Finding Related Tables in Data Lakes for Interactive Data Science

Yi Zhang, Zachary G. Ives [original] [1] — Felix Campbell, Chenjie Li, Jinchen Tang [summary]

Abstract

As data lakes continue to grow in size, finding related tables becomes an intractable problem given the volume, schema-agnosticism, and unstructured nature of the data. The authors present a technique which they implement as an extension over the Jupyter notebook system to efficiently find related tables. This enables a low-latency, interactive approach to finding related tables for tasks such as data cleaning, data integration, or feature engineering for machine learning.

1 Background

The notion of what makes a table related to another is variable depending on the goal. The authors present a few main tasks that they target for determining relatedness: data cleaning, data integration, and feature engineering. The following section describes the metrics used, and how they can be composed together to create similarity metrics specific for a task.

While data similarity is an obvious metric, the authors note that the workflows produced by data scientists in notebook-based platforms such as Jupyter or RStudio can be useful in determining relatedness between tables. By tracking the execution order of cells and their data dependencies, a workflow graph can be stored that would otherwise be untracked.

2 Similarity

The paper introduces a set of data similarity measures, relating to either the row or column similarities, or the additional information added in the form of new rows or columns, or the reduction in null values. Also, the similarity of workflows (provenance) between notebooks is also introduced. We briefly summarize the definitions of those similarity metrics with some toy examples:

2.1 Data Similarity

2.1.1 Row Based Similarity

To measure the similarity of rows, the authors introduced a key mapping which is defined as follows: *When the pairs of attributes k_S , k_T from relation **S** and relation **T** in the key mapping are equal, we have a value-based match between the rows.* Intuitively, this definition tries to find a pair of columns from 2 tables that when they are the same, the rest of the attributes from both tables are uniquely determined, which is very close to functional dependency definition. As shown in Figure 1, we have {*Name*, *Office-Phone*} from table **S** and {*FirstName*, *Work-Phone*} from table **T**, when those 2 sets of attributes' share the same values, the *Address* from **S** and *Hobby* from **T** are uniquely determined.

2.1.2 Column Based Similarity

Compared to row based similarity, which matches exactly the values between tables, the column based similarity focuses more on the domain similarity between columns, i.e., if 2 candidate columns are sharing the same information, e.g. 2 columns are all about phone numbers but they are not the same values. To quantify the relateness based on domain values, they introduced the notion of *Data Profile*, which is denoted

Table **S**

Name	Office-Phone	Address
Peter	312 555 5000	home1
Alice	847 555 6000	home2
Bob	708 555 6000	home3

Table **T**

FirstName	Work-Phone	Hobby
Peter	312 555 5000	Basketball
Alice	847 555 6000	Hockey
Bob	708 555 6000	Coding

Figure 1: Row Based Similarity Example

as $\psi(c, d)$ for column c w.r.t domain d. The definition of ψ is unspecified, but intuitively it is a function that predicts the likelihood of column c belongs to domain d. We provide an example. As shown in Figure 2, *Name* column has 3 distinct values. Suppose we have 3 candidate domains $\mathbf{D} = \{D_{name}, D_{hobby}, D_{number}\}$. If the function we chose to test the membership in terms of which domain from $\mathbf{D}$ the column *Name* belongs to is recall. then we have perfect score for $\mathrm{D_{name}}$ since all values from column *Name* exist in $\mathrm{D_{name}}$.

Table **S**

Name	Office-Phone	Address
Peter	312 555 5000	home1
Alice	847 555 6000	home2
Bob	708 555 6000	home3

Domains **D**

D_{name} = {Peter, Alice, Bob, Kate}
D_{hobby} = {basketball, hockey, coding}
D_{number} = {1234, 234, 2341, 3412}

Figure 2: Column Based Similarity Example: Domain Matching

2.1.3 Provenance Similarity

It has been noted in Section 1 that notebook-based environments can be augmented to provide a workflow graph of the operations performed over the tables in the data lake. Comparing these workflow graphs can be used to determine the similarity of tables, as similar tables may have similar operations performed on them, e.g. cleaning. Given these stored workflow graphs, the graph edit distance is used as the metric to compare provenance similarity of two tables. As shown in the Figure 3, the two graphs on the right capture the workflow of the 2 code snippets on the left. According to the authors' definition, *"The edit operation on a graph G is an insertion or deletion of a vertex/edge or relabeling of an edge."* In this example, to convert the graph below to the graph above requires 2 operations: the deletion of edge labeled with "assignment" and the deletion of the node labeled as "new_iq_for_peter".

