages 491–500, 2001.

[67] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278, 2002.

[68] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1-3):89–113, 2004.

[69] Xin Luna Dong and Felix Naumann. Data fusion–resolving data conflicts for integration. pvldb. *PVLDB*, 2:1654–1655, 08 2009.

[70] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. The return of jedai: End-to-end entity resolution for structured and semi-structured data. *Proceedings of the VLDB Endowment*, 11:1950–1953, 08 2018. doi: 10.14778/3229863.3236232.

[71] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. Jedai: The force behind entity resolution. pages 161–166, 11 2017. ISBN 978-3-319-70406-7. doi: 10.1007/978-3-319-70407-4_30.

[72] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1):1282–1293, 2009.

[73] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment*, 9(9):684–695, 2016.

[74] George Papadakis, George Alexiou, George Papastefanatos, and Georgia Koutrika. Schema-agnostic vs schema-based configurations for blocking meth-

ods on homogeneous data. *Proceedings of the VLDB Endowment*, 9(4):312–323, 2015.

[75] William W Cohen, Pradeep Ravikumar, Stephen E Fienberg, et al. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, volume 2003, pages 73–78, 2003.

[76] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. A survey of current link discovery frameworks. *Semantic Web Journal*, 8, 12 2016. doi: 10.3233/SW-150210.

[77] Pei Li, Xin Dong, Andrea Maurino, and Divesh Srivastava. Linking temporal records. *PVLDB*, 4:956–967, 01 2011.

[78] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. Pay-as-you-go entity resolution. *IEEE Transactions on Knowledge and Data Engineering*, 25 (5):1111–1124, 2012.

[79] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. Progressive duplicate detection. *Knowledge and Data Engineering, IEEE Transactions on*, 27:1316–1329, 05 2015. doi: 10.1109/TKDE.2014.2359666.

[80] Norases Vesdapunt, Kedar Bellare, and Nilesh Dalvi. Crowdsourcing algorithms for entity resolution. *Proceedings of the VLDB Endowment*, 7(12):1071–1082, 2014.

[81] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J Franklin, and Jianhua Feng. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 229–240, 2013.

[82] Jóakim v. Kistowski, Jeremy A Arnold, Karl Huppler, Klaus-Dieter Lange, John L Henning, and Paul Cao. How to build a benchmark. In *Proceedings of the 6th*

ACM/SPEC International Conference on Performance Engineering, pages 333–336, 2015.

[83] Karl Huppler. The art of building a good benchmark. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 18–30. Springer, 2009.

[84] Susan Sim, Steve Easterbrook, and R.C. Holt. Using benchmarking to advance research: A challenge to software engineering. pages 74– 83, 06 2003. ISBN 0-7695-1877-X. doi: 10.1109/ICSE.2003.1201189.

[85] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering web pages based on their structure. *Data & Knowledge Engineering*, 54(3):279–299, 2005.

[86] Rakesh Agrawal and Samuel Ieong. Aggregating web offers to determine product prices. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 435–443. ACM, 2012.

[87] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Supporting the automatic construction of entity aware search engines. In *Proceedings of the 10th ACM workshop on Web information and data management*, pages 149–156. ACM, 2008.

[88] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[89] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[90] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[91] Robert Meusel, Peter Mika, and Roi Blanco. Focused crawling for structured data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1039–1048. ACM, 2014.

[92] Rahul Gupta and Sunita Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment*, 2(1):289–300, 2009.

[93] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[94] Jayant Madhavan, Shawn R Jeffery, Shirley Cohen, Xin Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. CIDR, 2007.

[95] Nilesh Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 335–348. ACM, 2009.

[96] Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, volume 1, pages 109–118, 2001.

[97] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 337–348. ACM, 2003.

[98] Marnix de Bakker, Flavius Frasincar, and Damir Vandic. A hybrid model words-driven approach for web product duplicate detection. In *International Conference on Advanced Information Systems Engineering*, pages 149–161. Springer, 2013.

[99] Petar Petrovski, Volha Bryl, and Christian Bizer. Integrating product data from websites offering microdata markup. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 1299–1304. ACM, 2014.

[100] Vishrawas Gopalakrishnan, Suresh Parthasarathy Iyengar, Amit Madaan, Rajeev Rastogi, and Srinivasan Sengamedu. Matching product titles using web-based enrichment. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 605–614. ACM, 2012.

[101] Petar Petrovski and Christian Bizer. Extracting attribute-value pairs from product specifications on the web. pages 558–565, 2017.

[102] Nikhil Londhe, Vishrawas Gopalakrishnan, Aidong Zhang, Hung Q Ngo, and Rohini Srihari. Matching titles with cross title web-search enrichment and community detection. *Proceedings of the VLDB Endowment*, 7(12):1167–1178, 2014.

[103] Andrea Horch, Holger Kett, and Anette Weisbecker. Matching product offers of e-shops. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 248–259. Springer, 2016.

[104] Xiangnan He, Ming Gao, Min-Yen Kan, and Dingxian Wang. Birank: Towards ranking on bipartite graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):57–71, 2017.

[105] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems*, 65:137–157, 2017.

[106] Pankaj Gulhane, Amit Madaan, Rupesh Mehta, Jeyashankher Ramamirtham, Rajeev Rastogi, Sandeep Satpal, Srinivasan H Sengamedu, Ashwin Tengli, and Charu Tiwari. Web-scale information extraction with vertex. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1209–1220. IEEE, 2011.

[107] Hung-sik Kim and Dongwon Lee. Harra: fast iterative hashed record linkage for large-scale data collections. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 525–536. ACM, 2010.

[108] Kostas Stefanidis, Vassilis Christophides, and Vasilis Efthymiou. Web-scale blocking, iterative and progressive entity resolution. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 1459–1462. IEEE, 2017.

[109] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, and Ashwin Tengli. Exploiting content redundancy for web information extraction. *Proc. VLDB Endow.*, 3(1-2):578–587, September 2010. ISSN 2150-8097.

[110] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. A framework for learning web wrappers from the crowd. In *Proceedings of the 22nd international conference on World Wide Web*, pages 261–272. ACM, 2013.