

# Chapter 1

## Introduction

Storing and accessing information is one of the most important goals in computing. The data can be in many shapes, sizes, and formats, and each of these **data instances** is collected in a repository. **Open data** is one such repository that contains a collection of data instances created and published by many individual authors, some of them government agencies. The rich repertoire of information in open data can be accessed by the general public without paying for the data. Along with the data instance itself, the repository also contains **metadata** for the data instance which can help someone to quickly understand what information is stored. Since each data instance typically stores different data values and even different kinds of information<sup>1</sup>, the metadata is very useful to describe and summarize the data stored, so that a user can browse a large number of data instances without too much effort trying to understand them. In general, the functions of metadata are explanation, summarization, and provenance [1]. The metadata can also help a user write queries on the data to answer his or her questions. The user or machine can easily perform searching and filtering that uses the metadata to group all the similar data into one place and access them together. At the same time, the metadata can aid machines to organize data, and track where the data came from, and link back to the original sources.

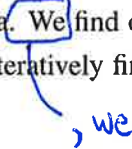
---

<sup>1</sup>For example, you can have 7 instances of the same related table—a snapshot of each day of this past week.

1. Each table contains a list of likely incomplete text items (metadata tags) describing the contents of the data instance. The list can be empty.
2. The authors made an attempt at creating the metadata, and it was not generated without inspection; therefore, we trust the provided metadata.
3. The goal of augmenting metadata tags for a table is for the user to get approximate answers quickly, but not precise answers. That is, metadata tags are augmented in a pay-as-you-go fashion.
4. The user already knows what kinds of metadata are useful and interesting to look at; therefore, we just have to augment tags that the user needs.
5. All attributes in one schema are independent of one another. This assumption may not be true, but it is plausible and allows us to simplify our problem.
6. The set of all topics in the repository are the only topics used for augmenting tables, that is, we do not create new topics that did not already exist in the repository. This assumption reduces a lot of uncertainty in our evaluations.

## 1.9 Contributions

The contributions we make in this thesis are the following:

1. We give a different perspective on the problem of augmenting tags in tables with incomplete metadata. We propose an iterative approach, where given a base table with incomplete metadata,  we find other tables in the repository similar to the base table, and then iteratively find as much overlap between all related tables as possible.
2. We give a detailed explanation of why high-quality metadata can reveal new information to the user and how it can help organize existing data in a repository. We outline that it is easy for users to navigate a repository, perform filter and search, and understand data and metadata of tables without looking at the values in the tables. Metadata tags also allows more automation in that they increase precision and recall for machine operations.

document.

In [33], <sup>notes</sup> topics can be extracted from academic research papers. The values of many attributes are part of a natural language which can also be found as part of English sentences. These values may not be found in ontologies. In order to find new topic words from English sentences, pre-created templates are used to identify distinct concepts in the text. The metadata can be augmented with these new topic words.

If one needs to extract how the topics are related, existing literature in **ontology** extraction is able to find such relationships in sentence-like text data. An ontology is a collection of related concepts. Each concept has a number of properties, and each concept is related to other concepts based on how many properties the concepts share [10]. The concepts, properties, and relationships together can be represented in a graph. For example, chemical reaction networks have chemicals as concept nodes; pH<sup>1</sup> and decay rate of a chemical are properties of a concept; and interactions between chemicals are relationship edges. Each concept in an ontology is rich in semantics when it has many properties and relationships with other concepts. In [25], an iterative method was proposed to extract ontological relationships from text data. Beginning with seed relationships extracted from a corpus, they iteratively induced more parent-child ISA or HASA relationships by using a thesaurus to discover relationships between pairs of words (or phrases) in the corpus. We note that if the document is not based on sentences, but rather on tabular data, it is difficult to apply this iterative approach.

When there are too many concepts, one would store them in a database so that information can be retrieved in a systematic way. A **knowledge base** is a database storing a collection of facts that can be understood and processed by humans or machines [50]. A knowledge base can be in many forms, such as a domain dictionary storing definitions of medical terms, an online Wikipedia repository storing facts, and a database storing employee payroll information. The purpose of knowledge bases is to enhance the reusability of the data and aid machines to interpret the data. For tables from the web exhibiting different forms of heterogeneity, it is possible to extract knowledge and construct a knowledge base. In [48], they performed su-

---

<sup>1</sup>Which stands for potential hydrogen and tells us how much hydrogen is in liquids—and how active the hydrogen ion is.

pervised learning with features such as whether the text is bold or whether caption appears below the table, to discover facts such as the title and author of the table. The knowledge base constructed is a collection of these facts from the tables, and these facts can be queried.

