Dataset Discovery and Exploration: A Survey

DOI:
10.1145/3626521

Document Version
Accepted author manuscript

Dataset Discovery and Exploration: A Survey

NORMAN W. PATON, JIAOYAN CHEN, and ZHENYU WU, Department of Computer Science, University of Manchester, UK

Data scientists are tasked with obtaining insights from data. However, suitable data is often not immediately to hand, and there may be many potentially relevant datasets in a data lake or in open data repositories. As a result, data discovery and exploration are necessary, but often time consuming, steps in a data analysis workflow. Data discovery is the process of identifying datasets that may meet an information need. Data exploration is the process of understanding the properties of candidate datasets and the relationships between them. Data discovery and data exploration often go hand in hand, and benefit from tool support. This paper surveys research areas that can contribute to data discovery and exploration, in particular considering dataset search, data navigation, data annotation and schema inference. For each of these areas, we identify key dimensions that can be used to characterize approaches and the values they can hold, and apply the dimensions to describe and compare prominent results. In addition, by surveying several adjacent areas that are often considered in isolation, we identify recurring techniques and alternative approaches to related challenges, thereby placing results within a wider context than is generally considered.

CCS Concepts: • **Information systems → Data federation tools**; **Mediators and data integration**; **Data cleaning**.

Additional Key Words and Phrases: data search, data navigation, data annotation, data lake, schema inference

1 INTRODUCTION

There are ever more repositories containing structured or semi-structured data that are candidates for future analysis. These may be collected within organisations in data lakes [38], published as open data by public sector organisations [5], or obtained from the Web, either from Web tables [29] or by extraction from the deep Web [36]. Analyses can be applied to these data directly, or can combine external and internal data to obtain insights that are not provided by either in isolation [61].

In principle, this data deluge presents significant opportunities, but evidence suggests that unlocking value from large data repositories comes only with significant costs. It is often quoted that data scientists, who are hired to obtain insights from data, spend in the region of 80% of their time on data preparation, and thus rather less on actually analysing the data and interpreting the results. This is supported by survey evidence [76], where it is also reported that 19% of a data scientist's time is spent discovering and selecting suitable datasets, the focus of this survey.

To support data scientists and data engineers in the management and use of data repositories, a variety of metadata models have been developed [31, 43, 83], which in turn underpin data catalogs [39, 43, 88]. Such catalogs often support policies around data governance [51], and thus record the provenance of datasets [64] along with usage information [51], and potentially the results of statistical data profiling over individual data sets or groups of data sets [1, 62]. However, in a setting where there may be millions of datasets, and where these datasets may change rapidly [39], it is impractical to manually curate data catalogs with rich descriptions of all the available data and it can still be challenging to understand what data is available to support a specific task.

Authors' address: Norman W. Paton, norman.paton@manchester.ac.uk; Jiaoyan Chen, jiaoyan.chen@manchester.ac.uk; Zhenyu Wu, zhenyu.wu@postgrad.manchester.ac.uk, Department of Computer Science, University of Manchester, Oxford Road, Manchester, UK, M13 9PL.

0360-0300/2023/10-ART
https://doi.org/10.1145/3626521

School1

Name	Postcode	Type	Town	Gender	Borough
Goresbrook	RM9 4TX	Free	Dagenham	Mixed	Barking and Dagenham
St Mary's	NW3 6PG	Independent	London	Mixed	Camden
City of London	EC4V 3AL	Not applicable	London	Boys	City of London

School2

Name	Address	Town	Postcode	Phase	LA_Name
Argyle	Tonbridge Street	London	WC1H 9EG	Primary	Camden
Millenium	50 John Harrison Way	London	SE10 0BG	Primary	Greenwich
Marshalls Park	Pettits Lane	Romford	RM1 4EH	Secondary	Havering

School3

Name	Pupils	A*-C	At Least One	Average Points	LA_Name
St Mary's	167	34%	99%	325.5	Camden
Riverston	39	62%	100%	384.8	Greenwich

Deprivation

Code	Name	Income Deprivation Rate	Deprivation Rate Quintile
E09000002	Barking and Dagenham	19.4%	1
E07000066	Camden	13.8%	2

Brownfield Sites

Id	Organisation	Site Name Address	Hectares	Planning Status
670	Barking and Dagenham	9-10 The Triangle	0.05	Permissioned
19	London Borough of Camden	Park Village East/ Augustus St	0.01	Not permissioned

Fig. 1. Running example with Open Government Data.

As a result, there has been significant effort invested in the development of techniques that, for example, support search over datasets that have been minimally annotated, automate annotation of datasets with terms from ontologies, facilitate exploration of relationships between datasets, identify recurring domains, and infer recurring structural patterns. This research all seeks to support data scientists in identifying datasets that, individually or together, may be useful for a task at hand. The associated body of research results on data discovery and exploration at scale has yet to be reviewed as an integrated body of work.

In terms of the applicability of this work, the market for data catalogs, which may be expected to build on the results of such analyses, has been predicted to grow from $523M in 2020 to $1,788M in 2026[1], reflecting the growing business need for a consolidated view of the available data. To illustrate the concepts and techniques associated with data discovery and exploration, we will use the running example in Figure 1, which is derived from several UK Open Government sites[2].

We assume throughout that we are dealing with datasets that have an explicit structure, represented in Figure 1 by tables with named attributes, but most techniques discussed can also be applied with minimal adaptation to semi-structured models such as XML or JSON. We do not consider text or specialised data types such as videos or images. To reflect this, we will use the word *dataset* to represent a table or a semi-structured document, for example in JSON or XML, except where the context is specific to an individual paradigm.

Data discovery and exploration is challenging, addressing the requirement to identify several, potentially interrelated datasets within huge repositories, where the datasets within the repositories were likely produced independently of each other. Independently produced datasets tend to manifest a variety of heterogeneities [52] that complicate the data environment facing data scientists and engineers. For example, such heterogeneities may involve inconsistent naming of attributes, capturing a concept using one table or attribute or several, and inconsistent formatting of values. There is no silver bullet for understanding and resolving these heterogeneities

[1]https://www.mordorintelligence.com/industry-reports/data-catalog-market

[2]Specifically: https://data.london.gov.uk, https://webarchive.nationalarchives.gov.uk/, https://www.gov.uk/government/ collections/english-indices-of-deprivation

at scale, so data discovery and exploration involves a collection of different tasks and techniques. In describing these tasks, we use the following notation which assumes tabular data, although in practice some surveyed approaches use different data models. A *collection* C consists of a set of datasets $d \in C$. Each such dataset is associated with a set of $n > 0$ attributes, which we denote together as $d(a_1, \ldots, a_n)$ or individually as $d.a_i$.In this survey, we focus on the following inter-related areas:

Dataset Search: Given a repository of data, a search aims to retrieve datasets that, individually or together, satisfy an information need. Thus, given a collection of datasets C and a search *request* R, $\textsc{Search}(C, R) \rightarrow S$ returns a collection $S \subseteq C$. There may additionally be a function Rank or Score applicable to the result collection S that imposes an order on the results of the search reflecting their similarity to the request R. A request may be expressed in different ways. For example, a request could be based on keywords; with reference to Figure 1, a search for *school* would likely retrieve all three school-related tables, though a search for *school attainment* would likely not be able to establish that *School3* is about attainment, whereas *School1* and *School2* are not. Alternatively, given one table, a search could look for similar tables; in relation to Figure 1, a search with *School1* might be expected to identify a high level of similarity with *School2*. The dataset search techniques surveyed assume there is access to a collection of datasets, and so do not support repository discovery.

Data Navigation: Having identified a dataset, for example as a result of a search process, it is often useful to be able to explore related datasets that may provide additional information or context. Relationships used in navigation are typically between the attributes of datasets. Thus, given a dataset $d(a_1, \ldots, a_n)$ and a collection of datasets C, there there may be an operation of the form $\textsc{RelatedAttributes}(d.a_i, C) \rightarrow S$ that returns a collection of attributes from datasets in C that are related to $d.a_i$. Different approaches infer relationships with different semantics, and as with search, results may associated with ranks or scores. For example, in Figure 1, *School1.Borough* may be a candidate for joining with *Deprivation.Name*, thus providing additional information about income in the vicinity of a school. However, joining is not the only relationship it may be interesting to explore. It may also be useful to know which columns share the same domain[3] as a given column. For example, in Figure 1 the attributes *School1.Town* and *School2.Town* seem to belong to the same domain, and for the values given, it is possible that *Brownfield Sites.Organisation* and *School1.Borough* belong to the same domain, even though there are representational inconsistencies relating to *Camden*.

Data Annotation: In a large repository, there are likely to be a variety of naming conventions, for example because the data was originally produced by different publishers. This is a nuisance, as it places the burden on the user of resolving inconsistencies as part of the process of selecting data for analysis. Data annotation is the process of associating intensional or extensional data items with a term from a vocabulary or a concept from an ontology. Thus, assuming we have a *vocabulary* V that defines a collection of terms $t \in V$, there may be operations of the form $\textsc{AnnotateDataset}(d, V) \rightarrow A$ or $\textsc{AnnotateAttributes}(d.a_i, V) \rightarrow A$ where $A \subseteq V$ that, respectively, annotate datasets or attributes within datasets with terms from the vocabulary V. It is then possible, for example, to search or browse repositories using the annotations; this has the effect of linking a potentially *ad hoc* collection of datasets in a repository to a well defined description of generic or application-specific concepts. For example, in Figure 1 the columns *School1.Borough*, *Brownfield Sites.Organisation* and *School3.LA_Name* could all be annotated with the type *GovenmentOrganization* from *Schema.org*[4].

[3]A *domain* is the set of values that a property may legitimately hold. For example, if the domain is *color* than legitimate values would include *red*, *green* and *black*, but not *California*, which belongs (among others) to the domain of *US States*.
[4]https://schema.org

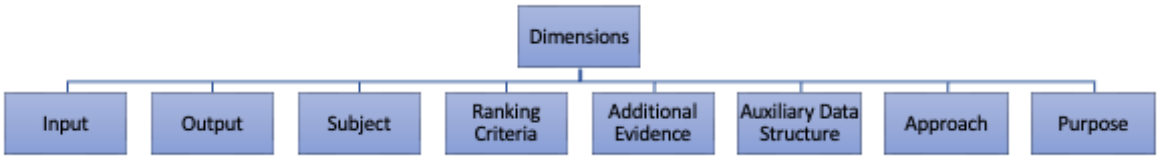

Fig. 2. The dimensions

Schema Inference: Data integration can be associated with the notion of a *global schema* – a schema that an end user interacts with, and behind which the heterogeneity of different sources is hidden, for example using mappings expressed as views [56]. As such, given a collection of datasets C, there may be an operation of the form InferSchema$(C) \rightarrow S$ that returns an inferred schema S that defines a collection of dataset definitions where each such definition may be of the form $d(a_1, \ldots, a_n)$. Note that, typically, the objective is that the inferred schema should be smaller than the underlying collection (i.e., $|S| \ll |C|$) as the inferred schema should contain a single dataset definition for, say, *school* whereas there may be many different tables about schools in C. The inferred schema may also be associated with a function Mappings that describes how the dataset definitions in the inferred schema relate to the datasets in C. Such an inferred schema provides a consolidated but abstract representation of a complex data environment. For example, in Figure 1 the tables *School1* and *School2* could be associated with a *School* table in a global schema that contains the attributes of both *School1* and *School2*, except in the case of *School1.Borough* and *School2.La_Name* which are represented by a single attribute in the global schema, such as *School.GovenmentOrganization*.

Together, the above activities can support data discovery and exploration. *Data discovery* is the process of identifying datasets that may meet an information need. This may, for example, be done directly through a search, by navigating from related datasets, or by browsing the datasets with a specific annotation. As such, data discovery can benefit directly from the techniques covered in this survey. *Data exploration* is the process of understanding the properties of candidate datasets and the relationships between them. This may, for example, be carried out by exploring the relationships of a given dataset, by viewing shared annotations at dataset or attribute level, or by exploring relationships that are shared by several datasets in an inferred schema. As such, data exploration can benefit directly from the techniques covered in this survey. In practice, data discovery and data exploration often go hand in hand.

The areas surveyed can be considered to represent early stages in data integration, that can inform the construction of application-specific data integration pipelines, for example that make use of Extract-Transform-Load platforms [90]. However, we note that data discovery and exploration is a prerequisite to analysis not only in enterprise settings [70], but is also a key activity for individual data scientists using notebooks, who also stand to benefit from access to tool support for dataset discovery and exploration [13, 102].

To characterise the state of the art in dataset discovery and exploration, in this paper we make the following contributions:

(1) We provide the first survey of automated techniques that span data discovery and exploration, which underpins the population of current and future data catalogs to support data scientists in obtaining value from large data repositories.
(2) We define dimensions for each of Dataset Search, Data Navigation, Data Annotation and Schema Inference that identify key recurring features of these areas.
(3) We apply the dimensions from (2) to compare and contrast key papers in each of these areas, and identify potential gaps and opportunities for future work.

In addressing these objectives, we identify complementary and competing approaches to challenges facing data scientists, clarify the relationships between different technical areas, and highlight directions that stand to benefit from further work. With a view to ensuring that the survey focuses on results that are of good quality and influential, in selecting papers for inclusion in the survey, we have prioritised: (i) papers from well regarded outlets — over 80% of the papers discussed are from top outlets in databases, artificial intelligence or closely related disciplines (e.g., PVLDB, ACM SIGMOD, IEEE ICDE, VLDB J., EDBT, AAAI, ISWC, Semantic Web Journal, ACM SIGIR); (ii) papers that are well cited — over 70% of the papers discussed have over 25 Google Scholar citations, with most of the others being too recent to have obtained many citations; and (iii) papers that cover diverse approaches, so that distinctive and innovative techniques are suitably represented.

The remainder of the paper is structured as follows. In Sections 3 to 6 we introduce key concepts in data discovery and exploration in the form of dimensions, and use the dimensions to describe representative results from each of the areas. Then, in the light of the review material, for each topic we highlight the characteristics of the state-of-the-art and identify open issues. Overall lessons learned are presented in Section 7.

2 DIMENSIONS

The survey uses a collection of dimensions to provide a framework for summarising results in each of the areas covered by the survey, namely Dataset Search, Data Navigation, Data Annotation and Schema Inference. This section introduces the dimensions, which are illustrated in Figure 2. These dimensions form table headers when surveying papers in each area, but typically the specific values that are used to populate the dimensions vary from area to area (e.g., the *Input* to a *Dataset Search* task is not the same as the *Input* to a *Data Navigation* task). Furthermore, not all dimensions apply to every area (e.g., a *Dataset Search* tends to involve *Ranking Criteria*, but this is not the case for *Schema Inference*). The dimensions are as follows:

Input: The input describes the data that is specific to a run of a task, such as the *Keywords* that specify a *Dataset Search*.

Output: The output describes the data produced by a run of a task, for example indicating that an *XML Schema* is the result of a *Schema Inference* task.

Subject: The subject describes the focus of a task, for example indicating that a *Keyword* search is being conducted against the *Header* of each dataset in a collection.

Ranking Criteria: The ranking criteria indicate how results are ordered, where multiple alternative results may be available. For example, where a *Dataset Search* is looking for results that are similar to a given dataset, the results could be ordered by *Header Similarity*.

