# Building a Knowledgebase for Incremental Schema Recovery

Poonam Kumari
University at Buffalo, SUNY
poonamku@buffalo.edu

Gourab Mitra
University at Buffalo, SUNY
gourabmi@buffalo.edu

Oliver Kennedy
University at Buffalo, SUNY
okennedy@buffalo.edu

## ABSTRACT

There are few things sadder than an unlabeled (or poorly labeled) dataset. Without names, data access becomes inordinately more difficult or even impossible. In this paper we describe the first steps towards building a new tool that makes it easier for users to properly label data, and to help preserve those labels for future use. Specifically, we outline the design of **LOKI**, a knowledge-base for storing column-naming heuristics, as well as an interactive tool: the **LOKI** editor for populating the knowledge-base. The **LOKI** editor primes the knowledge base by learning from example data (e.g., from open data portals), and assists domain experts in reviewing and refining the resulting heuristic naming schemes. We identify specific issues arising from training and show how the **LOKI** editor streamlines the process of manually repairing these issues.

## KEYWORDS

Schema Design, Knowledge Base, Data Loading, ETL

**ACM Reference Format:**
Poonam Kumari, Gourab Mitra, and Oliver Kennedy. 2018. Building a Knowledgebase for Incremental Schema Recovery. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

It's a story as old as time: A student gathers data, makes a graph with the data, writes a paper about the data. Then the student graduates and the data languishes, without so much as a wiki page or README file documenting it. The next student to use the data needs to spend hours, days, or even weeks reverse-engineering it. Then they also graduate and the whole process can start over again.

This tragic cycle of data abandonment must be broken. We propose to end the suffering with *Label Once, and Keep It* (**LOKI**), a data-ingest middleware for incremental, re-usable schema recovery. When a user first points **LOKI** at a new tabular data set, **LOKI** proposes a schema for it. It then collects feedback, both learning and also preserving schema metadata for later use. In short, **LOKI** will allow users to assemble schemas on-demand, both (re-)discovering and incrementally refining schema definitions in response to changing data needs. However, to accomplish this, we first need a knowledge-base of heuristics, domain knowledge, and dirty tricks.

In principle, we might implement such a knowledge base as a neural network, train it on example data, and use it to suggest schemas. However, this approach suffers from a host of limitations: (1) **Lack of generality**: A neural network tuned for use with one type of training data may need to be retuned for other scenarios. (2) **Lack of extensibility**: Adding new training data or knowledge requires retraining from scratch. (3) **Lack of transparency**: If the network misclassifies a column name, it can be hard to debug and refine the result. Instead, we opt for an interactive approach. We present the **LOKI** knowledge-base, a flexible, extensible infrastructure for collecting schema-naming heuristics. We show how simple, efficient off-the-shelf techniques can be used to quickly prime the knowledge-base using example data. Then, we introduce an interactive editor that we are designing as part of **LOKI** to allow domain experts to refine the knowledge-base. In particular, we focus on our preliminary work to streamline manual refinement of knowledge learned from one type of example data. We show how, the editor can be used to help experts to quickly identify and resolve errors and ambiguity in the **LOKI** knowledge-base.

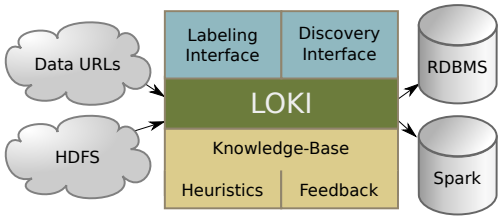



**Figure 1: System Overview**

Concretely, the contributions of this paper include: (1) We introduce **LOKI** in Section 2 and detail the structure of its knowledge-base in Section 3. (2) We illustrate how the **LOKI** editor pre-populates the knowledge-base by learning from example data in Section 4. (3) We identify specific errors that arise in the training process and show how the **LOKI** editor facilitates efficient detection and manual repair of the error in Section 5.