2.2 Similarity Approximation

Since in practice the exact matchings/overlappings between 2 data sources is very unlikely to have the property discussed in Section 2.1, the authors proposed the *approximate key constraints* and *approximated domain matches* which is a relaxed version of the previous definition. The approximation is very straightfoward: instead of matching exactly, we provide a threshold smaller than 1. As long as we have "enough" matches/overlaps, we say it is matched/overlapped.

In their practical implementations, the row and column similarity measures are based on the *Jaccard similarities*. The paper also introduces pre-defined *registered matchers* that can be used to identify the

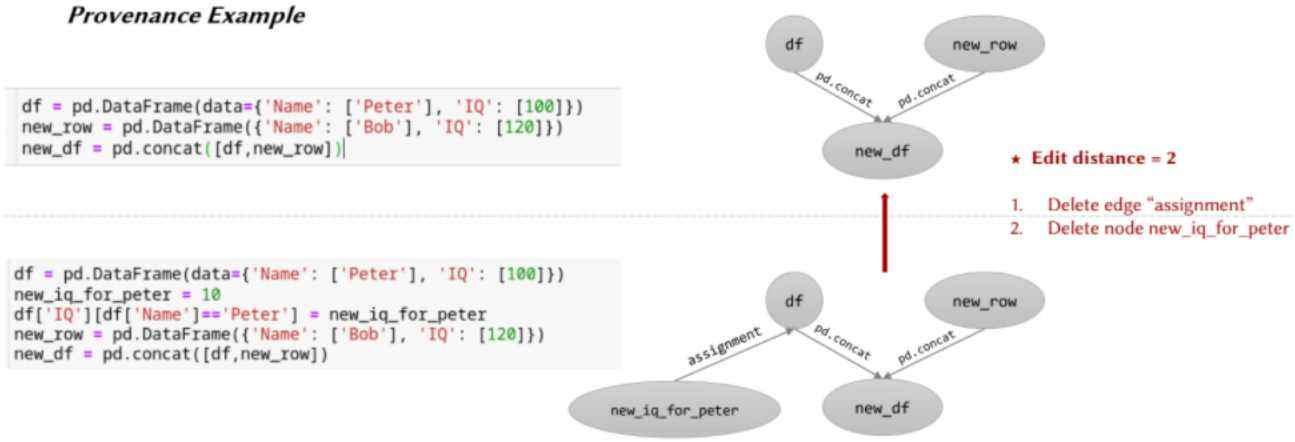


Figure 3: Provenance Similarity Example: Edit distance of workflow

domain of a column. For example, a column matching a regular expression detecting phone numbers is likely to be exactly that. Many of these data similarity measures require the computation of a relation mapping between the tables being compared, which is a relatively expensive algorithm whose use the authors try to minimize. In order to begin filtering some of these tables out, the authors introduce a cheaper form of similarity metric introduced in Section 4.

3 Top-k Processing

Top-K searching algorithm is frequently used concept in Computer Science literature. The essential idea of the approach is to use special data structures to achieve early stopping/reducing search space.

The authors used common top-k processing techniques in order to efficiently score and return related tables. As stated in the previous section, some of these metrics can be expensive when doing them naively, as they rely on the calculation of a relation mapping before the similarity metric can be computed. In order to mitigate this cost, the authors divide their top-k processing algorithm into two initial stages to prune bad candidates.

- **Stage 1** - Tables are matched by their columns' data profiles. The union of these tables linked by matching column domains are processed by the following stage.
- **Stage 2** - Tables are matched by their workflow graphs, further checking for related columns. The resulting set of tables is then processed by a threshold-based top-k algorithm.

At this stage, only tables which have some potential to be related are included for further processing. Partial relation mappings can be used to calculate upper bounds on measures that would otherwise require expensive full relation mappings.

4 Practical approaches for each matching/similarity objective

Given the formal definitions described by the authors before, in real implementations, those measures need to be adjusted/approximated in order to make the searching step feasible. We briefly mention for each measure how the authors generate those approximated measures.