Additional Evidence: The additional evidence describes existing data resources that may be used in addition to the *Input* to carry out a task. In contrast with the *Input* dimension, additional evidence is not specific to a run of the task. For example, an existing *Ontology* could be used to generalize or rewrite the *Keywords* for many runs of a *Dataset Search*.

Auxiliary Data Structure: An auxiliary data structure is used to support the carrying out of a task. For example, an *LSH Index* can be used to support efficient, scalable and approximate *Dataset Search*.

Approach: A task is carried out using an approach, which makes explicit the algorithm type or method. For example, *Schema Inference* is often initiated by *Clustering* similar datasets from a collection.

Purpose: The purpose specifies the reason why a task is being carried out, where a task could address different goals. For example, in *Schema Inference*, one possible purpose is to produce a format that is precise enough to allow users to *Query* the underlying data resources, and a different purpose is to concisely *Summarize* the resources.

As such, the dimensions seek to capture key features of dataset discovery and exploration tasks, capturing information on: (i) inputs, outputs and output ordering; (ii) how the task is carried out in terms of the evidence

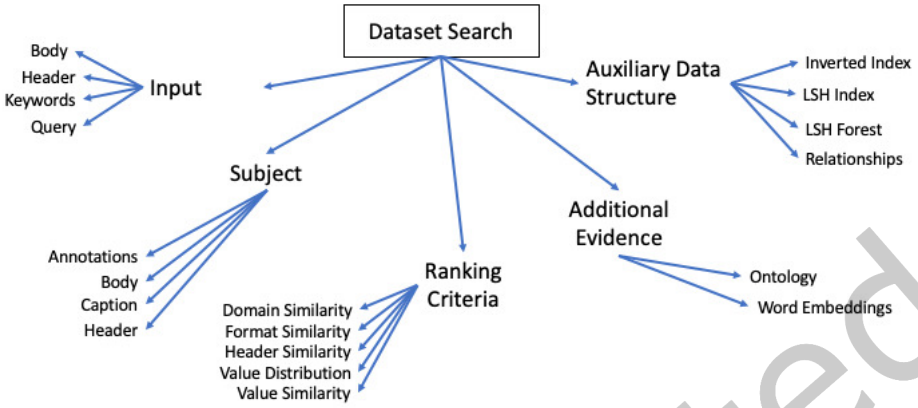

Fig. 3. Data Search dimensions and their values

used, key data structures and algorithmic approach; and (iii) the specific purpose that is being pursued. The specific values for the dimensions used by the different areas are introduced in Sections 3 to 6.

3 DATASET SEARCH

Dataset search is the process of identifying datasets on the basis of some information on what is required. It refers to the task of finding datasets from repositories that are most relevant to given keywords, datasets or queries. Datasets are ranked according to the extent to which they are predicted to satisfy the requirements expressed in the search. In practice, dataset search can be the first step taken by a data scientist when looking for data on which to act (e.g., *I wonder what data there is on schools in England*) or can be used later in the process to supplement earlier discoveries (e.g., *I wonder what other datasets are similar to this one*). Dataset search is constrained by the available information on what is required; for example, when looking for schools in England, there may be a keyword search for *school*, perhaps qualified with location information. In contrast, when looking for something similar to a known dataset, the input to the search is the known dataset. The need to meet such diverse requirements means that there is not a single *best* approach to dataset search; different approaches are complementary, and may be expected to coexist in a data discovery and exploration system. Furthermore, dataset search must provide rapid responses at scale, accommodate imprecise requests, and produce a plausible ranking of candidate solutions.

Chapman *et al.* [21] has previously provided an overview of dataset search, but focuses primarily on keyword search over published metadata. Datasets can also be identified from publications by way of the data citation index [80, 87]. Here, we characterise and compare recent generic results in dataset search and provide equal attention to dataset search for different query input categories, thus complementing the earlier survey and techniques that depend on curation or explicit references.

3.1 Dataset Search Dimensions

In this subsection, we introduce the dimensions used to characterize dataset search techniques; Figure 3 illustrates dimensions and their associated values, on which we now elaborate. The *Output* of a search is always a collection of datasets, as described in Section 1.

Input: Dataset search is assumed to act over large repositories, and to be carried out by a user who has some idea as to what is required. This idea as to what is required constitutes a request, and can be expressed in a variety of forms. As in information retrieval for textual data, a search for relevant structured data can be carried out using *Keywords* that may be compared with features of the datasets in a repository. However, as the focus is on datasets that are structured or semi-structured, data contain *intensional* and *extensional* elements. A search that provides only *intensional* elements, such as the name of the collection or the names of attributes would be considered to be searching using the terms from the *Header*. In contrast, a search that is given the *Body* of a dataset is *extensional*, and would be considered to be seeking to answer the question *what other datasets contain values that are similar to those given*. Although a classical database query, say in *SQL*, assumes familiarity with the *intensional* aspects of a database, which typically are not known with any confidence during dataset search, some languages, such as *SPARQL* [72], can express queries that range over both extensional and intensional data (e.g., to retrieve the types of the resources that have any attribute with a given value), and thus query languages can be designed that express data search tasks.

Subject: The *Input* to the Search, of one of the types described above, is then compared in the search to some features of the datasets in the repository. The *Subject* of the Search may be closely associated with the *Input* to the Search; for example, if the *Input* to the Search is the *Body* of a dataset, there is a good chance that the *Subject* of the Search will also be the *Body* of a dataset, and thus there is a similarity search at the extensional level. However, if the *Input* to the Search is *Keywords*, then these keywords can be matched with the *Header* or *Body* of datasets in the repository, or with supplementary data such as human or computer generated *Annotations* or the textual *Caption* of a Web table.

Ranking Criteria: Search requests often yield multiple results that need to be ranked, in terms of closeness to the input request; this raking tends to involve computing metrics that involve one or several criteria, drawn from: *Header Similarity* – how similar are the names of datasets or their attributes; *Value Similarity* – how similar are the extensional values in the datasets; *Format Consistency* – how consistently are the values in two columns formatted; *Value Distribution* – how consistent is the distribution of numerical values in two columns; and *Domain Similarity* – how consistent are the predicted domains of different columns.

Additional Evidence: A search may use additional information, for example to reduce the impact of representational inconsistencies on the recall of a search. For example, an *Ontology* can identify relationships between terms that can inform either the search or the ranking, and *Word Embeddings* provide vector encodings of textual values that are closer in the vector space for words that are expected to have similar meanings [3].

Auxiliary Data Structures: Search needs to scale, and as a result is likely to depend on data structures that support the efficient evaluation of requests over large repositories. Search may depend directly on an *Inverted Index* to provide a mapping from content, such as words or numbers, to their locations in a dataset. However, exact lookups may provide low recall, so approximate indexes are often deployed. For example, Locality Sensitive Hashing (LSH) is an approximate indexing scheme that builds on hash functions for which the collision probability is higher for values that are more similar [44]. Hash functions have been developed that support different notions of similarity, such as Jaccard and Cosine similarity. An *LSH Index* exploits such hashing for similarity lookup, and *LSH Forest* is a refinement that reduces the need for manual parameter tuning [7]. In addition to indexes, *Relationships* between datasets in a repository may provide

additional information on similarity. For example, in a search where the *Input to the Search* is the *Body* of a dataset, the use of relationships could establish that the given *Body* matches several joined datasets better than any individual dataset.

3.2 Dataset Search Proposals

In this section, we review specific dataset search techniques in the light of the dimensions from Section 3.1; the values of the dimensions for specific techniques are provided in Table 1. In practice, dataset search techniques can be grouped based on the *Input to Search* dimension; the techniques used tend to follow from the input, and we discuss proposals following the values for this dimension.

3.2.1 Keyword-driven Search. Keyword search is a well established paradigm for use with documents that can also be applied to initiate a discovery and exploration process for data. This task returns a ranked list of datasets from the corpus in descending order of their relevance scores with respect to one or several keywords. Keyword search allows users to target data with minimal knowledge of dataset structure and relationships. Instead, users can submit a set of keywords, and the search system returns a ranked list of datasets that are relevant to the query.

As there is an existing survey that reviews results on keyword search for datasets with an emphasis on metadata search [21], we do not cover this ground again here, though we include Google Dataset Search in Table 1, as a prominent example of such an approach. Google Dataset Search [16] provides a keyword-based search over published annotations of Web datasets. The annotations, represented using Schema.org[5] or W3C DCAT[6], are retrieved via Web crawls. The crawled annotations are then processed to resolve inconsistencies in how terms are used in annotations, to reconcile the annotations with a knowledge graph, and to identify replicated data sets. The resulting representation is then indexed and searched using Web search technologies. Such an approach is potentially effective, but requires that datasets are suitably annotated.

Analogous to Web search, it is also possible to apply an information retrieval style search to datasets. For example, OCTOPUS [17] supports keyword searches on the content and context of Web datasets, and returns the top-k most relevant datasets. Using syntactic similarity measures, these datasets are clustered into groups of unionable datasets. In contrast to OCTOPUS, which matches datasets syntactically, more recent solutions tend to prefer semantic matching. For example, TRSS [100] represents both *Input* keyword queries and the datasets that are the *Subject* of the search in a semantic space. Specifically, they consider both word-based searches where the query and the dataset consist of lexical tokens, and entity-based searches where entities in the dataset are terms from a knowledge base that occur in the *subject attribute* of the dataset. The *subject attribute* is considered to capture what a dataset is about (e.g., in Figure1 the *Name* is the subject attribute of the school tables, as the tables are about schools). The resulting terms are then mapped into a vector space using word or graph embeddings, and the similarity between keyword-dataset pairs is used to calculate the ranking.

Semantic matching solutions can also apply deep contextualized language models, such as BERT [27], to match keywords with datasets on the basis of their semantics rather than their syntax. In Natural Language Processing (NLP), BERT has achieved impressive results on a wide variety of tasks (e.g. [2]). TSBERT [23] uses BERT to match keywords and datasets, following a *select-then-rank* framework. Firstly, a feature selection technique chooses the most relevant content from the cells, rows and columns of each dataset, then BERT is used to encode the concatenated text sequence of the selected dataset content as features. The resulting encoding, potentially combined with other features, is then used to derive the similarity between search terms and the dataset.

Instead of matching and calculating relevance scores of keyword-dataset pairs or keyword-annotation pairs, WWT [75] defines a keyword search over datasets as a column mapping task: given multiple-keyword sets, the

[5]https://schema.org

[6]https://www.w3.org/TR/vocab-dcat-2/

Proposal	**Input**	**Subject**	Ranking	**Additional Evidence**	**Auxiliary Data Structure**
Google Dataset Search[16]	Keywords	Body Header Caption	Value Similarity	Ontology	Relationships
OCTOPUS [17]	Keywords	Body	Value Similarity Format Similarity	-	Inverted Indexes
TRSS [100]	Keywords	Body Annotations	Value Similarity	Word Embeddings	Relationships
TSBERT [23]	Keywords	Body	Value Similarity Header Similarity	Word Embeddings	Relationships
WWT [75]	Keywords	Header Body	Value Similarity Header Similarity	-	Relationships
TUS [66]	Body	Body	Value Similarity Domain Similarity	Ontology Word Embeddings	LSH Forest
D3L [12]	Header Body	Header Body	Header Similarity Value Similarity Format Similarity Value Distribution	Word Embeddings	LSH Forest Relationships
QCR [81]	Body	Body	Value Distribution	-	Inverted Indexes
InfoGather [94]	Body	Body Header Caption	Domain Similarity Value Similarity	-	Inverted Indexes Relationships
EntiTables [99]	Body	Body	Value Similarity	Word Embeddings Ontology	Relationships
Aurum [33]	Query	Header Body	Value Similarity		LSH Index Relationships
KGLac [42]	Query	Body Caption	Value Similarity	Word Embeddings Ontology	Relationships

Table 1. Dimensions for dataset search proposals

system treats each keyword set as a column query, and therefore matches each keyword set with columns from different datasets, finally consolidating relevant columns from matching tables into a new dataset. Intuitively, WWT identifies a set of datasets via column search, with both column headers and the context of datasets taken into account to calculate the relevance. As a strategy, by using keywords to provide example values for a target table, WWT can be considered to have some similarity to dataset-driven search, as discussed next.

3.2.2 Dataset-driven Search. Given a dataset, that may have been identified by other search or navigation tasks, techniques that search based on the *Header* and/or *Body* of that dataset can be used to identify additional data of relevance. Such a follow-on search may be to address a specific requirement. For example, Table Union Search

(TUS) [66] aims to retrieve individual datasets that are *union compatible* with a given dataset. TUS considers similarity between the values in different columns, in the domains represented in different columns and in the word embeddings of values from different columns. With less of an explicit focus on unionability, Dataset Discovery in Data Lakes (D3L) [12] seeks to identify collections of one or more datasets that, when joined, are as similar as possible to the given dataset. A further difference between TUS and D3L is that, although TUS considers several types of evidence in identifying related attributes, a probabilistic model is used to select the single most promising type of evidence for each column pair. In contrast, D3L uses a ranking function that considers the Euclidean distance between columns taking into account all the available evidence.

Compared with TUS and D3L, QCR [81] addresses a more specialised dataset search problem, namely join-correlation search: find the top-k datasets that are joinable with the given dataset and contain columns highly correlated with the numerical columns in the given dataset. For example, in Figure 1, an analyst may want to study the potential relationship between average grades in school and deprivation levels. They may find *School3* and *Deprivation*, which are joinable and have correlation in Column *School3.Average Points* and *Deprivation.Income Deprivation Rate* and *Deprivation.Deprivation Rate Quintile*. The proposed solution, QCR, relies on a hashing scheme that builds on sketches that capture correlations between numerical columns. Using this hashing scheme, the join-correlation search task identifies the top-k candidate datasets taking into account both joinability and the presence of correlations.

In essence, TUS, D3L and QCR all work by taking as input a dataset and returning related datasets. There is, however, a related problem of *table augmentation*: given a dataset, search for relevant datasets that can be used to extend the given dataset with additional attributes or rows. InfoGather [94] and EntiTables [99] treat datasets as rows of entities. In Figure 1, *School1* and *School3* both contain data about entities of type *School*, though *School1* primarily contains location attributes and *School3* primarily contains performance attributes. Thus, in principle, each table could be used to augment the other with columns that provide additional information about individual schools. InfoGather [94] identifies three forms of augmentation: *augmentation by attribute name* where the goal is to populate a given attribute based on its name; *augmentation by example* where as well as the attribute name some example values are provided; and *attribute discovery* that identifies potentially relevant attributes. Using schema matching techniques [78], a preprocessing phase creates a weighted, directed graph where nodes represent datasets and edges represent matches between their attributes, and derives properties for this graph that support efficient ranking of datasets at search time. At search time, given a search dataset and the graph, a pagerank variant identifies tables that are candidates for augmenting the search dataset. EntiTables [99] considers two forms of dataset augmentation: the addition of rows and the addition of attributes; both tasks are seen as ranking problems, where the goal is to identify a ranked collection of rows/entities or a ranked collection of attributes, respectively. Then probabilistic models are used to rank candidate entities and attributes. The approach is evaluated using the Wikitables corpus [10], by removing rows or columns from existing tables, and using the approach to identify candidate replacements.