## 2 SYSTEM DESIGN

The overall goal of **LOKI** is to streamline the process of developing schemas for existing unlabeled or poorly labeled data sets. As illustrated in Figure 1, **LOKI** lives alongside an existing RDBMS or Spark deployment, and takes as input tabular data in the form of a URL or HDFS file path. **LOKI** provides users with two modes of interaction: (1) A *labeling* interface that assists users in assigning names to existing columns of data, and (2) A *discovery* interface that helps users to search for columns representing particular concepts of interest. Both interfaces are supported by a knowledge-base that combines expert-provided heuristics, learned characteristics, as well as historical feedback gathered from users about already-loaded datasets. Once the user has labeled or discovered a sufficient set of columns, **LOKI** generates appropriate data loading/initialization code (e.g., a `CREATE TABLE` or Spark DataFrame initializer).

### 2.1 Labeling and Discovery

We assume that data arrives in unlabeled tabular format. An $N$-column input table $T$ consists of a set of rows $t \in T$. Each row is an

*Conference'17, July 2017, Washington, DC, USA*
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

indexed list of attribute values $t = \langle\, t[1], \ldots, t[N] \,\rangle$ with a common schema $sch(T)$. The columns of $T$, denoted $T_C = \{A_1, \ldots A_N\}$, are the *bag* of values in the projection of the rows of T onto the specified attribute: $A_i \stackrel{def}{=} \{\!|\ t[i] \mid t \in T \ |\!\}$. We define the domain of $A$, denoted $dom(A)$ to be the *set* of such values (i.e., $set(T_A)$). A *schema* $S$ for $T$ is a list of $N$ *names*, or human interpretable strings $S[1], \ldots, S[N]$ identifying each attribute. Note that this definition differs slightly from the classical definition of schemas; for us a schema consists only of a name and not the attribute's domain.

**LOKI** addresses two closely related schema construction problems that we term labeling and discovery queries. Both types of queries make use of a form of best-fit heuristic matching that we discuss in depth in Section 3. Specifically, a **Labeling** query derives a schema $S$ for a table $T$. If the user previously provided a names for one of the columns, this name is used in the derived schema. Otherwise, the best-fit heuristic matching is used. A **Discovery** query addresses the dual problem: finding the attribute $A_i$ in a table $T$ most likely to match a given column name.

## 2.2 Other Implementation Concerns

In order to load data into a relational database management system, **LOKI** needs a way to infer the type (e.g., Numeric, String, or Ordinal) of columns of ingested data. For this, we adopt a simple counting-based technique [10]: We attempt to parse each value using an array of regular expressions, one for each registered type. A majority vote of successful parses in a column decides the column's type, with a threshold of 50% indicating a string column and fewer than 100 distinct values indicating an ordinal. Admittedly, more advanced techniques that relate columns, for example using functional dependencies, are possible. However majority vote suffices for most use cases, and as such we leave more interesting type annotation schemes to future work.

Furthermore, **LOKI** needs to actually load data into the underlying RDBMS. We use each platform's native loading scheme: `DataFrameReaders` on Spark, and native `LOAD` commands on RDBMSes, with a fall-back to manual `INSERT` operations if necessary. As before, more intricate loading options are possible [1, 5], but are beyond the scope of this paper.

# 3 KNOWLEDGE-BASE STRUCTURE

The core component of **LOKI** is a knowledge-base that is used to identify column names. The knowledge-base is broken down into two parts, one heuristic-based, and the other feedback-based. The latter part is used to preserve column names explicitly labeled by the user for future use. This is simply a mapping from a table identifier and column position to an explicit name. Hence, for the balance of the paper we will focus on the former, more complex heuristic-based part of the **LOKI** knowledge-base. Specifically, in this section we define the representational semantics of the heuristic knowledge-base and define the evaluation semantics for labeling and discovery queries.

## 3.1 Matchers and Concepts

The heuristic content of the knowledge base is best expressed as a bipartite graph between *matchers* and *concepts*. An example is shown in Figure 2. A matcher is any function $M : T[A] \to [0, 1]$

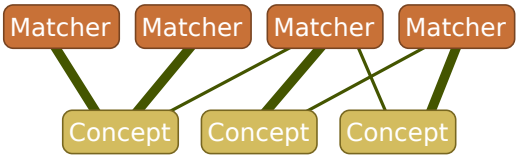



**Figure 2: An example LOKI heuristic knowledge-base**

which assigns any column of data to a $[0, 1]$-valued measure that we call the matcher's *score* for the column. Concepts correspond to names. However, because the same name may be used in different contexts, multiple concepts can use the same column name. Similarly, a single concept may be identified by multiple synonymous names, so a single concept may be associated with multiple names. An edge between a matcher and a concept indicates candidacy: For a given column ranked highly by a matcher, any concept linked to the matcher by an edge is a viable name for the column. Each matcher has a *default* edge, indicated in Figure 2 in bold.

