PCIean: Probabilistic Scripts for Automating Common-Sense Data Cleaning

Anonymous Author(s)
Affiliation
Address
email

Abstract

A key design challenge in implementing data-cleaning systems is deciding how users should encode domain knowledge relevant to the cleaning task; an ideal system would empower users to flexibly and concisely communicate all sorts of relevant information, then use it — in addition to any patterns learned from the data table itself — to improve accuracy during cleaning. This makes data cleaning an ideal setting in which to study the interplay between machine learning and knowledge representation. Inspired by a recently introduced framework for viewing data cleaning as probabilistic inference, we present PClean, which detects errors, proposes corrections, and imputes missing values in unclean tabular datasets guided by short, declarative, user-written scripts in a domain-specific probabilistic programming language. We show that these scripts can encode various kinds of knowledge useful for data cleaning, and describe a hybrid sequential Monte Carlo / EM algorithm for learning patterns from the data, detecting and correcting errors, and imputing missing values guided by these probabilistic scripts. We report preliminary results of two experiments, which suggest that PClean's scripting language can encode assumptions that recover the behavior of existing systems on some datasets. And through an extended example, we show how PClean enables common-sense reasoning about data without any training on labeled examples.

1 Introduction

Datasets are often messy and incomplete, sprinkled with typos, `NULL` values, numbers reported in the wrong units, and other errors. Solving a data-cleaning task — that is, imputing missing values, detecting errors, and suggesting corrections (we do not consider record linkage [6] here) — requires both learning regularities in the data (so that anomalies can be flagged), and taking advantage of domain knowledge that may be impossible to glean from the messy data alone. This makes automated data-cleaning [1, 2, 5, 7] an interesting setting in which to explore the interplay of knowledge representation (how can the user convey symbolic knowledge to the system?) and machine learning (how can the system uncover patterns in the data that make the cleaning task easier?).

We follow the recently introduced *probabilistic unclean databases* framework [8] in framing the data-cleaning task as a probabilistic inference problem. To use this framework, one first specifies a probability distribution $p(I)$ over clean database instances I, and a conditional distribution $p(J \mid I)$ over corrupted versions J of a particular instance I. The cleaning task is then reduced to posterior inference: we wish to find the posterior distribution $p(I \mid J)$ over likely clean database instances, given an observed corrupted one.

Where does the probabilistic model come from? Automated data cleaning systems like HoloClean [7] and BayesDB [5, 9] combine user-provided qualitative constraints with automatically learned properties of the data to synthesize a probabilistic model, then perform inference in that model to accomplish data-cleaning tasks (imputation, in the case of BayesDB, and error correction, for HoloClean). But each system accepts only a limited class of qualitative constraints, and is capable of synthesizing only a limited class of probabilistic models. For example, BayesDB lets users specify arbitrary independence relationships between variables, which it takes into account in generating a multiple-mixture model. In HoloClean, users express logical "denial constraints" and "matching constraints," which are softened and stitched together into a factor graph with learned parameters. In neither system can users express arbitrary probabilistic information, like "cities appear in the database in proportion to their populations," or "height and weight are positively correlated."

In this work, we explore a new approach to combining learning with flexibly encoded domain knowledge. Our core insight is that *probabilistic programming languages*, in which it is easy to express a broad range of probabilistic models concisely, can be used to vastly expand the kinds of domain knowledge users can express to a data-cleaning system. We propose a system, PClean, in which users encode their domain knowledge in the form of a short *probabilistic script*, which can be thought of as a stochastic simulator of fake (clean and dirty) data. In the program, users describe a hypothetical process by which data might be generated and made dirty; PClean then does inference based on this program to reason about likely *clean* values of each cell, given the dirty values of every cell in the row. These models may contain unknown parameters, which PClean can estimate by learning from the data. In the sections that follow, we give an overview of the approach with a simplified example, describe our implementation, and report on some early experimental results.

2 Overview

2.1 An example: messy apartment listings

As a simplified example to motivate our approach, consider the dataset in Figure 1a, which could have come from a bare-bones website for apartment listings.