3.2.3 Query-driven Search. The provision of a *Query* based search process involves specifying the *language* to be used for querying and the *model* over which the language is to be evaluated. In Aurum [33], the *model* is a graph that describes the columns in the underlying repository, such that each column is a node, and the edges represent relationships between the columns; the creation of this graph is described in more detail in Section 4. The *language* can then use functions to carry out a keyword search over headers, identify columns that are similar or related to a given column, or identify columns that are directly or indirectly related to a given column. For example, assume that we would like to identify tables that have columns that are similar to those in *School1* in Figure 1 and that mention *London*. This could be captured as:

```
drs = columns("School1")
res = table(match(drs)) AND table(valueSearch("London"))
```

In this example, *drs* is a *discovery result set* that contains the *columns* from *School1*, *table(match(drs))* returns the tables that have columns that are similar to those in *drs*, *table(valueSearch("London"))* returns the tables that contain the value *London*, and *AND* computes the intersection of its operands. Result ranking is an extensibility point in Aurum, but tends to build on the similarity of result values to requests.

Some approaches use structured queries by building their data search systems/frameworks on existing database engines, enabling them to interact with their users and locate relevant datasets via structured queries, as in *SPARQL* [72]. SPARQL allows users to query information from any data source that can be mapped to RDF. It can express queries that range over both data and metadata, as mentioned in Section 3.1, and thus it can express data a variety of discovery and exploration tasks. KGLac [42] creates a knowledge graph to interconnect datasets and columns, capturing metadata and semantics between them. The graph is then represented in RDF, so users can execute SPARQL queries over the graph to search for datasets that meet quite specific criteria.

3.3 Dataset Search Conclusions

State-of-the-art. Dataset search is the process of identifying the data sets that meet a user-specified requirement. As the requirement can be specified in quite different ways, it transpires that dataset search proposals are quite diverse. The input to the search can take the form of keywords, a dataset or a query, giving rise to quite different technical solutions. Furthermore, where the search starts from a dataset d, the result can either be other datasets that are similar to d, other datasets that when joined are similar to d, or other datasets that can be used to augment the data in d in some way. Furthermore technical solutions need to address data heterogeneity and scale both when trying to identify and rank candidate solutions. Thus dataset search is a rich and diverse topic, and there is no one-size-fits-all solution.

Benchmarking and Performance. Evaluations of dataset search techniques investigate both *effectiveness* and *scalability*.

As searches assess the relevance of the datasets in a collection to an *Input* request, the notion of *effectiveness* involves metrics that take into account the ranking of solutions. This can be as direct as the fraction of the searches that contain a relevant result in the top-k (as in [17]) or can also consider the position in the top-k of the relevant results, for example using the Mean Reciprocal Rank, the Normalised Discounted Cumulative Gain (as in [23, 100]) or precision/recall at k (as in [12, 66]). These metrics require an understanding as to the relevance of a result for a query, which can be obtained through crowsdsourcing [17], expert annotation [12, 100] or by generating data sets for experimentation [66]. Evaluations sometimes make use of web tables; for example among keyword search results, TRSS [100] uses tables from the WikiTables corpus [10], and the same datasets and keyword queries are used when evaluating TSBERT [23]. Several of the dataset-driven search evaluations use open government data sets [12, 65, 81], and the benchmark of synthesized data proposed for TUS [66] has been used to compare TUS, D3L and Aurum [12]. There has also been some evaluation relating to usability: Aurum [33] used questionnaires on usefulness and time savings; and Voyager [13], which builds on D3L [12], uses a questionnaire to assess usability and task times to assess productivity.

In relation to *scalability*, keyword search papers tend not to put much emphasis on scalability evaluations, most likely because keyword search at scale is well understood from web indexing. However, dataset-driven search techniques carry out similarity comparisons at the dataset level, which involves maintaining custom indexes for this purpose. Aurum, D3L and TUS have been compared in relation to *indexing time* and *search time* [12] for repositories with up to 12,500 tables and 100,000 attributes. These systems all use LSH indexes and general lessons are: that LSH indexes take less space than the underlying datasets, with D3L requiring the most space because it indexes more features than Aurum or TUS; that index creation times are dominated by tasks other than the inserting of data into the index, such as graph building in Aurum and ontology access in TUS; that the time taken to build the index grows broadly linearly with dataset size; and that search times are highest for TUS,

due to the overhead of accessing an external ontology and because computation of similarity scores requires significant post-processing.

Directions. Dataset search has given rise to techniques that, given a variety of different types of input, can identify datasets from repositories that match syntactic or semantic features of the input. In relation to the *Input*, the most investigated cases are *Keywords* and datasets (some or all of the *Header, Body* or *Caption*). It would be good to have further usability studies to investigate how suitable these are in practice; how straightforward are these to supply, and how often do searches really lead to a useful result? The usefulness of a result to a human user may not be quite the same as relevance used in an empirical evaluation of a method. There are a few evaluations that explore usability aspects data search and exploration (e.g., [13, 60]), but these are more providing summative evaluations on existing approaches than formative evaluations can inform the tasks to be supported. Furthermore, dataset search techniques often use quite limited information to inform the search, either intrinsic (e.g., dataset-driven approaches often only search using the table body) or extrinsic (i.e., searches rarely use much in the way of additional evidence, such as annotations or usage patterns). In addition, most work has focused on tables, without refinements to allow for the fact that searches can be expected to be over heterogeneous data models and formats. There are also likely to be further technical opportunities to exploit, such as making more extensive use of large language models.

4 DATA NAVIGATION

Data navigation is the process of exploring a collection of datasets by following relationships between the datasets. Navigation can usefully follow different categories of relationship, for example to combine datasets using join paths, or to identify similarity relationships that reveal where the same type of data may recur. There are several different types of relationship in Figure 1 that may usefully be used for navigation. For example, a join of *School1* on its *Borough* attribute with *Deprivation* on its *Name* attribute would provide information on the economic environment of the schools. Furthermore, there are several different attributes that describe the *domain* of local government regions, thus capturing a recurring type of data (e.g., *School1.Borough*, *School2.LA_Name* and *BrownfieldSites.Organisation*). When using independently produced datasets, there are also likely to be representational inconsistencies to contend with. For example, *BrownfieldSites.SiteNameAddress* combines site name and street information in a single attribute, whereas street information is stored on its own in *School2.Address*. Furthermore, it is likely to be important to be able to identify and maintain information about relevant relationships at scale.

There is earlier work of relevance to navigation that has likely influenced the results considered here, but that has already been covered by comprehensive surveys. *Data Profiling* [1] is the process of deriving metadata that describes properties of individual datasets and certain relationships within and between datasets. For example, data profiling may discover features such as primary keys, functional dependencies and inclusion dependencies. These can be used, among other things, to identify candidate join paths between collections. *Schema Matching* [78] is the batch process of identifying similar datasets and attributes; schema matching has often been used as evidence to inform the writing or generation of schema mappings for data integration [8], and is related to ontology mapping for knowledge bases [47].Schema matching algorithms can be used to identify potentially equivalent attributes of relevance to dataset discovery and exploration, though in this section we will give priority to techniques that were specifically designed for these tasks. Experimental work has been carried out comparing matching with other data navigation techniques [54].

4.1 Data Navigation Dimensions

As in Section 3, we use dimensions to capture key features of different techniques using consistent terminology; the dimensions are illustrated in Figure 4. Taking them in turn:

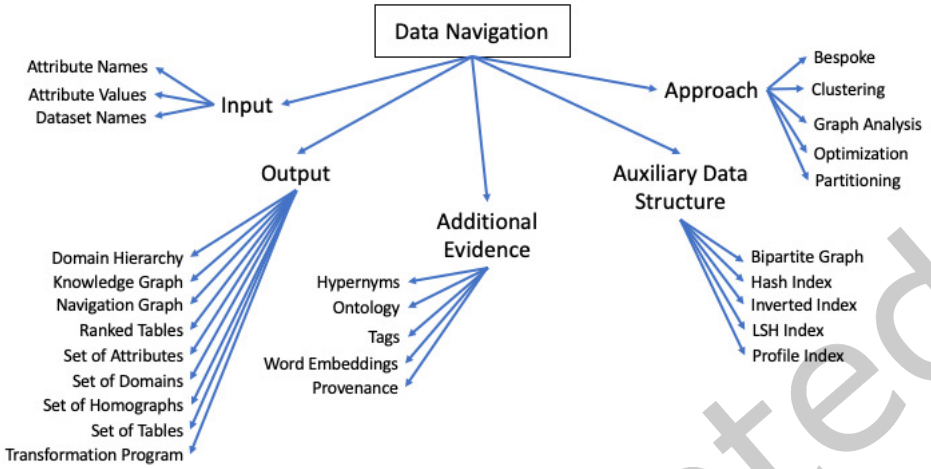

Fig. 4. Data navigation dimensions and their values

Input: The data that is the starting point for the data navigation process. In the scope of this survey, navigation is not typically from one dataset to *similar* datasets, as identification of the datasets that are similar to a given dataset is considered to be a form of Dataset Search (as discussed in Section 3). Navigation between datasets often involves related attributes, and as a result it is common to have as a starting point *Attribute Names* and/or *Attribute Values*, perhaps with some other contextual information such as *Dataset Names*.

Output: The output of a navigation task is typically either a set of related items or a graph that represents the occurrence of a relationship between many data sets. For example, given *School1.Borough* and a request to find other attributes with the same domain, the result could be the set *{School2.LA_Name, School3.LA_Name, Deprivation.Name, BrownfieldSites.Organisation}*. Alternatively, a request could, for example, ask for a graph to be constructed in which the nodes are the datasets and the edges are connections between the datasets that represent pairs of attributes that share the same domain. These approaches provide complementary ways of representing similarity between the attributes of datasets, where the former provides a local perspective, and the latter is more global. The nature of the relationship over which navigation takes place also varies between approaches, and surveyed papers include those that support navigation based on join paths, shared domains and semantic similarity.

Additional Evidence: The information, in addition to that in the Input, that informs the identification of relationships for navigating. Additional evidence is typically used to help identify relationships between attributes that are related but that manifest representational inconsistencies either in their names or values. For example, *Word Embeddings* [3] or an *Ontology* may be used to reflect the semantics of attribute values in addition to or instead of syntactic similarity. A further source of evidence for a relationship between datasets is *Provenance*, which makes explicit which values are derived from others [64].

Auxiliary Data Structures: The construction of the result may require the maintenance of additional data structures to provide access to the evolving solution space. Scale presents a significant challenge to identifying and maintaining relationships, as it is likely to be prohibitively expensive to conduct all-against-all comparisons of attributes in a repository. As a result, auxiliary data structures tend to be indexes;

Table 2. Data Navigation dimensions applied to proposals.

Proposal	Input	Output	Additional Evidence	Auxiliary Data Structures	Approach
JOSIE [105]	Attribute Values	Set of Attributes		Inverted Index	Bespoke
LSH Ensemble [107]	Attribute Values	Set of Attributes		LSH Index	Partitioning
PEXESO [28]	Attribute Values	Set of Tables	Word Embeddings	Inverted Index	Bespoke
Auto-Join [106]	Attibute Values	Transformation Program			Bespoke
Seeping Semantics [34]	Attribute Names Dataset Names	Knowledge Graph	Word Embeddings, Ontology		Bespoke
[65]	Attribute Values	Navigation Graph	Word Embeddings, Tags		Bespoke
C^4 [58]	Attribute Values	Domain Hierarchy			Clustering Optimization
D^4 [69]	Attribute Values	Set of Domains			Bespoke
RAF-STD [73]	Attribute Values	Set of Domains		Linkage Graph	Bespoke
DomainNet [57]	Attribute Values	Set of Homographs		Bipartite Graph	Graph Analysis
FRT [82]	Attribute Names Attribute Values	Ranked Tables	Hypernyms	Hash Index	Bespoke
Juneau [103]	Attribute Values	Ranked Tables	Provenance	Profile Index	Bespoke

for example, an *Inverted Index* [74] containing values from different attributes would group together information about attributes with overlapping values, and thus could be used to accumulate evidence of similarity between the attributes. An *LSH index* [44] could play a similar role, but accommodate some inconsistency in the way that values are represented.

Approach: The technique that is used to identify the relationships to be navigated. Many techniques for identifying navigation opportunities involve custom algorithms, contending with challenges from one or more of heterogeneity or scale.

4.2 Data Navigation Proposals

In this section, we review data navigation techniques, applying the dimensions from Figure 4. Here the subsections are ordered by the nature of the relationship over which navigation can take place.

4.2.1 Join Path Discovery. A join path captures a relationship between datasets that can underpin a join. With reference to Figure 1, even in the absence of declared foreign keys, analyses could establish that *Deprivation.Name* is a key for *Deprivation*, and that the values from *School1.Borough* are included in *Deprivation.Name*, thus inferring a foreign key relationship that could be used for joining. Join paths can be used in different ways during data discovery and exploration.

Although this survey groups results within topics such as *Dataset Search* and *Data Navigation*, individual approaches may have elements that belong to more than one group. For example, from the *Dataset Search* approaches, D3L [12] constructs a graph of join paths behind the scenes, with a view to identifying which collection of joinable tables together are the best match to a query table. Analogously, Aurum [33] constructs an explicit graph of join paths for users to query. There is not always a crisp distinction between categories. However, there are approaches that specifically focus on joins, in particular to contend with the fact that independently produced datasets that may be candidates for joining.

JOSIE [105] supports a version of the joinable table discovery problem: given a table column, identify the set of tables with which that column can most usefully be joined. In JOSIE, given a query consisting of the values in a column c, the aim is to find the *top-k* columns in a collection that have the highest overlap with c. An exhaustive search for the *top-k* columns is impractical for large repositories, and JOSIE provides techniques that dynamically identify subsets of the candidate attributes that are worthy of further consideration. For settings in which there are very large numbers of tables, for example Web tables, LSH Ensemble [107] provides an approximate solution to the same problem. In essence, an index is produced that aims to returns the attributes in a repository with the highest set containment, such that the containment of X in Y is defined as $t(X, Y) = \frac{|X \cap Y|}{|X|}$. As an individual LSH index can give low recall when indexing columns with highly variable cardinalities, LSH Ensemble provides techniques for identifying how to partition data over several LSH indexes to retain good recall and precision.

PEXESO [28] also addresses the joinable table discovery problem, though allowing for the possibility that there may be representational inconsistencies in columns that it may be semantically meaningful to join. For example, in Figure 1, it is semantically meaningful to join *Deprivation.Name* with *BrownfieldSites.Organisation*, but values may not be represented consistently (e.g., see *Camden*). In PEXESO, an offline component constructs an inverted index of candidate join columns, using word embeddings with a view to capturing the meanings of values as multi-dimensional vectors, thereby abstracting over representational inconsistencies. The online component then looks up the column index using the same embeddings; the idea is then that the user can choose which of the joinable columns may be most useful in a given setting.

Representational inconsistencies, in potentially joinable columns may give rise to a requirement for some form of similarity join (e.g., [86]) rather than an equi-join. Alternatively, it may be possible to infer a transformation that can reformat the values in one of the potentially joinable attributes. In Auto-Join [106], similar values in the potentially joinable columns are sampled, and used as training data to infer format transformation programs. These transformation programs can then extract, recombine or reorder tokens that enable equi-joins to be applied to the resulting values.

4.2.2 Related Collection Discovery. In join path discovery, as discussed in Section 4.2.1, relationships are identified between individual data items, that allow navigation from one instance to another. This is clearly useful, but it may also be useful to identify more abstract relationships between collections. When collections are related, there will typically also be relationships between data items within those collections, but the graph of relationships at the collection level is expected to be compact, and thus useful for comprehension.