Formally, a knowledge-base $\mathbf{KB} = \langle\, \mathbb{M}, \mathbb{C}, \mathbb{E} \,\rangle$, with matchers $M \in \mathbb{M}$, concepts $C \in \mathbb{C}$, edge matrix $\mathbb{E} : \mathbb{M} \times \mathbb{C} \to \{\mathbf{D}, \top, \bot\}$. Specifically $\mathbb{E}(M, C) \in \{\mathbf{D}, \top\}$ indicates an edge between $M$ and $C$. The value $\mathbf{D}$ identifies the default edge. We require that each matcher have exactly one default edge.

$$\forall M \in \mathbb{M} \quad \exists C \in \mathbb{C} \quad \mathbb{E}(M, C) = \mathbf{D}$$

$$\forall M \in \mathbb{M}, C_1 \in \mathbb{C} \quad \nexists C_2 \neq C_1 \in \mathbb{C} \quad \mathbb{E}(M, C_1) = \mathbb{E}(M, C_2) = \mathbf{D}$$

We denote by $\mathbb{D}(M)$ the concept connected to $M$ by default edge, and by $\mathbf{name}(C)$ the set of (synonymous) names associated with a concept.

## 3.2 Query Semantics

The **LOKI** knowledge-base is responsible for answering both labeling and discovery queries. For this paper, we adopt a simplified model where the answers to both types of queries depend solely on the highest score assigned to a column by a relevant matcher. This simplification carries two immediate limitations. First, query answering semantics do not attempt to combine inputs from multiple matchers — only the matcher with the highest score is used. However, we do discuss in Section 5 how expert feedback can be used to derive new matchers from existing ones. Second, this simplification precludes the use of contextual information about the table. For example, features like "the table describes medical data" could be used to significantly improve result quality. Although we do not address the use of context in this paper, we outline some potential approaches while discussing future work in Section 7.

**Labeling.** The input to a labeling query is a set of columns $T_C = \{A_1, \ldots, A_N\}$ and the knowledge-base $\langle\, \mathbb{M}, \mathbb{C}, \mathbb{E} \,\rangle$. The goal of the query is to identify a list of $N$ *distinct* concepts $C_1, \ldots, C_N$, one for each attribute, that best fit data in the columns. This can be phrased as a simple linear optimization problem that identifies $N$ matchers $M_1, \ldots, M_N$ that maximize the sum:

$$\sum_{i \in [1,N]} M_i(A_i) \qquad \textbf{subject to} \qquad \forall i \neq j \quad D(M_i) \neq D(M_j)$$

The concepts in the result are given by the resulting matcher's default edges $C_i = D(M_i)$.

**Discovery.** The input to a labeling query is a name $v$, a set of columns $T_C = \{A_1, \ldots, A_N\}$, and the knowledge-base $\langle\, \mathbb{M}, \mathbb{C}, \mathbb{E} \,\rangle$. The goal of the query is to identify the position of the attribute $i \in [0, N]$ most likely to fit the column name. The set of matchers that can be used to identify this column are given by:

$$\textbf{candidates}_v = \{\ M \mid E(M, C) \neq \perp \wedge v \in \textbf{name}(C)\ \}$$

The answer to the discovery query then is given by

$$\arg\max_{i \in [0,N]} \left( \max_{M \in \textbf{candidates}_v} M(A_i) \right)$$

## 3.3 Constructing a Knowledge Base

To instantiate specific match-quality functions, we merge three sources of information: Learned Heuristics, Expert Augmentations, and User Feedback. The first category, learned heuristics, models the content and distribution of typical instances of columns with a similar name. This distribution serves as a baseline match-quality function. The second category, expert augmentations, modifies the first, increasing or decreasing values based on expert-provided descriptions of what should and should not appear in columns with this name. The last category, user feedback, provides users with a way to confirm or override automated system choices, while also preserving these associations for future use.