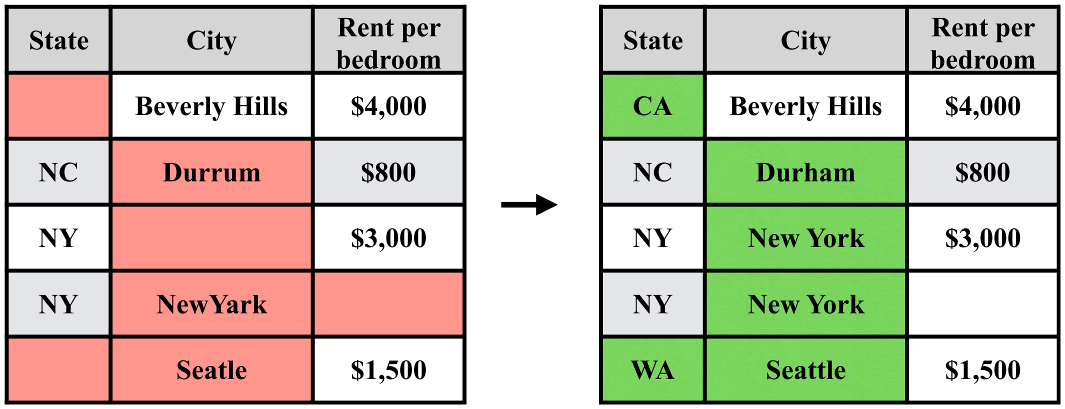
The errors in the dataset are not difficult to spot, and many people would have little trouble correcting them. But expressing the rules for doing so programmatically might be difficult, as the task requires a combination of domain knowledge and common sense. To know that "NewYark" should be corrected to "New York," and not "Newark," we must know that cities with small populations (Newark, NY has 9,000 residents) show up infrequently on the site. There are at least six cities called Beverly Hills in the United States, none of which have particularly large populations, but only in California would we expect an apartment to cost $4,000 a room. And in the middle row, although we can't know for sure what city this user came from, both price and population might lead us to decide that "New York" is the most likely answer: it's expensive to live in New York (so the $3,000 price point is well-explained), and millions of people do (so, all else equal, we'd guess a user from New York state was most likely from New York City). In other words, when cleaning the dataset, we weigh various factors to determine the plausibility of a particular story for how the dirty data came to be: did someone come from Seale, AL and add a 't' to get 'Seatle', or did a user from Seattle forget to type the second 't'? PClean lets us equip our machines with the kind of domain knowledge that enable them to perform these calculations automatically.

2.2 Encoding knowledge with PClean

In PClean, we encode this sort of knowledge by writing a short *probabilistic script* (Figure 1b). The PClean scripting language is embedded in Julia; PClean scripts are declared with `@pclean`, and can be used to clean a dataset by calling the `clean` library function. A PClean script encodes domain knowledge in the form of a generative probabilistic model over the clean and dirty values of each cell. The user writes code that randomly generates fictional rows of data, clean and dirty, inducing a prior distribution over the data itself, and a model of the ways in which it might become unclean.

In our example, the script encodes that:

1. States appear proportionally to their populations: line 2 declares the presence of a "State" column, and generates a value for it by sampling a random state, according to population counts from a knowledge base (`state_pops`). We store the result in the variable `state`.



State	City	Rent per bedroom
	Beverly Hills	$4,000
NC	Durrum	$800
NY		$3,000
NY	NewYark	
	Seatle	$1,500

→

State	City	Rent per bedroom
CA	Beverly Hills	$4,000
NC	Durham	$800
NY	New York	$3,000
NY	New York	
WA	Seattle	$1,500

(a) An excerpt from a messy apartment listings dataset. A human familiar with the US has no trouble fixing the errors, but the rules for doing so can be difficult to describe programmatically: why Beverly Hills, CA and not Beverly Hills, MO? Why Seattle, WA and not Seale, AL? Why New York in row 3?

```
1 @pclean rent_row_generator begin
2   @column "State" state = choose_proportionally(states, state_pops)
3   @column "City" city = choose_proportionally(cities[state], city_pops[state])
4   @dirty "City" = add_typos(city)
5   @column "Rent" = add_noise(mean_rents[city, state], 100)
6 end
7
8 cleaned_table = clean(messy_table, rent_row_generator)
```