Seeping Semantics [34], along with Aurum [33] (discussed in Section 3.2), aims to populate a knowledge graph, in which datasets are nodes and edges support both joins (through Aurum) and semantic similarity (through Seeping Semantics). In Seeping Semantics, it is assumed that there is access to an ontology that characterises relevant concepts in a domain. This ontology is then matched with schema information from datasets in the repository; the matching uses a composite matcher that considers both syntactic similarity and word embeddings. Initial syntactic matches are then filtered, to retain only those associated with semantically related terms. Where a schema element from a dataset is matched with several ontology concepts, a pruning step seeks to identify the most specialised generally applicable concept. Where several datasets match the same concept, these datasets are themselves connected in the knowledge graph, indicating semantic relatedness. As such, the knowledge graph

from Seeping Semantics and Aurum provides different types of edge with which to navigate between source datasets.

With a view to making navigation efficient, in the sense that a target dataset can be identified by following a small number of links, a proposal has been made for a graph structure that is specifically designed to support navigation [65]. Navigation takes place through a directed acyclic graph (DAG) in which different nodes represent sets of attributes, and thereby the datasets that contain those attributes. Each node is associated with a subset of the attributes of its parents, and thus with a smaller number of datasets. The user navigates through the DAG from the root until a leaf node is reached. The DAG is constructed to maximise the probability that a table can be found by navigating from the root, visiting a child at each point based on the similarity of the child to the user's requirement. The DAG construction algorithm penalises solutions with long paths. As each graph node represents a set of datasets, the DAG is likely to be much smaller than the Knowledge Graph generated by Seeping Semantics, in which each dataset forms a single node. As such, this approach is an attempt to abstract over the individual sources for the purposes of navigation. Further techniques that seek to abstract over the features of individual sources are discussed under Schema Inference in Section 6.

4.2.3 Domain Discovery. A *domain* is a collection of values that instantiate an application concept. For example, *Camden* and *Greenwich* are both members of the domain of *London Boroughs*, and both *Al Gore* and *Mike Pence* belong to the domain of *Former US Vice-Presidents*. The discovery of a domain, which may occur in many different attributes in a repository, identifies where these attributes are related in the sense that they are representing the same application concept. Note that a single value may belong to multiple domains; for example, both *Al Gore* and *Mike Pence* belong to the domain of *Authors*. Thus in relation to navigation, domain discovery can support users in answering the question *what other attributes contain information that represents the same application concept as this one.*

Motivated by the prevalence of tabular datasets in enterprises, Concept Construction from Coherent Clusters (C^4) [58] creates a hierarchy of domains, where each domain is a set of values drawn from potentially many columns. The approach is first to cluster values based on their co-occurrence in columns, so a pair of values is more likely to be part of a coherent domain if the pair of values appears together in a column (for example, in Figure 1 *Camden* and *Greenwich* appear together in both *School2.LA_Name* and *School3.LA_Name*, which is a signal that they could be part of the same domain). As the clustering process merges smaller clusters to give larger ones, a potentially deep hierarchy of clusters is produced. The hierarchy represents domain containment, which occurs in practice (e.g., with reference to Figure 1, the boroughs listed belong to the domain of *Local Government Areas in London*, which might be considered to be part of the wider domain of *Local Government Areas in England*). As such, there is an optimization step that, given a fixed (probably small) maximum depth of tree, selects a subset of the nodes from the tree of clusters based on how well the clusters are covered by the values in table columns. This optimization objective assumes that domains are likely to be found in their entirety in at least some columns.

Data Driven Domain Discovery (D^4) [69], a bespoke algorithm, explores overlap relationships between sets of column values with a view to identifying which sets of values can be considered to comprise separate domains. A particular focus is placed on coping with the fact that the same value may legitimately belong to more than one domain. For example, in Figure 1, in *School1*, *City of London* is the name of a school and the name of a borough, and both *School Names* and *Borough Names* are domains that it would be useful to identify in our open government scenario. A consequence of the emphasis on overlapping sets of values in the algorithm is that D^4, like C^4, assumes that the values in a domain are represented consistently across different tables.

Also providing a bespoke algorithm that builds domains from attribute extents, RAF-STD [73] has been developed in the context of online product specifications. RAF-STD iteratively extends the values in candidate domains, based on the similarity of the collections of values in different attributes. RAF-STD is not dependent

on values being represented consistently in different sources, but depends on the provision of a reliable linkage graph for the instances from which the domains are being inferred to obtain high recall.

Related to domain discovery, DomainNet [57] identifies homographs in data repositories, where a homograph is a data value that occurs with more than one meaning. In domain discovery based on overlaps between datasets, as in C^4 and D^4, homographs are values that risk the merging of domains that represent different concepts. For example, in Figure 1, the value *City of London* is used in *School1* to represent both the *Name* of a school and the name of a *Borough*.

These approaches to domain discovery are motivated by different applications, and features of these applications influence algorithm designs; it would be interesting to see an empirical comparison of each approach on the others data. Note that, unlike in data annotation (Section 5), none of these approaches give names to the discovered domains.

4.2.4 Multiple Relationship Frameworks. Several proposals have been made that support more than one type of similarity within an overarching framework.

In *Finding Related Tables* (FRT), Das Sharma *et al.* [82] identify related tables T_1 and T_2 as those that are related to a given target table T by queries Q_1 and Q_2, respectively. Then the nature of the relationship between T_1 and T_2 depends on properties of the queries and the relatedness of their results. In practice, the idea is that the queries select or project values from T_1 and T_2 that in turn can be unioned or joined to yield a result that is similar to T. In common with many other techniques, there is significant technical focus on deriving degrees of relatedness. In FRT, this builds on the *subject column* of a table, a column that is inferred to contain information about the entities the table is about [91]. In Figure 1, *Name* is likely to be inferred as the *subject column* of each of the *School* tables. The values in these columns are then used to look up the type of the column in an external hypernym store, such as WebIsA [84]. The similarity between query results is then based on the similarity of the types inferred for column values in the hypernym store. This use of subject columns potentially provides a useful way of abstracting over representational inconsistencies in column values, but also depends on having access to a suitable hypernym store. The approach has been developed in the context of Web tables, and uses a hypernym store inferred from a Web crawl.

A still wider range of relationships is explored in Juneau [103], which is designed to support data scientists in identifying related tables for data science tasks. Tasks identified include augmenting training data, linking tables that have high column overlap, identifying additional features for machine learning, and data cleaning. Each of these tasks require different emphases when looking for datasets that are similar to a given dataset. As a result, Juneau provides a large number of different similarity metrics, many covering similar ground to that covered earlier in this section on navigation, but also including similarity of provenance, informed by *how* datasets were produced by a Python program. The contribution of this work has been to combine these techniques in a single relationship framework to provide targeted support for data science tasks [102].

4.3 Data Navigation Conclusions

State-of-the-art. Obtaining value from data repositories often depends on combining datasets to obtain a more complete picture than is provided by datasets in isolation. This in turn involves navigating between datasets, to identify additional data that can be combined through joins, or to identify datasets that contain similar types of data. Specific results have to deal with challenges that result from scale (large numbers of datasets, or large datasets) or heterogeneity (different names in metadata, data values being formatted in different ways) or both. As a result, results are often approximate, building on models of similarity that are designed to capture an intuition as to what is likely to be suitable in a given setting [103].

Benchmarking and Performance. Evaluations of data navigation techniques investigate both the *effectiveness* and *scalability*.

In relation to *effectiveness*, although it is not unknown for techniques to provide exact solutions to a problem [105], most problem specifications give rise to approximate or best-effort solutions, to accommodate data inconsistencies [28, 34, 73, 106] or scale [107]. This is reflected in the data sets used for evaluation, where data sets can usefully be large and produced by multiple publishers; many techniques are evaluated using open government data (e.g., [69]) or web tables (e.g., [106]) or both (e.g., [28, 105, 107]). However, although there is some shared practice in the data sets used, variations in the problems addressed tend to mean that individual approaches develop their own test cases. For example, in join discovery, some techniques focus on overlapping attribute values and thus require consistent representations in joinable columns (e.g., Josie [105] and LSH-Ensemble [107]), while others focus on cases where values have inconsistent representations (e.g., Auto-Join [106]). It is a similar story in domain discovery, where some techniques build on value overlap (e.g., C^4 [58] and D^4 [69]) whereas others build on value similarity (e.g., RAF-STD [73]). In such settings, many experiments develop Ground Truths that reflect the task at hand; this may have been obtained by comparing an approximate algorithm with the exact result [107], or by manual annotation (e.g., [28, 69, 73]). For example, manual annotation is commonly used in domain discovery as only an expert user can determine what values constitute a domain, especially because domains can overlap. There are also cases where the suitability of a result is subjective, giving rise to user studies in which the results are assessed by individuals, for example in Seeping Semantics that postulates relationships between collections [33] and when navigating within data collections for discovery purposes [65].

In relation to *performance*, many of the techniques in data navigation involve the comparison of collections of collections of values, for example exploring the collection of attributes, each of which is a collection of values, to identify which attributes may join or which attributes draw values from the same domain. Such comparisons can be supported by indexes, whether exact inverted indexes (e.g., as in Josie [105]) or approximate indexes (e.g., as in LSH-Ensemble [107]). Experiments on scalability then report runtimes for different dataset sizes. The nature of the evaluation varies from problem to problem. For example, in join discovery, scalability results have been reported for the following problems: identify the columns that may join with a given column [105]; or identify the cost of evaluating a join involving syntactically inconsistent columns [106]. Thus although there has been some consistency of practice in dataset selection, problem variants can give rise to very different scalability challenges. As a result, there are fewer head-to-head performance studies than might be expected, though LSH-Ensemble and Josie are compared in [105].

Directions. Although existing techniques can identify a variety of navigation opportunities at scale, there is quite some diversity in these. This is reflected in the fact that, as well as the individual types of navigation, there are papers that describe several types of navigation in a single environment. It would be good to have more insight on how these can be used in practice in different settings, such as notebooks [13, 103], data lakes [70] or open data systems [60]. Such experience may cast light on the recurring dichotomy between techniques that assume consistent value representations when inferring relationships (that support joins, for example) and those that allow for inconsistent value representations (but that are then more difficult to compute over). Much of the work on navigation could also provide a foundation for the provision of more abstract representations of the data in repositories, as discussed further in Section 6.

5 DATA ANNOTATION

Data Annotation is to associate data elements with some pre-defined annotations from vocabularies and knowledge graphs (KGs)[7], so as to enriching the data meta information and semantics. Considering the tables in Figure 1, we

[7]The definition of KG often varies from literature to literature. For simplicity, we regard that KG includes relational facts, ontology, semantic network, catalog, etc.

could annotate that the *Phase* column of *School2* is composed of categorical values whereas the *Pupils* column of *School3* contains integers. Such annotations bring data type information. Given a KG such as Wikidata[8], we could further annotate the cell *Camden* in *Deprivation* with the entity *WD:Q149836*, and annotate the column *Name* with the semantic type *WD:Q149621 (district)*[9]. Different kinds of semantics have already been defined for these annotations in Wikidata, allowing us to capture a broader and more accurate understanding of the tables (e.g., "Camden" does not refer to the city Camden in New Jersey in USA, but an area of London in UK). In many semi-structured datasets in formats such as TSV and CSV, there is a shortage of meta information such as the file name and column header, or the meta information has no concrete meanings (e.g., the column header *Name*). In such cases, both data type annotations and semantic annotations become especially important to curate and fully understand the data.

There have been quite a few studies on data annotation, especially for tabular data, mainly from the Database, the Semantic Web, Natural Language Processing and Machine Learning communities. Table annotation is sometimes called table interpretation, while table annotation with knowledge base constructs is also known as *semantic table interpretation*, *semantic table annotation*, *table to KG matching* and so on [18, 45, 101]. The specific task of annotating table cells by KG entities is also equivalent to *table entity linking* [10]. There is some literature that partially reviews table annotation studies from specific perspectives. Pujara *et al.* [77] summarizes a tutorial in KDD'21 on recent advances in table understanding, where some semantic table annotation works on table cell typing and entity linking are reviewed. Cafarella *et al.* [18] briefly reviews Web table processing and application in the past ten years, covering some works on subject column discovery, column type annotation and inter-column relationship annotation. Zhang *et al.* [101] focuses a comprehensive literature review on processing Web tables, which includes a part on table interpretation. Different from these review works, in this section we aim to give a more comprehensive overall picture on data annotation, especially different kinds of annotations towards tabular data.

5.1 Data Annotation Dimensions

Input: Data annotation mainly focuses on semi-structured data, especially tabular datasets without high quality meta information, such as Web tables, CSV files, TSV files and spreadsheets. For simplicity, we refer to such tabular data and relational databases as *tables*. Besides tables, semi-structured data for annotations also include *graphs*, and tables can sometimes be transformed into graphs using tools such as GraphDB OntoRefine and BootOX [35, 46]. The annotations can be a pre-defined *vocabulary*, which includes *ad hoc* labels defined for a task (e.g., "Primary School" and "Secondary School" for annotating a column's semantic type), and widely recognized terms (e.g., *xsd:integer* for annotating a column's data type). The annotations can also be data-level and schema-level semantics defined in a *KG* such as instances (a.k.a. entities), semantic types (a.k.a. classes or concepts), data and object properties, etc.

Output (Annotations) and Annotation Subject: Data annotation outputs different annotations for data elements of different subjects. A *table cell* or *graph node* can be annotated by an entity if it is an entity mention, or a data type if it is a literal such as a number and a date; a *table column* can be annotated by a semantic type according to its cells. Specially, one whole *table row* is sometimes annotated by an entity and one whole *table* is sometimes annotated by a class. In this case, the table usually has a subject column, whose cells are subject entities of their corresponding rows. One example is the *Name* column in *School2* in Figure 1. Note that one column can be annotated by multiple classes, and they are usually hierarchical with a subsumption relationship (e.g., *School* and *Organization*). The *relationship* between two columns can be annotated by hierarchical object properties (i.e., relations) if both columns contain entities, and hierarchical

[8] https://www.wikidata.org/

[9] *WD:* is short of the IRI prefix https://www.wikidata.org/wiki/.

data properties if one column contains entities while another column contains literals as e.g., the entities' attribute values [45, 59]. The annotation for the relationship between two graph nodes is similar.

Additional Evidence: Quite a few additional resources could be utilized to support data annotation. In table annotation, one common kind of resource is the *table context*, such as the table caption, and the surrounding text from Web pages or articles [104]. Similarly, *table headers* (table schema), i.e., the column names, sometimes could also be utilized. Besides the additional evidence from the data to annotate, vocabularies and KGs are sometimes augmented by some efficient knowledge retrieval tools, especially lookup services which enable efficient keyword-based entity search based on a lexical index (e.g., DBpedia Lookup[10]), and the query engine which enables complex knowledge access via logical queries (e.g., Wikidata Query Service[11]). Both lookup services and query engines are especially important when the KG or vocabulary for annotation has a huge size, as they can help to efficiently extract a candidate annotation set with a reasonable size and a high recall [22, 30, 68]. Meanwhile, some external resources such as search engines and synonym dictionaries could also be used as additional evidence for matching [104]. Nowadays, with the application of machine learning and Natural Language Processing (NLP) methods applied in data annotation, some pre-trained word embeddings and models such as Word2Vec and BERT have also been widely used, especially for text understanding and inference.