# 4 TRAINING BY EXAMPLE

Matcher functions in **LOKI** come from three places: Learned Heuristics, Expert Augmentations, and User Feedback. In this Section, we address the first of these three. Because the learned heuristics are only the first step in an iterative process, our goal is not to develop an exhaustively correct matching scheme. Rather, we want something that is both simple and fast. Accordingly, throughout this section we make several simplifying assumptions.

## 4.1 Learning Matchers

Specific modeling techniques vary by data type, so our first step was to assess what types exist. We sampled a collection of 62 data sets from open data portals.

We then categorized the 458 columns in our sample into three broad types: (1) Numeric data, or any records consisting of digits, at most a single decimal point, and an optional exponent; (2) Enumerated types, based on an arbitrary threshold of atleast 50% distinct values in the column; and (3) Textual data, or anything else. By far, the dominant type was numeric, so the preliminary efforts we outline in this paper focus on modeling exclusively *numeric* data.

Regardless of the specific data type however, the overall learning process is similar.

The input to the process is a collection of labeled tabular data. Each column is split out and a type-specific modeling process is run for each column's data pair. The output of the modeling process is zero or more matchers that reliably model the column. If the process returns at least one matcher, we instantiate a new concept node with the column's name. We then add nodes for each matcher returned by the modeling process and edges to the newly instantiated concept node.

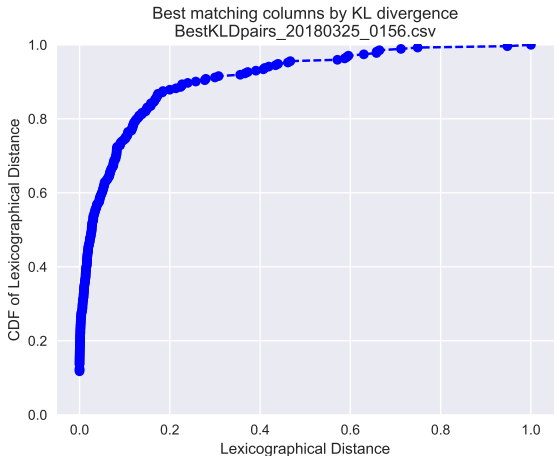



**Figure 3: CDF of Lexicographical Distance for the best match for each column.**

## 4.2 Creating Matchers for Numeric Data

We considered a range of options for modeling numeric data and settled on an approach based on numerical distributions. In comparison to more complex approaches like neural networks, this approach is simple, efficient, and well understood. Simply put, given a number of example column instances, we explore a range of numerical distributions and select the one with the best fit. Our preliminary implementation of **LOKI** explores three different distributions: Uniform, Normal, and Log-normal.

We used Kullback-Leibler (KL) divergence to measure how one distribution diverges from from another. A KL divergence of 0 suggess that distributions could be similar, if not the same. A KL divergence of 1 suggests that two distributions are very different. If knowledge of a specific problem domain suggests that similarity of data distributions could be better represented by another measure, KL divergence can be easily swapped with for it in our system.

Lexicographical Distance indicates the difference in the column labels of two distributions. We used two measures of Lexicographical Distance - Levenstein Distance and NGram Distance. Section 5 talks about performance of both of the measures in estimating Lexicographical Distance on our dataset. A Lexicographical Distance value of 0 suggest that the two columns labels are exactly the same, whereas, a value of 1 suggests that the column labels are very different.

Learned heuristics require an assumption to build baseline match-quality functions. We proceed with the initial assumption that column pairs with *similar* data would be labelled *similarly*. The value of KL divergence of two columns is inversely proportional to *similarity* of their data distributions. The value of Lexicographical distance of two columns is inversely proportional to *similarity* of their column labels. We have verified this assumption by plotting the Cummulative Distribution Function of Lexicographical Distance of column pairs with minimum KL Divergence in Figure 3. 80% of column pairs with most similar data have their labels which are at most 0.129 lexical distance apart. This indicates that even without human intervention, columns pairs with the most similar data are already labelled similarly.