4.1 Finding Relation Mappings

4.1.1 Value-Based Overlap

Jaccard Similarty or some sketches related work can be used here.

4.1.2 Domain-Based Overlap

This is used to handle disjoint/union-able data values. They utilized the following approaches:

- Registered Matchers: e.g. regex matching or domain value type detectors
- Data Profiles: some practical functions used to measure the data profiles we discussed in Section 2.1.2 such as Bayes Rule considering the co-occurring of columns.

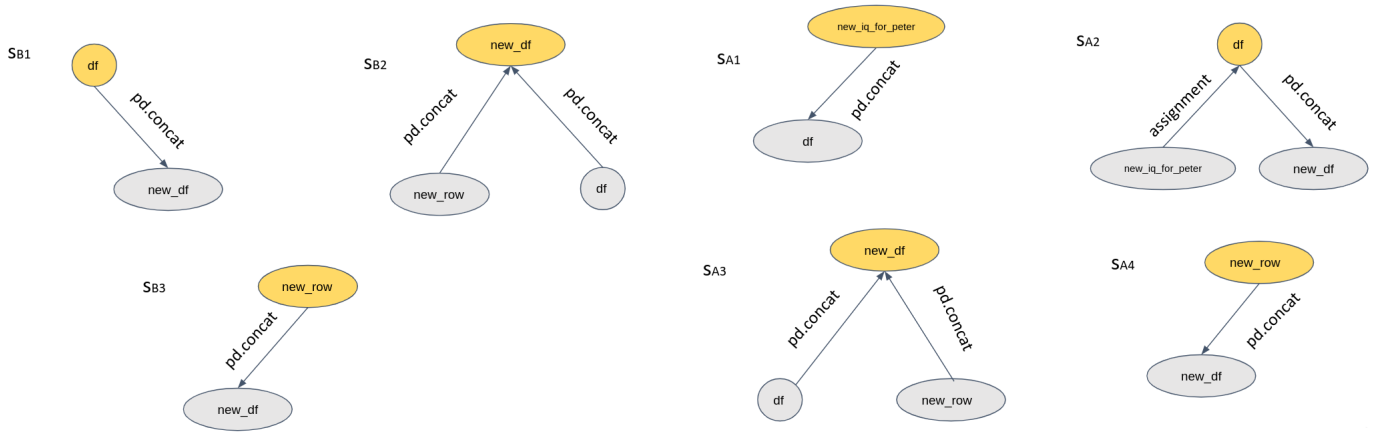
4.2 Relation Mapping

Given the matching scores between every pair of columns, the mappings between tables are solved using linear programming.

4.3 Provenance Similarity Approximation

Computing provenance similarity i.e., the graph edit distance between two variable provenance graphs is NP-hard. In order to have a feasible solution, the authors proposed a *star structure* of the workflows, which is essentially a collection of single level rooted tree. Given the star structures, the edit distance between 2 collection of star structures is defined as: the sum of the min distance from the set of 1st collection of stars to the set of 2nd collection of stars. We provide an example from Figure 3 to illustrate this distance definition.

Suppose the star structure collection S_B represents the workflow of the upper side from Figure 3 and collection S_A represents the workflow of the lower side from Figure 3. Intuitively, by breaking a big workflow graph to smaller "stars", it becomes easier to find the min distance for each star, thus we can speculate the overall edit distance between the original graph. It turns out this transformation provides the lower bound of the distance. (details please refer to the paper).



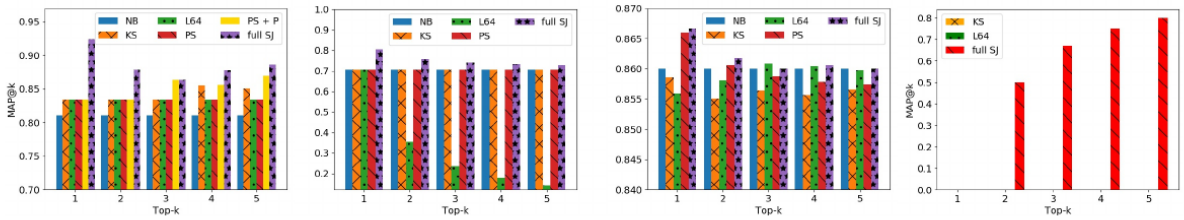
(a) Star Structure for provenance B

(b) Star Structure for provenance A

Figure 4: 2 Star Structures from example in Figure 3

5 Implementation & Experiments

The system the paper implements is named JUNEAU and is built on top of the Jupyter notebook system. Upon the execution of a cell, the tables' data profiles are indexed, and its provenance information stored in