Approach: The key challenge in data annotation is capturing the contextual meaning of the target data element and the disambiguation of the candidate annotations. We divide these widely used techniques into four categories: (i) *Lexical Matching*, which compares the surface forms of a data element and an annotation for equivalence matching, usually with the help of lexical similarity metrics (e.g., edit distance and cosine similarity) or lexical indexes; (ii) *Joint Optimization*, which jointly or collectively annotates multiple related data elements with their correlation and consistency considered for disambiguation (e.g., the type of the entity annotation to a cell should be consistent with the type annotation of this cell's column); (iii) *Neural Networks & Embedding*, which first represents the data elements and annotations in a vector space with their semantics such as text and contexts conserved, often using neural networks (NNs), pre-trained embeddings and models, and then calculates the vector distance or predicts the matching with a machine learning classifier; (iv) *Crowdsourcing*, which aims at developing platforms and tools for easier manual annotation with human labour reduced. It is worth mentioning that many successful table annotation systems or methods often combine technologies of more than one category. One typical paradigm is initially matching data elements with annotations such as KG entities and classes by lexical matching with different string similarity metrics, and then adjusting the matching degrees via joint optimization which models the influence between annotations or maximizes an overall score using iterative algorithms or probabilistic models [59, 79, 104].

5.2 Data Annotation Proposals

Here we review data annotation proposals, grouped by the nature of the annotation produced.

5.2.1 Entity Annotation. In textual data, there are often words or phrases that describe entities, often known as entity mentions in NLP. For tables and graphs, such entity mentions may appear at the level of a whole table cell, a phrase/word inside a table cell, a graph node, or even a whole table row with one cell as the subject and the remaining cells as its attributes. A straightforward solution for annotating such entity mentions is directly comparing their surface forms with annotations via string similarity metrics such as TF-IDF, cosine similarity and BM25. Some strategies, such as ensembling scores from different metrics, matching by some tokens of the entity mention, looking for different forms or synonyms of words from external resources, could be adopted to

[10]https://lookup.dbpedia.org/
[11]https://query.wikidata.org/

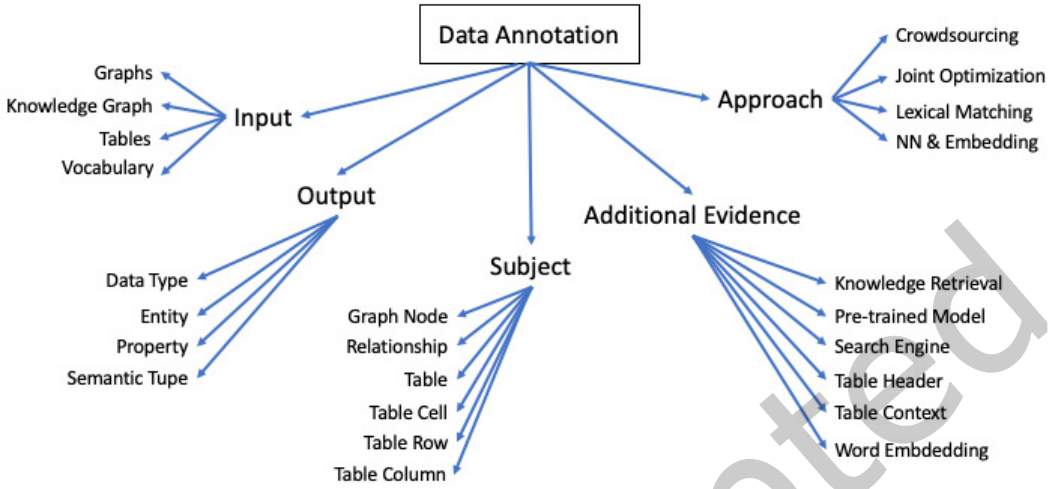

Fig. 5. Data Annotation dimensions and their values

improve the performance. However, it still suffers from the scalability issue, and in many cases cannot capture the semantics for disambiguation, especially when a large scale KG is considered for annotation candidates. For the scalability issue, successful solutions, such as Efthymiou *et al.* [30] and MTab [68], adopt modern KG retrieval tools such as lookup services based on lexical indexes.

To capture the semantics for disambiguation, the contexts of both mentions to annotate and entity annotations should be considered and utilized. Efthymiou *et al.* [30] proposes to use semantic embedding for better performance, besides KG lookup. It includes two stages: an *off-line* stage, which calculates the embedding of each KG entity via a random walk of its neighbourhood and Word2Vec for its textual information; and an *on-line* stage, which first constructs a disambiguation graph whose vertices are candidate entities of each mention accessed by lexical matching and edges are normalized cosine similarities between candidate entities, and then applies a weighted PageRank algorithm to compute the scores of candidates of each mention. TURL [26] embeds a table for different semantic table annotation tasks. It pre-trains a structure-aware Transformer with a Masked Entity Recovery objective, and fine-tunes this model with labeled samples when applied to entity annotation. Instead of pre-training a new Transformer model from scratch, BERTMap [40] prefers to fine-tune a pre-trained BERT model. Note that although BERTMap is originally developed for matching two ontologies, it can be applied to annotate graph nodes with KG entities.

Joint optimization is also widely used for utilizing the correlation between annotations for disambiguation. TabEL [10] adopts the Iterative Classification Algorithm to collectively disambiguate all mentions in a given table, using different kinds of features, such as prior probability features, semantic relatedness features, entity-mention similarity features and so on. The on-line stage of Efthymiou et al. [30] also belongs to joint optimization, where entity candidates of each mention are considered for disambiguation. Limaye et al. [59], TableMiner+ [104] and T2K Match [79] consider the correlation of not only the entity annotations, but also other kinds of annotations. The first method uses a probabilistic graph model to optimize an overall score that takes all the annotations of

Table 3. Data annotation dimensions applied to proposals.

Proposal	Input	Output (Annotation)	Annotation Subject	Additional Evidence	Approach
[30]	Tables KG	Entity	Table Row	Knowledge Retrieval Word Embedding	Lexical Matching NN & Embedding Joint Optimization
MTab [68]	Tables KG	Entity Semantic Type Data Type Property	Table Cell Table Column Relationship	Pre-trained Model Knowledge Retrieval	Lexical Matching
TURL [26]	Tables KG	Entity Semantic Type Property	Table Cell Table Column Relationship	Table Context Table Header	NN & Embedding
BERTMap [40]	Graphs KG	Entity	Graph Node	Pre-trained Model	Lexical Matching NN & Embedding
TabEL [10]	Tables KG	Entity	Table Cell	—	Lexical Matching Joint Optimization
[59]	Tables KG	Entity Semantic Type Property	Table Cell Table Column Relationship	Table Context Table Header	Lexical Matching Joint Optimization
TableMiner+ [104]	Tables KG	Entity Semantic Type Property Data Type	Table Cell Table Column Relationship	Table Context Table Header Search Engine	Lexical Matching Joint Optimization
T2K Match [79]	Tables KG	Entity Semantic Type Property	Table Row Table Relationship	Table Header	Lexical Matching Joint Optimization
OpenRefine [92]	Tables KG	Entity	Table Cell	—	Crowdsourcing
ColNet [22]	Tables KG	Semantic Type	Table Column	Knowledge Retrieval Word Embedding	Lexical Matching NN & Embedding
SDType [71]	Graphs KG	Semantic Type	Graph Node	—	Joint Optimization
[67]	Tables KG	Data Type	Table Column	Table Header Knowledge Retrieval	Lexical Matching
TTLA [4]	Tables Vocabulary	Data Type	Table Column	—	Joint Optimization

a table into consideration, while the following two methods adopt iterative algorithms to let the annotations impact each other. One simple iteration idea often adopted by such methods is first determining a column's semantic type via its cells' partial entity annotations and then adjusting the cells' entity annotations under the constraint of the column type.

It is worth mentioning that table or graph entity annotation has a significant overlap with entity linking in NLP, which has been widely investigated [85]. Some of the entity linking methods could also be applied to table or graph entity annotation after extension. Meanwhile, developing a user friendly system for easier human

annotation and crowdsourcing is also a feasibility solution. OpenRefine [92] is such a table annotation system that supports entity annotations from Wikidata.

5.2.2 Semantic Type Annotation. Like entity annotations, semantic type annotations can be directly inferred from the entities. In semantic column type annotation, a column is usually annotated with the common types of the entity annotations of its cells. For example, MTab [68] first gets cells' high quality entity annotations via data preprocessing (e.g., data type prediction, header prediction and subject column prediction) and different matchings with WikiGraph — a target KG built from DBpedia, Wikipedia and Wikidata — and then infers semantic column types. However, this solution will fail if the entities are challenging to annotate, or some or even all of the entity mentions in a column do not have entity associations in the KG, or the semantic types for annotation do not have enough instances [22]. In joint optimization methods with different kinds of annotations considered, such as Limaye et al. [59], TableMiner+ [104] and T2K Match [79], column type annotations and entity annotations are improved together, usually in an iterative way.

Another more straightforward solution is directly predicting the semantic column types without entity linking. ColNet [22] first gets relevant KG entities and semantic types via lexical matching by table cells, and then trains a one-vs-rest classifier for each candidate semantic type using synthetic columns generated from relevant KG entities. Word embeddings and a Convolutional Neural Network are adopted by the classifier for embedding the column semantics for disambiguation. TURL [26], which embeds a table via pre-training a structure-aware Transformer, can also be applied to predict semantic column types via fine-tuning with labeled samples.

Semantic type annotation for graph nodes has been investigated for years in domains such as KG construction and refinement. For example, SDType [71] calculates the statistical distribution of the types in the subject and object positions of each property, and then simply infers the type of an entity according to its associated properties with a weighted voting approach. Modern KG embedding techniques have also been applied to type KG entities, where the types are embedded with their relationships concerned (e.g., [97]).

5.2.3 Property Annotation. As with semantic type annotation, properties can be annotated by utilizing the annotated entities. Given a pair of entities e_1 and e_2, a straightforward solution, is to match e_1 with all the associated entities of e_2 in the KG, and adopt the properties that associate the matched entity with e_2. Similarly, e_2 can also be matched with the associated entities of e_1. To annotate the relationship between two table columns, simple majority voting or weighted voting (as in e.g., T2K Match [79]) can be adopted based on all entity pairs from table cells. Some systems, such as MTab [68] and TableMiner+ [104] also detect the subject column for assistance, and utilize the names (headers) of surrounding columns as indicators. As in semantic column type annotation, table properties are also jointly annotated with other kinds of annotations for disambiguation in systems such as T2K Match [79], Limaye *et al.* [59] and TableMiner+ [104]. TURL [26] also fine-tunes its pre-trained structure-aware table representation model to directly predict property annotations with the table contexts embedded for disambiguation. For graph data, the relationship between two nodes can also be annotated by relations (object properties), which is sometimes known as link prediction. There have been quite a few methods using graph algorithms and deep learning techniques, including random walk, graph kernel, graph feature engineering, graph embedding, graph neural networks and so on [19, 63].

5.2.4 Data Type Annotation. Data types often need to be recognized before annotating entities, semantic types and properties, and some semantic table annotation systems firstly annotate data types for cells and columns. For example, MTab [68] first predicts a table cell's type as empty, named entity or literal type (such as numerical tags, email, URL and phone number), and then determines a column's data type via majority voting with its cells. Similarly, TableMiner+ [104] addresses data type annotation as a preprocessing step. It uses regular expressions over each cell to classify them into empty, named entity, number, date expression, long text or other.

Some methods, in contrast, are developed specifically for data type annotation for table columns. Neumaier *et al.* [67] constructs contextual data types using data properties and the subjects of data properties, such as *height of people* and *height of building*, represent these types as a tree, and develop a nearest neighbours classification model to predict the type of a numeric table column. Alobaid *et al.* [4] defines a number typology (vocabulary), which includes types such as nominal, ordinal and interval-ratio and sub-types such as categorical, counts and sequential, for annotating numerical table columns. Their method, TTLA, uses complex type-wise heuristic rules and a type detection order to detect the type and sub-type of a column from a set of values.

5.3 Data Annotation Conclusions

State-of-the-art. Data annotation, which associates data elements of tables or graphs with pre-defined (and often shared) semantic labels, can enhance data understanding and has started to play a fundamental role for supporting other dataset discovery and exploration tasks including dataset search, data navigation and schema inference. Different kinds of annotations, including data types and KG defined entities, semantic types and properties, have been supported by the current methods for data elements in tables and graphs. Some typical technologies and paradigms together with external resources for annotation, including lexical matching with KG index and retrieval service, joint optimization of multiple annotations for disambiguation, and table semantic embedding, have been well investigated in the existing studies.

Benchmarking and Performance. For semantic table annotation, there are several widely used benchmarks with different ground truth annotations. One typical benchmark is the table set extracted from Wikipedia pages, which was originally proposed in [59] and recently released by [22, 30]. Wikipedia tables often have high quality and their cells' hyperlinks to other Wikipedia pages can be utilized to get the ground truth entity annotations via the correspondence between Wikipedia pages and entities of KGs such as Wikidata. Another typical benchmark is called T2Dv2 which was extracted from regular pages of the Web with manual entity annotations from DBpedia [79]. Note that with the entity annotations to the cells, the columns' type annotations and inter-column property annotations can be inferred via semantics in the KG. These two benchmarks both have a medium scale with hundreds of tables. Recently, the yearly SemTab challenge[12] on matching tabular data to KG has developed quite a few real-life and synthetic benchmarks with varying sizes and tasks of cell entity annotation, (hierarchical) column type annotation and inter-column property annotation, covering KGs of DBpedia, Wikidata and Schema.org [45].

Regarding the evaluation, most of the current data annotation works focus on the annotation quality with metrics such as Precision, Recall and F1 Score. Although some works report the computation, efficiency and scalability are much less considered in the current studies. SemTab also focuses on the accuracy evaluation. It attracts around 10 systems for evaluation in each year, covering all the approaches listed in Table 3. According to SemTab's evaluation reports and studies of the methods reviewed in this paper, those systems that ensemble different lexical matching methods, such as MTab [68], can often easily achieve quite good performance. Neural network and embedding-based approaches have shown strong capability in capturing the semantics and often achieve better performance on matching cases with different surface forms, but they often need to be carefully combined with some lexical matching methods or KG indexes for higher overall performance and higher scalability.

Directions. Many techniques have been proposed and evaluated for data annotation, especially semantic table annotation, but there are likely to be significant remaining opportunities for annotating data repositories. One problem, for example, is that the current studies often regard annotation as an isolated task, ignoring the annotation's downstream tasks and resulting requirements. Meanwhile, state-of-the-art machine learning techniques, such as pre-trained (large) language models (e.g., BERT [27] and ChatGPT[13]) in combination with

[12]https://www.cs.ox.ac.uk/isg/challenges/sem-tab/
[13]https://openai.com/blog/chatgpt

School							
Name	Postcode	Address	Type	Town	Gender	Phase	Borough
St Mary's	NW3 6PG		Independent	London	Mixed		Camden
Argyle	WC1H 9EG	Tonbridge Street		London		Primary	Camden

Fig. 6. Example of an inferred *School* table with example rows from *School1* and *School2* in Figure 1.

prompt learning, which have significantly improved many natural language understanding and inference tasks, have not been widely explored for data annotation. Data annotation tools that can better utilize such techniques are expected to have better performance, especially for data with semi-structured and unstructured text.