Without any training data, it is still possible to perform *knowledge extraction* from tabular data. In [43], they performed clustering of schema attributes, and assigned to each cluster a topic from a given vocabulary created by the community of authors. We will discuss this work in detail in Section 2.2.1. We will then discuss another work that requires training data [8] in Section 2.2.2, that takes tabular data without headers as input and predicts a header for each column.

The complexity of data and metadata can be reduced by summary generation methods. In [20], heterogeneous text collections are summarized by omitting redundant and irrelevant information, the authors relied on user feedback to improve the accuracy of the text summaries. Schema summarization summarizes the schema in the metadata [49]. The authors represented schemata as graphs and relied on foreign key dependencies and the link distance between two elements in the graph to create a less complex graph. We will discuss this work in more detail in Section 2.2.3.

. The

### 2.2.1 Clustering to reduce complexity

A project on integration of data, the OpenII project [43], proposed to perform clustering on the schema attributes of multiple schemata in the repository to find a common vocabulary for the repository. As explained in [43], a vocabulary is created by a community of authors in advance, and each cluster of attributes is assigned a topic from the vocabulary. It is then possible to map every attribute in each cluster with the assigned topic. The input to clustering is a set of correspondences for the schemata in the repository, where each correspondence is a pair of attributes between two schemata and a similarity score, and clustering is performed with the constraint that disallows attributes from the same schema to be in one cluster. The final set of topics for the schemata becomes the summary of the entire repository. For their approach to work, they assumed that correspondences on attributes are already available as input, and a vocabulary can be readily used for labeling each

global-as-view (GAV), and global-and-local-as-view (GLAV). The term *local* refers to the individual schemata, and *global* refers to the mediated schema, since a schema in a data instance is local relative to the mediated schema. LAV, GAV, and GLAV enforce different cardinality constraints between the schemata and the mediated schema. LAV mappings have a cardinality of many-to-one between mediated schema attributes and local schema attributes. GAV mappings have a cardinality of one-to-many. GLAV mappings have a cardinality of many-to-many.

We will talk about one work of data integration [24], where LAV mappings created for each schema act as the descriptions of the individual data instances via the mediated schema. The heterogeneity between schemata is addressed by the mediated schema and mappings with individual schemata. Using Figure 1.1 and Figure 1.2 as an example, we let the schemata be:

*Park*(*park\_name*, *location*, *service\_classification*) and

*ParkSpecimenTrees*(*location*, *park*, *tree\_species*).

Let the mediated schema (and its mediate attributes) be:

*MedSchema*(*park\_name*, *location*, *tree\_species*, *park\_service*),

with LAV mappings (where each attribute of a schema has a mapping with a mediated attribute):

*Park*(*park\_name*: *park\_name*, *location*: *location*, *service\_classification*: *park\_service*),

*ParkSpecimenTrees*(*location*: *location*, *park*: *park\_name*, *tree\_species*: *tree\_species*).

When a user issues a query such as *MedSchema*(*park\_name*), the LAV mappings show that the queries issued to the individual data instances should be *Park*(*park\_name*) and *ParkSpecimenTrees*(*park*).

### 2.3.1 Relational-to-ontology schema mapping

Mediated schema and data mappings can be used in a different way to address similar problems. We discuss one work that uses semantic labeling to assist in the integration of schemata in a repository [46]. In this problem, the semantic labeling act as mappings between the individual schemata and a common ontology, and

the semantic similarity between a pair of values using WordNet's built-in functions. The details have been explained in Section 3.1.2.

We can also compute the semantic similarity between two values using pre-trained word embedding vectors. Words in an attribute are encoded as word embedding vectors using FastText [32, 33], a variant of Word2Vec. A pair of vectors can be compared, and if two words are used under similar contexts, they should also have similar vectors. When comparing two sequences of words, the distance between their vectors is computed.