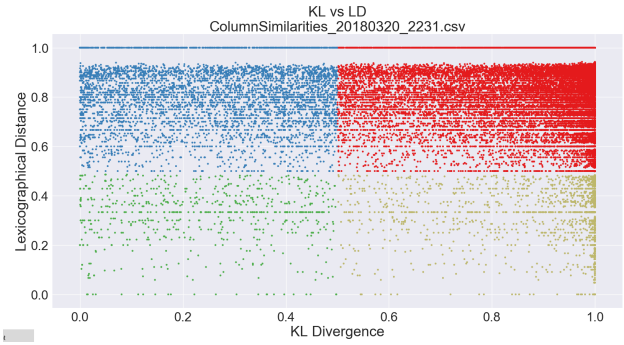



**Figure 4: Knowledge-base Editor: Showing Match Conflicts**

For each distribution, we find the parameters $(\ell, h, \mu, \sigma, a)$ that minimize the root-mean-squared (RMS) error between the theoritical and empirical distributions.

$$\mathbb{U}(\ell, h) \quad \mathbb{N}(\mu, \sigma) \quad \mathbb{L}ognorm(\mu*, \sigma*)$$

Parameter estimation is performed by maximizing a log-likelihood function. Log-likelihood estimation is a widely used and robust technique for estimating the likelihood of a statistical distribution and its parameters being representative of the empirical data distribution. We implemented this using the SciPy package in Python.

We then select the one distribution with the lowest overall RMS error. The resulting match-quality function is the probability of the column values being a representative sample drawn from this distribution.

## 5 EXPERT REFINEMENT

While example data sets provide a good starting point for the **LOKI** knowledge-base, we anticipate that — at least initially — the training data will be too sparse to provide high quality matching suggestions. To mitigate the effects of sparsity, **LOKI** relies on domain experts to curate its knowledge-base, refine existing matchers and define new matching heuristics. This is intended to be an ongoing process, with incremental feedback from experts and users continuously refining the knowledge-base. In this section we outline the design of an interface that streamlines knowledge-base refinement, starting from a knowledge-base trained on example data. The central elements of this interface are (1) Visualizing the current quality of the knowledge base; (2) Identifying problems with name/match-quality function pairs; (3) Refining records in the knowledge base by removing or merging existing data, or adding expert knowledge.

### 5.1 Refinement Interface

The initial goal of the **LOKI** editor is to help domain experts quickly identify and resolve errors and redundancies in a preliminary knowlege-base freshly trained on new example data. The entry point into the refinement process is the **LOKI** editor's match-conflict explorer

**Match Conflict Explorer.** This view, illustrated in Figure 4, is a scatter plot with each point corresponding to one pair of concepts. A point's x- and y-coordinates express, respectively, how different the concepts and their related matchers are. In both cases, a value of 0 indicates that the concepts (resp., the corresponding matchers) are identical, while a value of 1 indicates that the concepts (matchers)

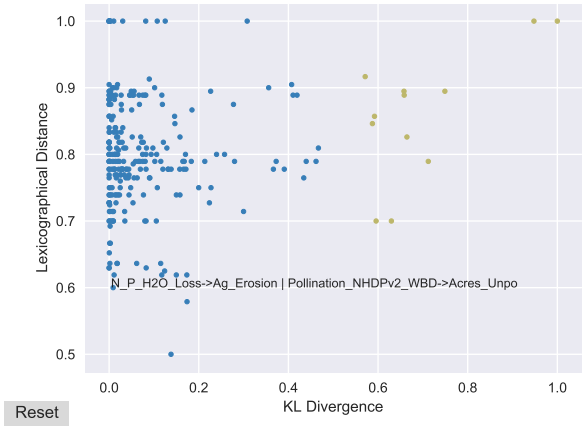



**Figure 5: Pair view in scatter plot**

are completely different. We refer to these as concept- and match-differences. In this preliminary version of the system, we define concept-difference to be the lowest Levenshtein distance between any two names associated with each of the concepts. We define match difference as the lowest difference between any two matchers associated with a concept, with inter-matcher differences defined based on concept types. For pairs of numerical distributions, this value is presently defined as the KL-Divergence between them.