6 SCHEMA INFERENCE

Schema inference is the process of identifying datasets that share structural elements, and thereby of generating a global schema that can represent the data captured by multiple datasets. Consider the example from Figure 1. The figure contains *5* tables, but two of them (*School1* and *School2*) contain information about school locations. Thus a schema inference process could infer a single table definition for inclusion in a global schema that includes the attributes from both *School1* and *School2*, merging pairs of attributes that are inferred to be equivalent, as illustrated in Figure 6. At scale, schema inference can generate a summary of the structural properties in a repository that is much more concise than the original list of datasets. In addition, schema inference can identify recurring relationships, and in some cases can infer inheritance hierarchies.

As a result, schema inference can be seen as complementing the other areas covered in this survey. By default, a Dataset Search is likely to return a ranked list of similar datasets; schema inference over such datasets can identify recurring patterns and make explicit relationships between the search results. Dataset Navigation typically identifies relationships between individual datasets; however, schema inference seeks to generalise from relationships between individual datasets to relationships between groups of datasets. Dataset Annotation can identify conceptually related datasets by associating them with related terms from a vocabulary; schema inference can potentially identify conceptually similar datasets in the absence of a relevant vocabulary.

There is a substantial literature on schema inference, with most results focusing on specific data models. Different flavours of schema inference may be referred to as schema extraction [41], schema discovery [50], structure inference [25], graph summarization [37] and schema induction [93]. An inferred schema may have a precise relationship to the underlying individuals; for example, when inferring an XML Schema from a collection of XML documents, the documents may all conform to the inferred schema. This has some obvious advantages in terms of the role and use of the inferred schema, but if the underlying documents are highly heterogeneous, the inferred schema may become cumbersome, or even a union of the underlying representations. As such, in other approaches, the inferred schema may not attempt to have such a crisp relationship to the underlying data, and may more seek to produce a summary that captures recurring features. In this paper, as the scope is discovery and exploration over heterogeneous and independently produced datasets, our focus is primarily on schema inference for summarising the structural features of datasets.

6.1 Schema Inference Dimensions

As in previous sections, we use a collection of dimensions to capture key features of different approaches using consistent terminology. The dimensions and their values are listed in Figure 7. Taking them in turn:

Input: The data that is the starting point for the schema inference process. This is usually data described using a single data representation that includes instances, such as the tuples in a table, an XML document or a JSON dataset. The *Input* may include some type information, e.g., an RDF document may include annotations on the types of resources and their properties.

Output: The result of the schema inference process is a schema described using some notation. The Output could use an existing representation, such as an E/R Diagram or an XML Schema, or a representation that has been developed specifically to capture the result of the inference process. The Output may or may not use a notation that is specifically designed for use with the Input. For example, if the input is an XML document, the output could be an XML Schema or an E/R diagram; the nature of the output tends to be influenced by the *Purpose* of schema inference, a further dimension that is described below.

Additional Evidence: The information, contained within or in addition to the *Input*, that informs the schema inference process. For example, when inferring a type from two existing datasets, the similarity of the *Attribute Names* and *Attribute Types* of the two datasets are likely to be useful for establishing how similar they are and which of their attributes may be able to be merged in the inferred schema. Furthermore, when inferring relationships between inferred types, it may well be useful to consider *Relationship Types* and *Relationship Values* between the datasets in the repository for which a schema is being inferred.

Approach: The technique that is used to carry out schema inference. There are often several facets of a schema inference process, for example inference of types and relationships may be in distinct phases, so summarising them in a few words is likely to lose relevant information, but several techniques recur. For example, *Clustering* on similarity of *Attribute Names* and *Relationship Names* is a popular approach to identifying the entity types in datasets. Fuller details of the approaches followed by the surveyed techniques are provided in the text.

Purpose: The reason why schema inference is taking place; the role to which the *Output* will be put. The purpose informs the level of detail that needs to be captured. Schema inference may provide enough information to allow the user to *Query* the available data; in this case all attributes in the underlying datasets need to be present in the inferred schema. As retaining all attributes from sources can lead to unwieldy inferred schemas, a less detailed model may be more suitable to *Document* the available data; such an approach may, for example, drop attributes that appear only rarely in sources, thereby abstracting over some specifics. Approaches that aim to *Document* the schemas of a collection of data sources may generate a conceptual schema with a visual representation, such as an E/R model. A further possibility is that the goal is to give a concise insight into the most commonly available data, in which case the role of the approach is to *Summarize* the available data.

6.2 Schema Inference Proposals

This section describes specific schema inference proposals, applying the dimensions from Section 6.1 to provide consistent terminology and to allow comparison. Schema inference techniques can be grouped in different ways; we will group papers primarily based on the *Purpose* of the schema inference, with an emphasis in the discussion on the applicability of these techniques for discovery and exploration. As there is a substantial literature on schema inference, we discuss representative examples, rather than seeking to provide exhaustive coverage. We aim to include examples that are representative of prior work by including approaches that: (i) support different purposes, thus representing the different roles that schema inference can play; and (ii) act over different data models, thus representing different communities that have worked on schema inference.

Existing surveys provide more detailed coverage by model, specifically in relation to XML [53], graph summarization [20] and the Semantic Web [49]. While these surveys provide focused material on schema inference for specific models, here we aim to complement them by revisiting schema inference as one of a collection of approaches to dataset discovery and exploration, where it can fulfil a variety of purposes over different data models.

6.2.1 Inferring Schemas for Querying. Before we move on to our main focus on techniques that support the documenting and summarizing of data, we consider some examples where the aim is to infer a schema that

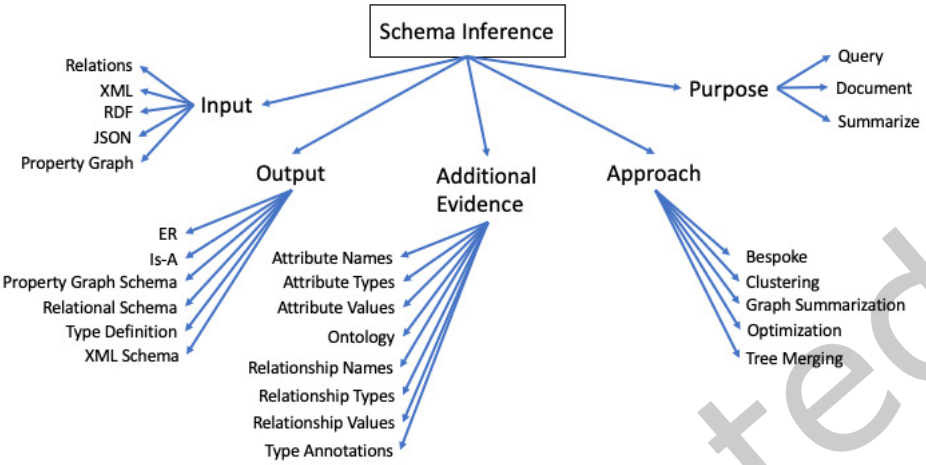

Fig. 7. Schema inference dimensions and their values

precisely describes the underlying data, for a specific data model. Baazizi *et al.*, propose a parametric schema inference technique for JSON [6]. The approach is *parametric* in the sense that different approaches can be used to combine type definitions obtained from different individuals or collections. Given a collection of JSON datasets, the approach infers a type for each JSON record, and then merges these types to produce a type that describes all the available data. The proposed approach is always *sound*, in the sense that the inferred type is always a correct type for the given collection. However, there are different ways of generating *sound* types, and different parameters trade off the preciseness of the result with the size of the result. For example, assume that + is the operation for fusing types, ? represents an optional field, and a record is represented as a set of $name : type$ pairs. Consider the following type fusion:

$\{name : Str, postcode : Str\} + \{name : Str, address : Str\}$

A possible result of the fusion is:

$\{name : Str, postcode : Str?, address : Str?\}$

Any valid input of type $\{name : Str, postcode : Str\}$ or $\{name : Str, address : Str\}$ can be represented as $\{name : Str, postcode : Str?, address : Str?\}$. However, for example, in the original fields *postcode* and *address* never appear together in a record, whereas in the fused type they can. So, in this case, the inferred type is concise, but not especially precise, as information has been lost. The trade-off between concise and precise representations can be controlled by parameterising the approach with different fusion strategies.

Note that although this approach is systematic, and guarantees soundness, it is not designed to resolve representational inconsistencies. As a result, if a name is represented by $\{firstname : Str, lastname : Str\}$ and by $\{forename : Str, surname : Str\}$, no attempt will be made to resolve the representational inconsistency, and both representations will be present in the inferred schema.

The approach of inferring schema definitions that precisely and concisely describe collections of inputs has also been investigated for XML [53]. For example, a bespoke algorithm has been developed that, given a collection of XML documents, infers an XML Schema with which the documents are consistent [9]. Given certain constraints on the underlying documents, a regular expression is inferred in which each XML element name occurs only once,

Table 4. Schema inference dimensions applied to proposals.

Proposal	Input	Output	Evidence	Approach	Purpose
[6]	JSON	Type Description	Attribute Names Attribute Types	Tree-Merging	Query
[9]	XML	XML Schema	Attribute Names Relationship Names	Bespoke	Query
[25]	RDF	E/R	Relationship Names	Clustering	Document
GMMSchema [14]	Property Graph	Property Graph Schema	Relationship Names Type-Annotations	Clustering	Document
[50]	RDF	E/R, Is-A	Relationship Names	Clustering	Document
HInT [48]	RDF	E/R	Relationship Names Type Annotations	Bespoke	Document
[37]	RDF	E/R	Relationship Names Relationship Types Ontology	Graph Summarization	Document
[89]	RDF	Type Description	Relationship Names Relationship Types	Clustering	Query
HIEDS [24]	RDF	E/R, Is-A	Relationship Names Type Annotations Relationship Values	Optimization	Document
[98]	Relations XML	XML Schema	Relationship Names Relationship Values	Bespoke	Summarize
[96]	Relations	Relational Schema	Attribute Values Relationship Values	Clustering	Summarize
[95]	RDF	Relational Schema	Type Annotations Relationship Names Relationship Types	Optimization	Summarize

and this expression is simplified to reduce the number of XML type definitions used. This approach identifies and seeks to address several challenges with the inference of a precise schema definition: the data over which inference is taking place may not represent all the legitimate cases, the training data contains only positive cases, and rich schema languages may present fundamental barriers to inference.

Approaches that generate schemas to which all instances must conform provide useful ways of inferring missing schema information for individual modelling languages, but for practical purposes are designed to work in settings where the available documents are largely consistent. For example, in XML, the relationship between pupils and the schools they attend could be modelled with *school* as a parent element of *pupil* (*/school/pupil*) or the other way around (*/pupil/school*). If both cases are represented in the available XML documents, then schema inference is likely to include both cases in the inferred schema, even though they both represent the same application concepts.

6.2.2 Inferring Schemas to Document a Collection. Due to our focus on data discovery and exploration, we subsequently focus on papers that seek to infer schemas that look to provide useful information about the underlying data, without seeking to infer a schema that directly captures the structure of all the existing data.

RDF is a graph-based data model that is central to the Semantic Web, as well as being used as the model for several graph data management systems. In both settings, RDF graphs can be created that share terminology, but the resulting graphs are not constrained to support only certain properties. Furthermore, the foundational role of

RDF in Linked Open Data [11] means that RDF is expected to be used in settings where numerous independent publishers are active, thus inevitably leading to inconsistent representations.

Approaches that cluster over an RDF graph can identify recurring patterns within the graph. Although nodes in an RDF graph may be annotated with types, independent publishers may use different types to describe the same real world concept. As a result, the annotated type is only one source of evidence, and the notion of similarity used by clustering algorithms can consider different kinds of evidence. Christodoulou *et al.* [25] uses hierarchical clustering to group individual resources into entity types. The similarity of two resources is represented by the Jaccard similarity of their sets of relationship names, and the similarity of two clusters is represented by the average similarities of their members[14]. In this setting, bottom-up hierarchical clustering is used to merge pairs of clusters, starting with unary clusters and ending up with a single cluster that represents the type of which all resources are members. For example, with reference to Figure 1, given that the Jaccard similarity between two sets A and B is: $Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$, the Jaccard similarity between the attributes of *School1* and *School2* (assuming that *Borough* and *LA_Name* are equated) is:

$$Jaccard(Attributes(School1), Attributes(School2)) = \frac{|Name,Postcode,Town,Borough|}{|Name,Postcode,Type,Town,Gender,Borough,Address,Phase|} = \frac{4}{8} = 0.5.$$

In contrast, Jaccard similarity between the attributes of *School1* and *School3* (assuming that *Borough* and *LA_Name* are equated) is:

$$Jaccard(Attributes(School1), Attributes(School3)) = \frac{|Name,Borough|}{|Name,Postcode,Type,Town,Gender,Borough,Pupils,A*-C,AtLeastOne,AveragePoints|} = \frac{2}{10} = 0.2.$$

As a result, *School1* and *School2* are going to be clustered together more readily than *School1* and *School3*. The *silhouette coefficient* is used to identify a suitable number of clusters, taking into account cohesion within clusters and separation between clusters. The relationships between resources in different clusters are considered to be valid relationships between the corresponding clusters.

Hierarchical clustering has also been used in schema inference for property graphs [14]. In contrast with [25], here the clustering is top-down, and clusters are divided into sub-clusters taking into account the annotations of the nodes in the clusters and the most frequent property names.

Kellou-Menouer and Kedad also propose an approach to schema discovery for RDF data that builds on clustering [50]. The input is an RDF graph, and the output is essentially an E/R model with inheritance, though this is represented as an RDF graph in the paper. Given an RDF resource (a node in an RDF graph), the user-defined properties associated with the resource are extracted; these properties constitute resource descriptions, and include incoming and outgoing relationships. The evidence used for schema discovery is the user-defined properties. The first step in the approach uses a density-based clustering algorithm [32] to group together resource descriptions based on a property vector that associates each property with a probability that the property is found in a resource description in the cluster. Each cluster represents a type. Given the types from clustering, the property vector may indicate that some properties are always present, and that some are sometimes present. This information is used to create a type hierarchy, where the less frequently occurring properties are used to identify subtypes.

In RDF, resources can be annotated (using *rdf:type*) with one or more types. As this type information is partial, the clustering approaches described above for RDF use relationship names as evidence, and ignore any available type annotations. With a view to making the most of the available evidence, HInT [48] combines relationship names with annotations when inferring types for the inferred schema. Initially, patterns are created that represent the relationships in which an instance participates, and these patterns are used as keys in an approximate

[14]In Table 4, when considering RDF, as triples can contain both literals and node identifiers; we do not distinguish between attributes and relationships, considering both to be relationships.

(LSH [44]) index that provides access to the corresponding type annotations. Then types for the global schema are created that share relationship and type information. In this approach, the result is essentially an E/R model, in which the patterns provide the relationships of the inferred types, which in turn have been created taking into account *rdf:type* annotations on RDF resources.

Some schema inference techniques build directly on the relationships in the underlying data. For example, graph summarization can identify nodes that are considered to be equivalent based on the graph of relationships in which they participate. For example, Goasdoué, *et al.* [37] propose several definitions of node equivalence that use different types of evidence, which may lead to more or less concise summaries; there is not necessarily a *correct* level of summarization. For example, with respect to Figure 1, is it most appropriate to infer that there is a single *school* type, or to have separate types for *school location* and *school attainment*? Also building on graph structures, Tsuboi and Suzuku [89] infer type descriptions using an expression that captures the co-occurrence of relationship names. The type descriptions are inferred by refining clusters created on the basis of relationship names, by combining similar clusters and identifying mutually exclusive patterns. The type descriptions inferred would be able to say that a *school* description contains either *address* or *attainment* data, if this is the case in the underlying data (as in Figure 1).