The minimum form of schema matching used a matching criterion that counts the proportion of overlapping characters of attribute names. A more sophisticated criterion at the character level is an N-gram distance function that computes the similarity between a pair of values [27]. Each value is transformed into a multiset of character n-grams. For example, we use the values '*EastPark*' and '*SouthPark*' to illustrate how to compute the similarity between them. The bigrams of '*EastPark*' are ['Ea', 'as', 'st', 'tP', 'Pa', 'ar', 'rk'], and the bigrams for '*SouthPark*' are ['So', 'ou', 'ut', 'th', 'hP', 'Pa', 'ar', 'rk']. The similarity is the proportion of shared n-grams between the two values, given by:

$$\frac{2 \times |ngrams(value1) \cap ngrams(value2)|}{|ngrams(value1)| + |ngrams(value2)|}$$

Between '*EastPark*' and '*SouthPark*', there are 3 shared bigrams ['Pa', 'ar', 'rk']. The similarity is therefore  $6/15 = 0.4$ .

Another variation of schema matching is to perform matching at the instance level. Matching at the instance-level compares the data values in the data instances instead of only comparing the attributes. For tabular data instances, matching is done between the table values instead of the table headers, by comparing data values in table columns, with the goal of finding which data column pairs share the most values. Matching at the instance-level is known as *entity matching*. The work in [32] used instance-level matching to determine whether or not a pair of attributes is in the same data domain. Works in [37] used natural language processing (NLP) and deep neural network (DNN) techniques to improve the ability to resolve values for entity matching. If precise matching is required, then each of the values needs to resolve to real-world entities (such as a concept in an ontology). This process

called *entity resolution*. In open data there is typically no real-world entity information provided [4], and therefore instance-level matching cannot be performed with precision. A <sup>coarse</sup> ~~course~~-grained instance-level matching approach uses document similarity measures (such as TF/IDF) [13], by treating each column as a document (with values of a column aggregated into a single document). This approach performs poorly since individual values such as an address of a place requires more effort to identify, and therefore similarity between documents is typically low.

One can also perform matching with a combination of matching criteria within the same algorithm [45]. If an instance-level matching criterion is used in addition to a schema-level matching criterion, the instance-level criterion can add additional evidence for choosing the best correspondences. The reason is that each schema matching criterion specializes at evaluating certain types of data, because each criterion has a distinct similarity function. For example, a criterion that counts the proportion of word overlaps in attribute names is likely to produce different similarity values than a criterion that computes the semantic distance between words using a dictionary.

There are two ways to combine different matching criteria within the same algorithm, *composite* and *hybrid*. Composite matching combines the output attribute pairs from multiple independent matchings by merging the different sets into one set. For example, the set of pairs from one matching (with some criterion) is combined with the set from another matching (with another criterion) by removing conflicting pairs. Hybrid matching uses multiple matching criteria within the same matching algorithm. For each pair of attributes compared, a score is computed using multiple matching criteria, and the two scores are combined. A typical way to combine the scores is to compute a weighted average, where the weights are provided before matching [6]. The weights can be trained by linear regression [14] such that the more effective criterion is given a greater weight. Other variations of hybrid matching exist. In [18], a decision tree is trained over a number of matching criteria, where the next matching criterion to use is decided based on the score produced by a previous score. In [31], a neural network is trained that maps attribute features to Euclidean space. The vectors in Euclidean space are compared to compute the similarity scores.

relationships to make data more searchable and understandable.

Once we have performed table search and table recommendation, a group of tables is found to be related to a base table, thereby organizing the metadata of the existing tables, and finding relatedness.

Therefore, we need methods to organize the metadata and discover the relatedness.

Our main algorithm uses the table search outlined in [32] to approximate tables that are relevant to the base table.

We emphasize that augmenting metadata is only possible when a group of related tables can be found. A user who manually searches for related tables would encounter the same challenge.

Finding the interesting tags can be achieved by table searching, adopted from [33] and [15], which we explained in detail in Section 3.4.1 and Section 3.4.2.

Partitioning was explained in Section 3.5.1. We explained how we normalize the tags using the semantic labeling, briefly in Section 3.3, and more in detail in Section 4.4. We can iteratively augment metadata of the base table and the tables we retrieved.