The match conflict explorer helps identify overlapping or erroneous matchers, as well as duplicate or overlapping concepts. Specifically, the view is divided into four quadrants. The lower-left, shown in green indicates high concept-similarity and highly overlapping matchers (true positives). The upper-right, shown in red indicates the low concept-similarity and low matcher similarity (true negatives). The upper-left, shown in blue identifies distinct concepts with similar matchers (potential false positives). The lower-right, shown in yellow identifies potentially similar concepts that have different matchers (potential false negatives).

**Concept Pair View.** To resolve potential conflicts displayed in the explorer, users can select regions with a simple marquee tool or click on individual concepts (zoom is available if desired). The selected concepts are displayed, along with their corresponding matchers as illustrated in Figure 5.

### 5.2 Understanding Refinement in Practice

As experts identify errors or redundancies in the knowledge base, they will need to rectify them accordingly. To better understand their needs during this process, we selected 200 concept pairs at random from the potential false-positives and potential false-negatives. We inspected each pair of these concepts to determine a root cause for such a matching. Then, we grouped these root causes into seven categories. The distributions of root causes of the potential false negatives and false positives are shown in Figures 6 and 7, respectively. We now discuss each category in detail, along with an assessment of remedial actions available for resolving them.

*5.2.1 Need a better model for training by examples.*
For 9 pairs in the lower right quadrant and 69 pairs in the upper left quadrant, the best fit matchers did not adequately describe the data distribution. The training phase categorizes data by either uniform,

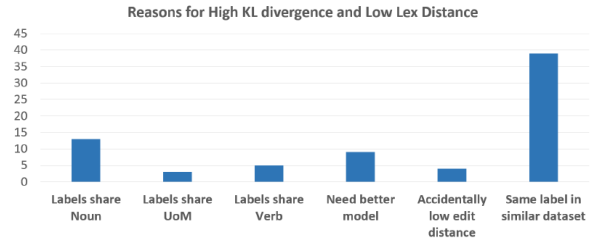



**Figure 6: Root causes of potential false negatives.**

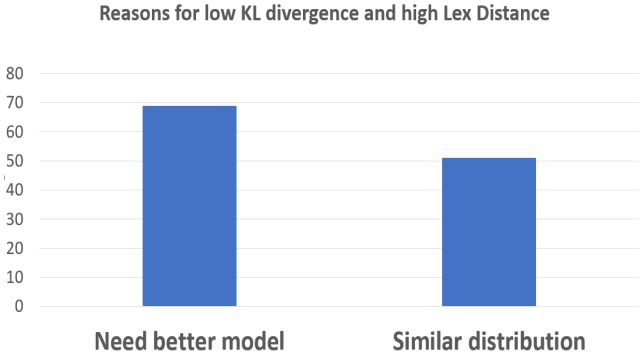
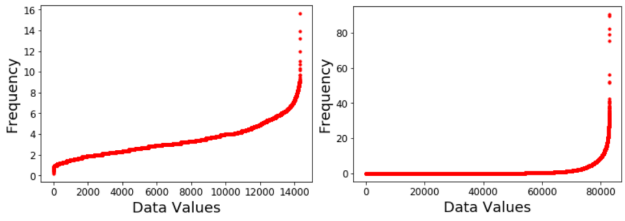



**Figure 7: Root causes of potential false positives.**




**(a) `BIGGA_AVG` (bimodal)** **(b) `PctPasture_slope9` (zipfian)**

**Figure 8: Example CDFs of poorly fitted matchers**

normal or log-normal distributions. However, we encountered distributions which can be described better using other distributions like zipfian and other power law distributions. Two examples are shown in Figure 8, following bimodal and zipfian distributions respectively, neither of which were supported by our preliminary implementation.

**Resolution**: Although a more robust learning process might address this issue ( by supporting more standard distributions like zipfian), another way to definitively address this issue is by allowing experts to manually define custom distributions (e.g., using splines).

*5.2.2 Same column name in a different context.*
33 points, nearly half of the points sampled from among the potential false negatives, used the same name but in different context. For example, among the datasets used for this case study, we had two similar datasets in biodiversity domain. Both datasets consisted of data about flora and fauna, but from two different regions.

**Resolution**: Depending on context, a domain expert may wish to unify both concepts or keep them separate. In the former case, the expert needs to be able to merge concept nodes in the knowledge-base together. In the latter case, the expert needs to be able to override the concept-difference to mark them as true negatives.