Most approaches that seek to produce a summary that captures the main features of the available data, at least initially produce an essentially flat schema, without inheritance-style relationships. However, it is also possible to generate a summary schema that puts hierarchical relationships first, as in HIELDS [24]. In HIELDS, a hierarchy is constructed, in which the most general type is at *Level 0*, of which all values are members. Then at *Level 1* is a collection of more specialised types, which can be distinguished using relationship-value pairs for a single relationship. At *Level 2*, the types from *Level 1* are further subdivided, using evidence from a further relationship-value pair. A suitable subdivision is identified using an optimization algorithm that seeks to satisfy various objectives relating to the size of the groups and their cohesion. For example, in an open government example, *Level 0* might group on the basis of the *rdf:type* to provide groups that include one associated with the type *School*. The school group could then be further divided on the basis of different properties of school, for example by *Borough* or *Town*.

6.2.3 Inferring Concise Summaries. The previously described results for schema inference have sought to infer complete (Section 6.2.1) or reasonably comprehensive (Section 6.2.2) descriptions of the available data. However, the resulting schemas may be substantial, and a more concise summary may be useful for highlighting the main concepts in a repository.

The benefits of producing a concise summary of a complex schema have been recognised for a considerable time, and initial investigations were in the context of a single schema, rather than heterogeneous sources. For example, Yu and Jagadish [98] provide an approach, primarily focused on hierarchical data models (though applicable to relational) that takes into account the relative importance of different schema elements that are candidates for inclusion in a summary and the coverage of the schema elements from the input in the summary. The objective is to include important elements in the summary, while also representing a significant portion of the input. Focusing on relational schemas, Yang *et al.* [96] use entropy to characterise the information content and thus importance of individual tables, taking into account the relationships in which they participate. Tables are then clustered in a way that considers both the similarity and importance of the tables, such that important tables have a good chance of being cluster centers, and thus summary tables. The number of tables in the summary is a parameter.

Changing the setting away from an individual database schema to a potentially large and complex RDF graph, Yan *et al.* [95] describe an approach to inferring a specified number of summary tables that represent important types and attributes. The identification of summaries is cast as an optimization problem, in which metrics are used to score tables and attributes on the basis of the value that they provide, and constraints are used to control

how many such tables are produced. There is also an option in the formulation of the optimization problem to prefer tight previews (in which tables in the summary cover closely related concepts) or diverse previews (in which tables in the summary may include attributes from more loosely related concepts).

6.3 Schema Inference Conclusions

State-of-the-art. There is considerable variety in the schema inference literature, reflecting the different purposes for which a schema can be inferred. Overall, schema inference seeks to consolidate complex heterogeneous structures, which supports data discovery and exploration by providing an integrated and more concise representation of the available data. However, most techniques are input data-model specific, not only in the sense that the input datasets must all conform to a single data model, but also in the sense that much of the evidence used to inform the schema inference process is specific to that data model. This has allowed existing approaches to exploit model-specific features to carry out inferences, but reduces applicability to homogeneous collections.

Benchmarking and Performance. Evaluations of schema inference techniques investigate both the *effectiveness* and *scalability* of the inference process.

The notion of *effectiveness* varies with *Purpose*. Where the *Purpose* is *Query*, the structure of the inferred schema tends to be derived very directly from the underlying datasets, and thus the inferred schema may be correct in some well defined sense. As a result, effectiveness may make use of metrics relating to succinctness (how much smaller the inferred schema is than the underlying data [6]) or precision (how closely the inferred schema aligns with a ground truth schema [9]). Where the *Purpose* is to *Document* the available data, steps in the inference processes tend to be based on uncertain similarity measures, and effectiveness typically relates the inferred schema to a manually created or pre-existing ground truth. For example, DBpedia [55] has been used in the evaluation of several RDF proposals (e.g., [15, 37, 93]), as individual RDF resources in DBpedia have been manually associated with classes from the DBpedia ontology. Thereby, inferred clusters of RDF resources can be related to manually created class annotations. Where the *Purpose* is *Summarize*, sometimes the technique produces a summary of a given size; in this case, a measure of how effective this has been is to compare the inferred summary to a manually produced summary of the same size [98]. A further measure relates to the fraction of the structural elements in the original datasets that are represented in the summary [37].

In relation to *scalability*, schema inference algorithms may run at the schema level or the instance level. For example, algorithms for RDF tend to cluster individual RDF resources [25, 48, 50], and algorithms for XML or JSON cluster individual documents [6, 9]. In contrast, algorithms over tabular datasets cluster the tables and not the rows, as every row in a table has a consistent schema [96, 98]; this clearly has a significant impact on the cardinality of the input to the inference process. A variety of techniques have been applied to support scalability, which tend to exploit parallelism or indexing. As can be seen in Table 4, a common initial step involves clustering data items, such as RDF resources, based on similarity. Let's assume that the similarity of RDF resources is based on the Jaccard similarity of their properties (as in [25, 48, 50] from Table 4). In this setting, one approach is to adopt a parallel clustering algorithm, where RDF resources are assigned to nodes for parallel clustering in a way that takes into account their Jaccard similarity [15]. Such similarity patterns can also be indexed using LSH indexes, to avoid the need for exhaustive comparison of instances during clustering [48]. Scalability techniques tend to be evaluated on a mixture of real and synthetic datasets (e.g., as in [37, 48], where RDF datasets of up to 100 million triples are experimented with). However, the diversity of data-models supported and purposes of the techniques militates against the creation of widely used community benchmarks.

Directions. There are now a wide range of results that support schema inference for different purposes over specific data models. However, the fact that existing techniques target specific input data models means that they are not applicable to heterogeneous repositories, thus reducing their applicability. The aspiration of providing

clarity in complex data environments is significantly blunted if heterogeneous environments are out of scope. Similarly, schema inference techniques often lean heavily on specific properties of the input, for example the hierarchical nature of JSON or XLM, or shared vocabularies in RDF. As a result, approaches do not account for common schematic heterogeneities; recourse to additional sources of evidence (e.g., external ontologies, word embedddings) could be of value for identifying and thus taking account of further heterogeneities.

7 CONCLUSIONS

Dataset discovery and exploration is an area of growing importance: there is ever more data with the potential to provide insights, and the need to make the most of such data as a differentiating factor in business is increasingly widely recognised [61]. This growing momentum, and the recognition that many organisations are not making full use of the available data, is driving research on different aspects of, and approaches to, data discovery and exploration. In this survey, we have reviewed and compared representative technical results in dataset search, data navigation, data annotation and schema inference. We have brought together these complementary areas in a single survey with a view to: (i) highlighting the variety of challenges faced when trying to identify and obtain an understanding of the relevant data in a repository; and (ii) making explicit the diverse technical proposals that have been made with a view to making complex data repositories more accessible to data scientists and engineers. In so doing, we have: (i) identified how common objectives can be addressed in different ways, for example with navigation between related datasets supported directly at the dataset level, implicitly within dataset search or more abstractly through schema inference; (ii) identified how these research directions can support each other, for example, table annotations with richer and shared semantics can directly augment dataset search and data navigation, while descriptive and summary schemas could be utilized for table annotation as evidence; and (iii) identified where recurring themes have surfaced across different tasks, for example the use of inferred relationships for both navigation and search, and the use of embeddings to support scalable approaches to inconsistent data representations in search, navigation and annotation. Our aim has been to provide a consolidated review of the literature, bridging adjacent technical areas, and encouraging a more integrated view of approaches to dataset discovery and exploration.

REFERENCES

[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *VLDB J.* 24, 4 (2015), 557–581.

[2] Francisca Adoma Acheampong, Henry Nunoo-Mensah, and Wenyu Chen. 2021. Transformer models for text-based emotion detection: a review of BERT-based approaches. *Artif. Intell. Rev.* 54, 8 (2021), 5789–5829. https://doi.org/10.1007/s10462-021-09958-2

[3] Felipe Almeida and Geraldo Xexéo. 2019. Word Embeddings: A Survey. *CoRR* abs/1901.09069 (2019). arXiv:1901.09069 http://arxiv.org/abs/1901.09069

[4] Ahmad Alobaid, Emilia Kacprzak, and Óscar Corcho. 2021. Typology-based semantic labeling of numeric tabular data. *Semantic Web* 12, 1 (2021), 5–20. https://doi.org/10.3233/SW-200397

[5] Judie Attard, Fabrizio Orlandi, Simon Scerri, and Sören Auer. 2015. A systematic review of open government data initiatives. *Gov. Inf. Q.* 32, 4 (2015), 399–418. https://doi.org/10.1016/j.giq.2015.07.006

[6] Mohamed Amine Baazizi, Dario Colazzo, Giorgio Ghelli, and Carlo Sartiani. 2019. Parametric schema inference for massive JSON datasets. *VLDB J.* 28, 4 (2019), 497–521. https://doi.org/10.1007/s00778-018-0532-7

[7] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In *Proc. 14th WWW*. 651–660. https://doi.org/10.1145/1060745.1060840

[8] Philip A. Bernstein and Laura M. Haas. 2008. Information integration in the enterprise. *CACM* 51, 9 (2008), 72–79. https://doi.org/10.1145/1378727.1378745

[9] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. 2007. Inferring XML Schema Definitions from XML Data. In *Proc. 33rd VLDB*. 998–1009. http://www.vldb.org/conf/2007/papers/research/p998-bex.pdf

[10] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2015. TabEL: Entity Linking in Web Tables. In *Proc. 14th ISWC, Part I (LNCS, Vol. 9366)*. Springer, 425–441. https://doi.org/10.1007/978-3-319-25007-6_25

[11] Christian Bizer, Tom Heath, and Tim Berners-Lee. 2009. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.* 5, 3 (2009), 1–22. https://doi.org/10.4018/jswis.2009081901

[12] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *36th IEEE ICDE*. IEEE, 709–720. https://doi.org/10.1109/ICDE48307.2020.00067

[13] Alex Bogatu, Norman W. Paton, Mark Douthwaite, and André Freitas. 2022. Voyager: Data Discovery and Integration for Onboarding in Data Science. In *Proc. 25th EDBT*. 2:537–2:548. https://doi.org/10.48786/edbt.2022.47

[14] Angela Bonifati, Stefania Dumbrava, and Nicolas Mir. 2022. Hierarchical Clustering for Property Graph Schema Discovery. In *Proc. 25th EDBT*. 2:449–2:453. https://doi.org/10.48786/edbt.2022.39

[15] Redouane Bouhamoum, Zoubida Kedad, and Stéphane Lopes. 2020. Scalable Schema Discovery for RDF Data. *Trans. Large Scale Data Knowl. Centered Syst.* 46 (2020), 91–120. https://doi.org/10.1007/978-3-662-62386-2_4

[16] Dan Brickley, Matthew Burgess, and Natasha Noy. 2019. Google Dataset Search: Building a Search Engine for Datasets in an Open Web Ecosystem. In *WWW*. ACM, 1365–1375. https://doi.org/10.1145/3308558.3313685

[17] Michael J. Cafarella, Alon Halevy, and Nodira Khoussainova. 2009. Data Integration for the Relational Web. *Proc. VLDB Endow.* 2, 1 (aug 2009), 1090–1101. https://doi.org/10.14778/1687627.1687750

[18] Michael J. Cafarella, Alon Y. Halevy, Hongrae Lee, Jayant Madhavan, Cong Yu, Daisy Zhe Wang, and Eugene Wu. 2018. Ten Years of WebTables. *Proc. VLDB Endow.* 11, 12 (2018), 2140–2149. https://doi.org/10.14778/3229863.3240492

[19] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Trans. Knowl. Data Eng.* 30, 9 (2018), 1616–1637. https://doi.org/10.1109/TKDE.2018.2807452

[20] Sejla Cebiric, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. 2019. Summarizing semantic graphs: a survey. *VLDB J.* 28, 3 (2019), 295–327. https://doi.org/10.1007/s00778-018-0528-3

[21] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. 2020. Dataset search: a survey. *VLDB J.* 29, 1 (2020), 251–272. https://doi.org/10.1007/s00778-019-00564-x

[22] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. 2019. ColNet: Embedding the Semantics of Web Tables for Column Type Prediction. In *The 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, The 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*. AAAI Press, 29–36. https://doi.org/10.1609/aaai.v33i01.330129

[23] Zhiyu Chen, Mohamed Trabelsi, Jeff Heflin, Yinan Xu, and Brian D. Davison. 2020. Table Search Using a Deep Contextualized Language Model *(SIGIR '20)*. ACM, 589–598. https://doi.org/10.1145/3397271.3401044

[24] Gong Cheng, Cheng Jin, and Yuzhong Qu. 2016. HIEDS: A Generic and Efficient Approach to Hierarchical Dataset Summarization. In *Proc. 25th IJCAI*. 3705–3711. http://www.ijcai.org/Abstract/16/521

[25] Klitos Christodoulou, Norman W. Paton, and Alvaro A. A. Fernandes. 2015. Structure Inference for Linked Data Sources Using Clustering. *Trans. Large Scale Data Knowl. Centered Syst.* 19 (2015), 1–25. https://doi.org/10.1007/978-3-662-46562-2_1

[26] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319. https://doi.org/10.5555/3430915.3442430

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. ACL, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[28] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In *37th IEEE ICDE*. 456–467. https://doi.org/10.1109/ICDE51399.2021.00046

[29] Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. 2015. Building the Dresden Web Table Corpus: A Classification Approach. In *2nd IEEE/ACM International Symposium on Big Data Computing, BDC*. 41–50. https://doi.org/10.1109/BDC.2015.30

[30] Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. 2017. Matching Web Tables with Knowledge Base Entities: From Entity Lookups to Entity Embeddings. In *The Semantic Web - ISWC Part I (LNCS, Vol. 10587)*. Springer, 260–277. https://doi.org/10.1007/978-3-319-68288-4_16

[31] Rebecca Eichler, Corinna Giebler, Christoph Gröger, Holger Schwarz, and Bernhard Mitschang. 2021. Modeling metadata in data lakes—A generic model. *Data & Knowledge Engineering* 136 (2021), 101931. https://doi.org/10.1016/j.datak.2021.101931

[32] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*. AAAI Press, 226–231. http://www.aaai.org/Library/KDD/1996/kdd96-037.php

[33] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *34th IEEE ICDE*. 1001–1012. https://doi.org/10.1109/ICDE.2018.00094

[34] Raul Castro Fernandez, Essam Mansour, Abdulhakim Ali Qahtan, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery. In *34th IEEE ICDE*. 989–1000. https://doi.org/10.1109/ICDE.2018.00093

[35] Manuel Fiorelli and Armando Stellato. 2020. Lifting Tabular Data to RDF: A Survey. In *Metadata and Semantic Research - 14th International Conference, MTSR (Communications in Computer and Information Science, Vol. 1355)*. Springer, 85–96. https://doi.org/10.1007/978-3-030-