After we retrieve one candidate table, we perform schema matching again between the base table and the candidate table in order to augment metadata tags. We use the matched topic pairs from table search to obtain candidate attribute pairs, and then perform attribute-to-attribute schema matching to verify that each pair of attributes belongs to the same domain. For example, given that the overlapping tag is parks, with semantic labeling of (*parks*, *Parks.park\_name*) and (*parks*, *ParkSpecimenTrees.park*), we know that the pair of attributes (*Parks.park\_name*, *ParkSpecimenTrees.park*) should be compared to verify that they are similar. Once the correspondence between the pair of attributes is confirmed, we can confirm that the tag *parks* is a correct tag for both tables.

The *iterative method* in Figure 4.4 takes  $s_q, S, D, L, \phi_{ts}, \phi_{pt}, \phi_{sc}$  as input, where  $D$  is the set of all data instances in the repository,  $L$  is the set of text items describing the data instances,  $\phi_{ts}$  is a threshold used for schema matching during the table search step,  $\phi_{pt}$  is a threshold used for schema matching during the partitioning step, and  $\phi_{sc}$  is a threshold used for schema matching during the verification step after items are placed in partitions.



### 4.5.3 Iterative method example

Let the base table schema be *Parks*(*park name*, *location*). Let its existing tags be [*environment*, *green*, *parks*]. After performing semantic enrichment, we have (*park*, 'a piece of open land for recreational use in an urban area'), (*name*, 'a language unit by which a person or thing is known'), (*location*, 'a point or extent in space'), (*environment*, 'the area in which something exists or lives'), (*green*, 'a piece of open land for recreational use in an urban area'), (*parks*, 'a piece of open land for recreational use in an urban area').

We let a schema in the repository be *PayParkingStations*(*meter station*) and its tags be [*parking station*] with semantics (*meter*, 'any of various measuring instruments for measuring a quantity'), (*parking*, 'space in which vehicles can be parked') and (*station*, 'a facility equipped with special equipment and personnel for a particular purpose'). We let another schema be *ParkSpecimenTrees*(*park*) and its tags [*parks*] with semantics (*park*, 'a large area of land preserved in its natural state as public property') and (*parks*, 'a large area of land preserved in its natural state as public property').

After topics of every schemata are collected, we have [*environment*, *green*, *Parks.parks*, *parking station*, *ParkSpecimenTrees.parks*]. We note that two topics may have the same tag name but have different semantics, such as *Parks.parks* and *ParkSpecimenTrees.parks*. We then perform semantic labeling using the semantically enriched attributes and topics. Let the semantic labeling of *Parks* be (*park name*, *Parks.parks*) and (*location*, *environment*), the semantic labeling of *PayParkingStations* be (*meter station*, *parking station*), and the semantic labeling of *ParkSpecimenTrees* be (*park*, *Parks.parks*). We note that *park* from *ParkSpecimenTrees* is labeled with the tag from *Parks*, because the attribute is meant to contain *park names* within a city rather than natural land.

*italics*

With Park as the base table, we first create 3 partitions for the 3 semantic labels, and apply the iterative method. We use the WordNet semantic distance to find topic overlaps between the base table and a table in the repository. For example, we find the overlap between *Park* and *ParkSpecimenTrees* is *Parks.parks*, and there are no overlapping topics between *Park* and *PayParkingStations*. We proceed to partitioning and schema matching between *Park* and *ParkSpecimenTrees*. We

perform comparisons of a candidate topic in *ParkSpecimenTrees* with all the topics in the base table partitions. We find that the candidate topic *Parks.parks* is similar to an existing topic *Parks.parks* in a partition. We now proceed to compare the attribute in the semantic label and all the attributes in the partition. Thus we compare *park* with *park name* using both the N-gram and the WordNet matching criteria. We find that the two attributes are highly similar because the combined score is above a threshold. We note that WordNet compares the word pairs (*park*, *park*) and (*park*, *name*), and takes the higher semantic distance score. We place the candidate topic and the attribute into the partition, now containing topics [*Parks.parks*, *Parks.parks*, *green*] and attributes [*park name*, *park*], where one of the *Parks.parks* is from the *Parks* table and the other is from *ParkSpecimenTrees*.