*5.2.3 Column names share measuring units, nouns, or verbs.*
In the case of 21 out of 79 potential false negatives, the limiting factor was our choice of similarity measure. Common terminology not relevant to the meaning of the concept resulted in low lexical distances. Specific examples we encountered were: (1) Common measuring units like `FRUITHECTARES` or `VEGHECTARES` (3 pairs) (2) Common generic nouns like `BG_Demand` or `MB_Demand` (13 pairs) (3) Common verbs or statistical measures like `AVG`, `MEAN` or `MAX` (5 pairs).

**Resolution**: As before, depending on context, a domain expert may wish to unify the underlying concepts (e.g., in the case of units like `HECTARES`), or not. Similarly, this means that the expert needs to be able to merge concept nodes or override the concept difference measure. However, in the case of this error, we can also pre-filter many of these cases by deleting common stop-words before computing the lexical distance (e.g., `AVG`).

*5.2.4 Accidentally Levenshtein distance.*
In 4 remaining cases, the Levenshtein distance incorrectly identifies two column names as similar. For example the column names `WTFL_AVG` and `WET_AG` are conceptually distinct, despite their low Levenshtein distances. Using NGram distance might solve this problem.

**Resolution**: This problem might be fixed by a more robust concept distance measure, such as one that relies on an ontology like YAGO [3]. However, any heuristic measure might still incorrectly label two distinct concepts as similar. Once again, the domain expert must be able to overide the concept-distance function.

*5.2.5 Distinct names with similar distributions.*
Out of 120 points analyzed from the upper left quadrant, 51 had similar distributions. Certain data distributions were incredibly common. For example, there were (unsurprisingly) a large number of uniformly distributed integer values starting from 0 which were Index attributes. These often had distinct names.

**Resolution**: As in many of the previous cases, based on context the expert may decide that this is a true positive, or may decide that the concepts are indeed distinct. Similarly, the concepts may need to be merged, or not. However, this case presents one additional challenges: it indicates matchers linked to distinct concepts that are liable to have similarly high match qualities on the same data — conflicting matchers. Hence, in both cases, the expert also needs to merge the conflicting matchers.

*5.2.6 True positives.*
Although not the result of an actual problem, true positives indicate the presence of duplicate concepts, and consequently conflicting matchers. As before, both the concepts and matchers need to be merged.

## 5.3 Editor Interface

In addition to the match-conflict view, the **LOKI** editor provides a textual interface to search the knowledge-base for both concepts (by name), and matchers (by matcher-specific properties like the shape parameters of a distribution matcher). As previously noted, selecting pairs of concepts in the match-conflict view shows the pair of concepts and their associated matchers (as the result of a query).

Specifically, to resolve the data issues described above, the editor interface needs to allow experts to: (1) Manually define new data distributions, (2) Merge Concept Nodes, (3) Merge Match Nodes, and (4) Override concept difference by forcing concepts to be distinct and defining stop-words

**Editor Commands.** In addition to directly querying the knowledge-base, the editor provides commands for editing it:

`EDIT [concept]`: Allows the expert to modify the names and matchers associated with a concept

`EDIT [matcher]`: Allows the expert to modify the shape parameters and concepts of a matcher, as well as the matcher's default concept.

`MERGE [concept] [concept]`: Combines two concepts, creating a single concept by unioning their associated names and directing the associated matchers of both concepts to the same node.

`MERGE [matcher] OR [matcher]`: Instantiates a new matcher that emits the highest match quality detected by either matcher. Concepts linked to the two matchers are migrated to the new matcher and the user is acked to pick one as a default for the new matcher.

`REPLACE [matcher] WITH [matcher]`: Deletes the former matcher and links the latter matcher to all of its concepts. The latter matcher's default concept is preserved.

## 6 RELATED WORK

Data distributions have been used for similar purposes in other work. For example GestureQuery [7] uses data similarity between two attributes to select candidate attributes for an equi-join. To maximize the join arity, the system counts the number of times each value from one attribute appears in the other and a histogram is constructed from the counts for all of the values.