71903-6_9
[36] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. 2014. DIADEM: Thousands of Websites to a Single Database. *Proc. VLDB Endow.* 7, 14 (2014), 1845–1856. https://doi.org/10.14778/2733085.2733091
[37] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. 2020. RDF graph summarization for first-sight structure discovery. *VLDB J.* 29, 5 (2020), 1191–1218. https://doi.org/10.1007/s00778-020-00611-y
[38] Rihan Hai, Christoph Quix, and Matthias Jarke. 2021. Data lake concept and systems: a survey. *CoRR* abs/2106.09592 (2021). arXiv:2106.09592 https://arxiv.org/abs/2106.09592
[39] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing Google's Datasets. In *Proc ACM SIGMOD*. 795–806. https://doi.org/10.1145/2882903.2903730
[40] Yuan He, Jiaoyan Chen, Denvar Antonyrajah, and Ian Horrocks. 2022. BERTMap: A BERT-Based Ontology Alignment System. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI*. AAAI Press, 5684–5691. https://ojs.aaai.org/index.php/AAAI/article/view/20510
[41] Jan Hegewald, Felix Naumann, and Melanie Weis. 2006. XStruct: Efficient Schema Extraction from Multiple and Large XML Documents. In *Proc. 22nd IEEE ICDE Workshops*. 81. https://doi.org/10.1109/ICDEW.2006.166
[42] Ahmed Helal, Mossad Helali, Khaled Ammar, and Essam Mansour. 2021. A Demonstration of KGLac: A Data Discovery and Enrichment Platform for Data Science. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2675–2678. https://doi.org/10.14778/3476311.3476317
[43] Joseph M. Hellerstein et al. 2017. Ground: A Data Context Service. In *8th CIDR*. http://cidrdb.org/cidr2017/papers/p111-hellerstein-cidr17.pdf
[44] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proc. 30th ACM Symposium on the Theory of Computing*. 604–613. https://doi.org/10.1145/276698.276876
[45] Ernesto Jiménez-Ruiz, Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, and Kavitha Srinivas. 2020. SemTab 2019: Resources to Benchmark Tabular Data to Knowledge Graph Matching Systems. In *The Semantic Web - 17th ESWC (LNCS, Vol. 12123)*, Andreas Harth et al. (Eds.). Springer, 514–530. https://doi.org/10.1007/978-3-030-49461-2_30
[46] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G. Skjæveland, Evgenij Thorstensen, and Jose Mora. 2015. BootOX: Practical Mapping of RDBs to OWL 2. In *The Semantic Web - ISWC Part II (LNCS, Vol. 9367)*. Springer, 113–132. https://doi.org/10.1007/978-3-319-25010-6_7
[47] Yannis Kalfoglou and W. Marco Schorlemmer. 2003. Ontology mapping: the state of the art. *Knowl. Eng. Rev.* 18, 1 (2003), 1–31. https://doi.org/10.1017/S0269888903000651
[48] Nikolaos Kardoulakis, Kenza Kellou-Menouer, Georgia Troullinou, Zoubida Kedad, Dimitris Plexousakis, and Haridimos Kondylakis. 2021. HInT: Hybrid and Incremental Type Discovery for Large RDF Data Sources. In *SSDBM 2021: 33rd International Conference on Scientific and Statistical Database Management*. 97–108. https://doi.org/10.1145/3468791.3468808
[49] Kenza Kellou-Menouer, Nikolaos Kardoulakis, Georgia Troullinou, Zoubida Kedad, Dimitris Plexousakis, and Haridimos Kondylakis. 2022. A survey on semantic schema discovery. *The VLDB Journal* 31 (2022), 675–710.
[50] Kenza Kellou-Menouer and Zoubida Kedad. 2015. Schema Discovery in RDF Data Sources. In *Conceptual Modeling - 34th International Conference, ER (LNCS, Vol. 9381)*. Springer, 481–495. https://doi.org/10.1007/978-3-319-25264-3_36
[51] Vijay Khatri and Carol V. Brown. 2010. Designing data governance. *Commun. ACM* 53, 1 (2010), 148–152. https://doi.org/10.1145/1629175.1629210
[52] Won Kim, Injun Choi, Sunit K. Gala, and Mark Scheevel. 1993. On Resolving Schematic Heterogeneity in Multidatabase Systems. *Distributed and Parallel Databases* 1, 3 (1993), 251–279. https://doi.org/10.1007/BF01263333
[53] Jakub Klímek and Martin Necaský. 2010. Reverse-engineering of XML Schemas: A Survey. In *Proc. Dateso International Workshop on DAtabases, TExts, Specifications and Objects*. 96–107. http://ceur-ws.org/Vol-567/paper19.pdf
[54] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *37th IEEE ICDE*. 468–479. https://doi.org/10.1109/ICDE51399.2021.00047
[55] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195. https://doi.org/10.3233/SW-140134
[56] Maurizio Lenzerini. 2002. Data integration: a theoretical perspective. In *PODS*. 233–246.
[57] Aristotelis Leventidis, Laura Di Rocco, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. 2021. DomainNet: Homograph Detection for Data Lake Disambiguation. In *Proc. 24th EDBT*. 13–24. https://doi.org/10.5441/002/edbt.2021.03
[58] Keqian Li, Yeye He, and Kris Ganjam. 2017. Discovering Enterprise Concepts Using Spreadsheet Tables. In *Proc. 23rd ACM SIGKDD*. 1873–1882. https://doi.org/10.1145/3097983.3098102
[59] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. VLDB Endow.* 3, 1 (2010), 1338–1347. https://doi.org/10.14778/1920841.1921005

[60] Chang Liu, Arif Usta, Jian Zhao, and Semih Salihoglu. 2023. Governor: Turning Open Government Data Portals into Interactive Databases. In *Proc CHI*. 415:1–415:16. https://doi.org/10.1145/3544548.3580868
[61] Jorn Lyseggen. 2017. *Outside Insight*. Portfolio Penguin.
[62] Michael V. Mannino, Paicheng Chu, and Thomas Sager. 1988. Statistical Profile Estimation in Database Systems. *ACM Comput. Surv.* 20, 3 (1988), 191–221. https://doi.org/10.1145/62061.62063
[63] Víctor Martínez, Fernando Berzal, and Juan Carlos Cubero Talavera. 2017. A Survey of Link Prediction in Complex Networks. *ACM Comput. Surv.* 49, 4 (2017), 69:1–69:33. https://doi.org/10.1145/3012704
[64] Paolo Missier, Khalid Belhajjame, and James Cheney. 2013. The W3C PROV family of specifications for modelling provenance metadata. In *EDBT Proceedings*. ACM, 773–776. https://doi.org/10.1145/2452376.2452478
[65] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. 2020. Organizing Data Lakes for Navigation. In *Proc. ACM SIGMOD*. 1939–1950. https://doi.org/10.1145/3318464.3380605
[66] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* 11, 7 (2018), 813–825. https://doi.org/10.14778/3192965.3192973
[67] Sebastian Neumaier, Jürgen Umbrich, Josiane Xavier Parreira, and Axel Polleres. 2016. Multi-level Semantic Labelling of Numerical Values. In *15th ISWC, Part I (LNCS, Vol. 9981)*. 428–445. https://doi.org/10.1007/978-3-319-46523-4_26
[68] Phuc Nguyen, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. 2019. MTab: Matching Tabular Data to Knowledge Graph using Probability Models. In *Proc. Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (CEUR, Vol. 2553)*. CEUR-WS.org, 7–14. http://ceur-ws.org/Vol-2553/paper2.pdf
[69] Masayo Ota, Heiko Mueller, Juliana Freire, and Divesh Srivastava. 2020. Data-Driven Domain Discovery for Structured Datasets. *Proc. VLDB Endow.* 13, 7 (2020), 953–965. https://doi.org/10.14778/3384345.3384346
[70] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *Proc. VLDB Endow.* 14, 12 (2021), 2863–2866. https://doi.org/10.14778/3476311.3476364
[71] Heiko Paulheim and Christian Bizer. 2013. Type Inference on Noisy RDF Data. In *Proc. 12th ISWC Part I (LNCS, Vol. 8218)*. Springer, 510–525. https://doi.org/10.1007/978-3-642-41335-3_32
[72] Jorge Pérez, Marcelo Arenas, and Claudio Gutiérrez. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009), 16:1–16:45. https://doi.org/10.1145/1567274.1567278
[73] Federico Piai, Paolo Atzeni, Paolo Merialdo, and Divesh Srivastava. 2023. Fine-grained semantic type discovery for heterogeneous sources using clustering. *VLDB J.* 32, 2 (2023), 305–324. https://doi.org/10.1007/s00778-022-00743-3
[74] Giulio Ermanno Pibiri and Rossano Venturini. 2021. Techniques for Inverted Index Compression. *ACM Comput. Surv.* 53, 6 (2021), 125:1–125:36. https://doi.org/10.1145/3415148
[75] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web Using Column Keywords. *Proc. VLDB Endow.* 5, 10 (jun 2012), 908–919. https://doi.org/10.14778/2336664.2336665
[76] G. Press. 2016. Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says. *Forbes* (2016). https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=4ded67636f63
[77] Jay Pujara, Pedro A. Szekely, Huan Sun, and Muhao Chen. 2021. From Tables to Knowledge: Recent Advances in Table Understanding. In *Proc. 27th ACM SIGKDD*. ACM, 4060–4061. https://doi.org/10.1145/3447548.3470809
[78] E. Rahm and P. A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDBJ* 10, 4 (2001), 334–350. https://doi.org/10.1007/s007780100057
[79] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML Tables to DBpedia. In *Proc. 5th Intl Conf on Web Intelligence, Mining and Semantics, WIMS*. ACM, 10:1–10:6. https://doi.org/10.1145/2797115.2797118
[80] Nicolás Robinson-García, Evaristo Jiménez-Contreras, and Daniel Torres-Salinas. 2016. Analyzing data citation practices using the data citation index. *J. Assoc. Inf. Sci. Technol.* 67, 12 (2016), 2964–2975. https://doi.org/10.1002/asi.23529
[81] Aécio Santos, Aline Bessa, Christopher Musco, and Juliana Freire. 2022. A Sketch-based Index for Correlated Dataset Search. In *2022 IEEE ICDE*. 2928–2941. https://doi.org/10.1109/ICDE53745.2022.00264
[82] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *ACM SIGMOD*. 817–828. https://doi.org/10.1145/2213836.2213962
[83] Pegdwendé N. Sawadogo and Jérôme Darmont. 2021. On data lake architectures and metadata management. *CoRR* abs/2107.11152 (2021). arXiv:2107.11152 https://arxiv.org/abs/2107.11152
[84] Julian Seitner, Christian Bizer, Kai Eckert, Stefano Faralli, Robert Meusel, Heiko Paulheim, and Simone Paolo Ponzetto. 2016. A Large DataBase of Hypernymy Relations Extracted from the Web. In *Proc. 10th Intl Conf on Language Resources and Evaluation LREC*. http://www.lrec-conf.org/proceedings/lrec2016/summaries/204.html
[85] Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Trans. Knowl. Data Eng.* 27, 2 (2015), 443–460. https://doi.org/10.1109/TKDE.2014.2327028

[86] Yasin N. Silva, Walid G. Aref, and Mohamed H. Ali. 2010. The similarity join database operator. In *Proc. 26th IEEE ICDE*. 892–903. https://doi.org/10.1109/ICDE.2010.5447873

[87] Gianmaria Silvello. 2018. Theory and practice of data citation. *J. Assoc. Inf. Sci. Technol.* 69, 1 (2018), 6–20. https://doi.org/10.1002/asi.23917

[88] Pranav Subramaniam, Yintong Ma, Chi Li, Ipsita Mohanty, and Raul Castro Fernandez. 2021. Comprehensive and Comprehensible Data Catalogs: The What, Who, Where, When, Why, and How of Metadata Management. *CoRR* abs/2103.07532 (2021). arXiv:2103.07532 https://arxiv.org/abs/2103.07532

[89] Yuroti Tsuboi and Nobutaka Suzuki. 2019. An Algorithm for Extracting Shape Expression Schemas from Graphs. In *Proc. ACM Symposium on Document Engineering 2019*. 32:1–32:4. https://doi.org/10.1145/3342558.3345417

[90] Panos Vassiliadis. 2011. A Survey of Extract-Transform-Load Technology. In *Integrations of Data Warehousing, Data Mining and Database Technologies*. Information Science Reference, 171–199. https://doi.org/10.4018/978-1-60960-537-7.ch008

[91] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering Semantics of Tables on the Web. *Proc. VLDB Endow.* 4, 9 (2011), 528–538. https://doi.org/10.14778/2002938.2002939

[92] Ruben Verborgh and Max De Wilde. 2013. *Using OpenRefine*. Packt Publishing Ltd.

[93] Johanna Völker and Mathias Niepert. 2011. Statistical Schema Induction. In *8th Extended Semantic Web Conference, ESWC Part I (LNCS, Vol. 6643)*. Springer, 124–138. https://doi.org/10.1007/978-3-642-21034-1_9

[94] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *Proc. ACM SIGMOD*. 97–108. https://doi.org/10.1145/2213836.2213848

[95] Ning Yan, Sona Hasani, Abolfazl Asudeh, and Chengkai Li. 2016. Generating Preview Tables for Entity Graphs. In *Proc. ACM SIGMOD*. 1797–1811. https://doi.org/10.1145/2882903.2915221

[96] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. 2009. Summarizing Relational Databases. *Proc. VLDB Endow.* 2, 1 (2009), 634–645. https://doi.org/10.14778/1687627.1687699

[97] Dani Yogatama, Daniel Gillick, and Nevena Lazic. 2015. Embedding Methods for Fine Grained Entity Type Classification. In *Proc. 53rd ACL*. 291–296. https://doi.org/10.3115/v1/p15-2048

[98] Cong Yu and H. V. Jagadish. 2006. Schema Summarization. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*. 319–330. http://dl.acm.org/citation.cfm?id=1164156

[99] Shuo Zhang and Krisztian Balog. 2017. EntiTables: Smart Assistance for Entity-Focused Tables. In *Proc. 40th ACM SIGIR*. Association for Computing Machinery, 255–264. https://doi.org/10.1145/3077136.3080796

[100] Shuo Zhang and Krisztian Balog. 2018. Ad Hoc Table Retrieval Using Semantic Similarity. In *Proc. WWW*. Republic and Canton of Geneva, CHE, 1553–1562. https://doi.org/10.1145/3178876.3186067

[101] Shuo Zhang and Krisztian Balog. 2020. Web Table Extraction, Retrieval, and Augmentation: A Survey. *ACM Trans. Intell. Syst. Technol.* 11, 2 (2020), 13:1–13:35. https://doi.org/10.1145/3372117

[102] Yi Zhang and Zachary G. Ives. 2019. Juneau: Data Lake Management for Jupyter. *Proc. VLDB Endow.* 12, 12 (2019), 1902–1905. https://doi.org/10.14778/3352063.3352095

[103] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proc. ACM SIGMOD*. ACM, 1951–1966. https://doi.org/10.1145/3318464.3389726

[104] Ziqi Zhang. 2017. Effective and efficient Semantic Table Interpretation using TableMiner$^+$. *Semantic Web* 8, 6 (2017), 921–957. https://doi.org/10.3233/SW-160242

[105] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proc. ACM SIGMOD*. ACM, 847–864. https://doi.org/10.1145/3299869.3300065

[106] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-Join: Joining Tables by Leveraging Transformations. *Proc. VLDB Endow.* 10, 10 (2017), 1034–1045. https://doi.org/10.14778/3115404.3115409

[107] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proc. VLDB Endow.* 9, 12 (2016), 1185–1196. https://doi.org/10.14778/2994509.2994534