After the iterative method terminates, the base table and all the related tables are augmented with additional topics from the partitions in the same way as *IterativeMethod*. We note that the resulting topics will be used to compare with a gold standard that we created. We will explain how we compare the augmented topics in our algorithm with the gold standard in Section 5.3.3.

) double-check  
wording

We also have guidelines for creating the list of augmented topics in the gold standard. Our goal is to try to reduce bias. Bias is present when we have augmented too many tags in one data domain, but missed many tags in another domain. This bias is common because data are collected based on human decisions, and people tend to only collect data from sources they are familiar with or they have access to.

The guidelines we give below aim to reduce bias in the gold standard we create. We manually collect tags from each table into a set, and cluster the tags (using N-gram as the distance function) to find groups of similar tags. We then examine each table one by one, while having access to the 700 tags, the clusters of tags, and all tables in the same domain. For existing tags  $L_s$  of the table  $s$ , we find the cluster  $Cluster(L_{sj})$  each tag  $L_{sj}$  is in. For each candidate tag  $L_{sj}$ , we retrieve all tables  $S_j$  already containing tag  $L_{sj}$ , and inspect the data and the metadata of each table  $s'$  in  $S_j$  to decide whether the candidate tag  $L_{sj}$  should be added. We also inspect each table  $s'$  containing the tag  $L_{sj}$  and discover additional similar tags  $L_{s'}$  that could be added to the gold standard of  $L_s$ . We built a user-interactive tool that assists us to perform all of the above. We added any tags that we missed and we removed tags that seemed incorrect after each test run. We did not count these test runs as part of the accuracy calculation.

### 5.3.3 Comparing output augmented tags with gold standard

After tables are augmented with tags using our algorithms, we measure the accuracy on the gold standard we created. To evaluate a given test set on the gold standard, we tailor the gold standard such that it only contains tables in the test set. Given a test set, we show an example of the gold standard compared with the output augmented tags. Let the base table be Parks, and let the repository contain the tables  $\{Parks, ParkSpecimenTrees, DrainageCatchBasins, ParkScreenTrees\}$ . The gold standard for this particular test should only contain tables related to Parks. For this test, the augmented tags for tables  $\{Parks, ParkSpecimenTrees, ParkScreenTrees\}$  are in the gold standard.

Each of the tables in the gold standard contains a set of augmented tags. Let the augmented tags of the *Parks* table contain  $[environment, green, nature, parks, trees]$  in the gold standard. Its original tags are  $[environment, green, nature, parks]$ .

- *italics*

However, we observe that *Improved Iterative* decreased its precision drastically as the mix shifted from 100% to 0%. This suggests that Improved Iterative requires more improvements for future work.

**Table 5.2:** Precision of different algorithms given a mix of related and unrelated tables in repository

mix	Brute Force	Data Driven	Iterative	Improved Iterative
0%	0.05915	0.1882	0.14285	0.1266
50%	0.22605	0.18335	0.23375	0.2282
100%	0.2064	0.17155	0.21975	0.48445

**Table 5.3:** Recall of different algorithms given a mix of related and unrelated tables in repository

mix	Brute Force	Data Driven	Iterative	Improved Iterative
0%	0.54345	0.1304	0.04345	0.413
50%	0.50515	0.0367	0.02445	0.40615
100%	0.51215	0.02215	0.0147	0.3332

We then performed roll-up of the results in both  $k$  and mix proportion. Improved Iterative has the highest precision of 0.28 while Brute Force has the lowest precision of 0.16, but Brute Force has the highest recall of 0.52 while Iterative has the lowest recall of 0.03. We think that Brute Force has a high recall because it blindly adds all tags it finds. The reason that Iterative has a low recall is because it only uses N-gram as the matching criterion, which fails to identify semantically similar tags.

We ranked the four algorithms based on how well they performed for each test. Of the 12 tests, Improved Iterative ranks first 7 times and ranks second 3 times. The only test it performs much worse on than another algorithm is for  $k=10$ , and the repository contains 9 unrelated tables. We think Improved Iterative cannot correctly reject unrelated tables, and therefore adds too many incorrect tags. The semantic distance is affected by the initialization step, which we may need to improve in the future.