Wrangler [6] and Potter's wheel [9] detect data domains through inclusion functions (e.g. regular expressions). Wrangler in particular infers the data type of a column and highlights errors based on inconsistent data types. Wrangler also has several operators like split and unfold that create new columns. The split operator decomposes composite data values into component distributions. The unfold operator reverses a table pivot, collapsing data laid out as key-value pairs into columns. A useful application of the **LOKI** knowledge-base that we hope to explore in future work is using it to detect opportunities for applying such operators.

An orthogonal approach to modeling and matching columns is to use ontologies, which express entities, facts, relations between facts and properties of relations Ontologies like Yago [3] could be used to identify semantic properties that relate columns.

A data summary called the data describer is used in [8]. The data types, correlations and distributions of the attributes in a private dataset are listed. Each attribute is categorized into either numerical or non-numerical. If non-numerical attribute cannot be parsed as datetime then it is considered to be a string.

Data describer takes in a CSV file and infers the data types and domains. The attribute datatypes are parsed as numerical, datetime or string. We are inferring the datatypes as well. When run in correlated attribute mode, data describer provides corelation between attributes. We could use this functionality in **LOKI**.

PADS [4] helps users to understand the layout and meaning of data by designing syntactic descriptions of the data. Based on the syntax, accumulators track the number of good values, the number of bad values, and the distribution of legal values. This technique could be used in **LOKI** to help capture expert knowledge.

There is an increasing number of datasets in which well-structured attributes (with or without a name) can be identified, each containing a set of values called a domain. There is lack of schema description in most of the datasets. LSH Ensemble is used in [11] to find domains that maximally contain a query dataset, which can help to find datasets that best augment a given set of data.

## 7 FUTURE WORK

We plan several extensions as future work focussed on building and refining a knowledge-base for storing column-naming heuristics.

**Use of contextual information.** Contextual information such as units, data domains, and links between concepts and ontologies could be used to augment the **LOKI** knowledge-base. For example, we could use a network of semantic relations such as BabelNet strengthen data models for training the knowledge-base as well as providing curation recommendations to experts. [2]

**Recommendation of columns for query.** Concepts in the knowledge-base could be used to recommend columns that *could be* in a query based on the columns that are already present.

***Smarter* Matchers.** We plan to develop matchers which regocognize a wider spectrum of data. For example, regular expression matchers could be used to detect geolocation data. Matchers which can identify *synonymous* labels could help experts in the curation process.

**Other applications of context.** Once discovered, contextual information like units or data domains can be used in other ways. For example, if a column is identified as having a particular type of unit (e.g., 'meters'), this knowledge could be used to warn users trying to merge it with an incompatible unit (e.g., 'inches').

## ACKNOWLEDGMENTS

This work was supported by NSF Awards IIS-1750460, ACI-1640864 and by a gift from Oracle. The conclusions and opinions in this work are solely those of the authors and do not represent the views of the National Science Foundation or Oracle.

## REFERENCES

[1] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. 2012. NoDB: efficient query execution on raw data files. In *SIGMOD Conference*. ACM, 241–252.
[2] Babelscape. 2018. BabelNet semantic knowledge base. (2018). http://babelnet.org
[3] MS Fabian, K Gjergji, WEIKUM Gerhard, et al. 2007. Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *16th International World Wide Web Conference, WWW*. 697–706.
[4] Kathleen Fisher and Robert Gruber. 2005. PADS: a domain-specific language for processing ad hoc data. In *ACM Sigplan Notices*, Vol. 40. ACM, 295–304.
[5] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. 2007. Database Cracking. In *CIDR*. www.cidrdb.org, 68–78.
[6] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3363–3372.
[7] Arnab Nandi, Lilong Jiang, and Michael Mandel. 2013. Gestural query specification. *Proceedings of the VLDB Endowment* 7, 4 (2013), 289–300.

[8] Haoyue Ping, Julia Stoyanovich, and Bill Howe. 2017. DataSynthesizer: Privacy-preserving synthetic datasets. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. ACM, 42.
[9] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *VLDB*, Vol. 1. 381–390.
[10] Ying Yang, Niccolo Meneghetti, Ronny Fehling, Zhen Hua Liu, and Oliver Kennedy. 2015. Lenses: An On-Demand Approach to ETL. *Proc. VLDB Endow.* 8, 12 (2015).
[11] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. LSH ensemble: Internet-scale domain search. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1185–1196.