(a) Augmenting Training Data (b) Feature Extraction (c) Data Cleaning (d) Linkable Data

Figure 5: Selected Experiments

a graph database. The system allows users to take the table outputted by a cell and query the system for the top-k tables related by a certain task. In order to continually serve the best results, the weights of each measurement are adjusted as new information is inserted into the data lake.

The paper presents several experiments demonstrating the running time and added precision from including related tables over certain queries. In order to observe the precision added by JUNEAU's recommended tables, the authors select a number of Jupyter notebooks from Kaggle[1] to use as a basis for their evaluation. These notebooks are artifacts produced by Kaggle users completing tasks or competitions, in which the notebook is performing some sort of processing with measurable precision (e.g., classification) or links related tables manually. Table accesses in the notebooks are modified to include the top-k tables recommended by JUNEAU, measuring the resulting average precision (proportional to precision & recall values), with the baseline value being dependent on each task. MAP@k demonstrates the mean average precision while including the top-k related tables in the notebook's processing.

The authors' suite of experiments show that the running time of finding related tables is greatly reduced when compared to other competing techniques. This more efficient technique also maintains the quality of the results as seen in Figure 5, and in some cases offering improvement. However, in some cases, adding a lower ranked table actually improves the mean average precision over its exclusion, as is the case in Figures 5a and 5c between the inclusion of top 3 and 4 tables.

6 Analysis of JUNEAU

As stated in Section 5, JUNEAU provides a much more efficient way to query related tables while maintaining results of similar quality to the baseline and other algorithms. However, the gains in quality are not much more beneficial over the baseline. JUNEAU performs best when being used for augmented training data, while providing marginal improvements in precision as stated by the authors. However, it is rather impressive that quality results can be achieved with in some cases orders of magnitude less execution time than required by a related-table-querying algorithm such as LSH ensembles [2]. This efficient querying is an important contribution, and could have its similarity measurements or approximations refined to provide better results while leveraging the threshold-based top-k processing approach they use.

A difficult problem remains to be examined by the authors, which is the disambiguation of semantic domains. For example, consider the trivial example of Boolean fields. In this case, the value-based and domain-based similarity for Boolean columns would always match, suggesting a relatedness between columns that may not make sense semantically (for example, relating whether or not a given park is dog friendly to whether or not an arrest was made in a given crime). Without reasoning about the problem, a proposal we make would to use of data already encoded that describes its use: the column name. Note that the authors use keyword search over the table level, but do not consider using this over the column level to determine semantic relatedness between columns with overlapping domains.

[1]Kaggle (https://www.kaggle.com) is an online notebook platform supporting data science competitions

7 Conclusion

By composing similarity metrics, the paper presents new metrics for determining the relatedness of two tables as they relate to a certain task. The computation of these metrics can be relatively expensive, which is made cheaper through adapting common top-k processing techniques including thresholding and approximation. Leveraging provenance information is also key to not only providing better results for certain classes of relatedness, but also serving as a more inexpensive form of relatedness to prune tables before calculating more expensive relatedness measures. The system provides clear performance gains while maintaining similar levels of precision, if not improving it. This efficient approach enables finding related tables interactively, allowing data scientists to use a more iterative approach while seeking related tables. The authors intend to expand their approach to include determining relatedness between nested data as well as non-relational data models. While the approximations of the current similarity measures are sound, the similarity measurements could be augmented to rank related tables more accurately. Furthermore, adapting the approach to make semantic distinctions between identical domains is important in filtering out columns that would otherwise appear to be related.

References

[1] Yi Zhang and Zachary G Ives. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1951–1966, 2020.

[2] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. LSH ensemble: Internet-scale domain search. *Proc. VLDB Endow.*, 9(12):1185–1196, 2016.