Using the precision and recall, we now compute the F1 score for the four algorithms. The plot for the F1 score at different  $k$  is shown in Figure 5.3. The plot for

italics

and vertical data exploration explores data that are the daughters. For example, one sense of the word *park* is '*a piece of open land for recreational use in an urban area*'. Its synonyms are commons and green, and they could be the sisters or the daughter of park. The authors argued that more related information is found by exploring the daughters than in the sisters. We therefore leave the experiment of adding synonyms as future work.

As another improvement, we can include adjectives to the context because overlaps in adjectives may also contribute to word sense disambiguation. We also intend to count the number of non-overlapping words, and select the word sense with the least number of non-overlapping words. The impact of selecting non-overlapping words will also be assessed as future work.

### 6.1.2 Semantic labeling discussion

We explain how semantic labeling is related to existing data integration techniques. In our work, since a tag can be shared across multiple tables, all the tags act like a mediated schema, where we use semantic labeling between metadata tags and schema attributes to maintain the semantic similarity across different tables. The semantic labeling acts as a source description. Assuming that there is a mediated schema that contains all possible mediated attributes, it is then easier to integrate all the schemata together. Integrated data also allows one to ask questions on a global mediated schema without the need to understand the individual local schemata.

An application of using mediated schema in matching was proposed, using an attribute dictionary to perform instance matching [5]. The dictionary is a knowledge base that stores attributes and their representative values, as well as probabilities of these values. The final matching between a pair of tables is determined transitively through the dictionary. The matching between a pair of elements involves matching to the dictionary first, then using an algorithm to solve the max-flow problem to find the best match.

### 6.1.3 Table search discussion

The main idea of table searching is to reduce the search space. Techniques that can reduce the search space include source selection [12], partitioning [31], and block-

## 6.7 Related areas not addressed by metadata augmentation

We describe a number of related areas of research, and list the differences between the problems they solved and the problem we aimed to solve. An area of study in information retrieval is query answering, where the goal is to either return a precise or approximate answer (containing specified data in the tables) to a query. We did not address query answering because we only augmented the metadata without considering how a query can be answered using the data and metadata. In our iterative approach, the table search step answers the query of finding all tables related to a given table. Once we have augmented tags for tables, a simple query is to provide a list of tags to find all tables having these tags, which we can implement easily and assess the performance. But if a query is to precisely find all data in a database that satisfy a constraint, then we cannot answer the query trivially.

Within the study of information extraction, we only addressed the problem of inducing additional information using some existing information as a seed. When we have tables and metadata as input, we did not create knowledge from raw data such as logs and sentences; we only improved the quality of the existing metadata. We discovered information from a pool of poorly maintained data and metadata exhibiting semantic heterogeneity. By our studies, we aimed to bridge the gap between schema matching which is the core task of data integration mainly for structured data, and natural language processing (NLP) which is the data integration task mainly for unstructured data. Schema matching performs very little information extraction, but we applied more natural language processing techniques to improve the power of schema matching. At the same time, our end goal of augmenting topics resembles more closely to one of the goals of natural language processing. The algorithm we implemented meets this goal. By word sense disambiguation, we are able to perform schema matching more easily, which formally operated on the data, and the postprocessing at the end of the algorithm reaches the goal of augmenting metadata.

Knowledge graphs and ontologies both represent relations between individuals in the form of triples, very much like the correspondences we created in schema matching. However, the difference is that Knowledge graphs have probabilistic

## Chapter 7

### Conclusion

Data in the modern computing landscape needs to be managed by multiple users, and the complexity of data schemata has grown to an extent that no single user can understand every detail. When a user needs to understand data not managed by him or her, we proposed attaching semantics-rich metadata (especially the metadata tags) to each separate data instance, which serves as a summary of the data. We first reviewed a number of works related to data summarization, searching data, and comparing data by matching. We then discussed how the metadata tags with enriched semantics provide more evidence and context to these works. We aimed to solve the issue of incompleteness in a repository containing open data, and then developed an iterative approach that augments the metadata of tabular data in open data. We made assumptions that allow us to augment metadata in a pay-as-you-go fashion, where we only augmented tags for tables related to a given base table. We showed that our algorithm outperforms the baseline algorithm, but also indicated many areas we still need future work.

that