(b) A PClean script encodes assumptions about the data and the ways in which it may be unclean. Here, we assume access to trusted knowledge bases (in red) with population counts and average rents of each US city.

```
1 @pclean rent_row_generator begin
2   @column "State" state = choose_proportionally(states, _)
3   @column "City" city = choose_proportionally(cities[state], _[state])
4   @dirty "City" = add_typos(city)
5   @column "Rent" = add_noise(_[city, state], _)
6 end
7
8 fit!(messy_table, rent_row_generator) # modifies rent_row_generator
9 cleaned_table = clean(messy_table, rent_row_generator)
```

(c) When knowledge-bases aren't available, underscores can be used to introduce learned parameters. PClean will estimate these parameters from the data.

Figure 1: Cleaning a dataset of apartment listings with PClean.

2. Possible cities depend on the state, and more populous cities are more likely to appear: line 3 introduces a "City" column, and samples a city from a state-specific list of cities `cities[state]`. We store the result in the local variable `city`.
3. The city field may, rarely, contain typos: line 4 introduces a *dirtying transformation* of the "City" field, using the `add_typos` function to randomly insert zero or more typos into a string. The distribution over number of typos is geometric, with a 90% chance of no typos.
4. The rent per bedroom depends stochastically on the city and state: line 5 samples a rent by adding noise to a known mean rent of the sampled city and state.

This knowledge makes it possible to clean the data table using probabilistic inference. For each empty cell, we ask what values make the entire row most plausible to have been generated by the script. This means that, for example, when imputing missing states and cities, PClean will never yield a (city, state) pair that is inconsistent, as such pairs have zero probability of being generated by the script. It also means populations, and rents will both be taken into account when trying to impute a missing city.

When a cell is non-empty, it is assumed to be clean, unless the column it appears in has a `@dirty` statement somewhere in the PClean script. In that case, the observed value for the cell is taken to be the output of the dirtying transformation, and system then tries to infer what the original sample

Table 1: Data Cleaning Performance on the Hospital Dataset.

	PClean	HoloClean	Holistic	KATARA	SCARE
Precision	1.0	1.0	0.517	0.983	0.667
Recall	0.713	0.713	0.376	0.235	0.534
F1 Score	0.832	0.832	0.435	0.379	0.593

must have been — again, in order to make the whole row (including the observed dirty value) most plausible. Weighing all of the relevant factors allows the system to disambiguate between multiple possible clean values for a cell.

2.3 Learning parameters

Of course, in many domains, the user will not necessarily have access to knowledge bases like `state_pops` and `mean_rents`. As such, PClean also allows users to introduce *learned parameters* with the underscore syntax (Figure 1c). PClean will estimate all parameters from the data table itself, using a Monte Carlo Expectation-Maximization algorithm (see Section 3). Parameters can be *indexed* by arbitrary expressions, indicating that PClean should learn a different value of the parameter for each value taht the indexing expression takes on; this is how we express, e.g., that each city and state has a different average rent.

3 Implementation

PClean works by compiling the user's script into a probabilistic model in a general-purpose probabilistic programming language, Gen [3]. The model is parameterized by θ, a vector of all parameters arising from underscores in the PClean script, and specifies a joint distribution over *clean values* x_c (where c is the name of a column), one for each `@column` statement in the PClean script, and *dirty values* y_c, one for each `@dirty` statement. (PClean also supports latent variables, but we do not discuss them here.)

Cleaning begins by initializing any parameters θ to some default values θ_0. Given fixed parameter values θ, we can clean each row indepenently: we use Gen's inference library, which provides a number of building blocks for popular approximate inference algorithms, to implement a sequential Monte Carlo algorithm for approximately sampling from the posterior $p(x_c^{\text{unobserved}} \mid y_c, x_c^{\text{observed}})$ induced by the data in an individual row. Given such a sample, we can use the complete data $(x_c, y_c)_i$ for all rows of the table to estimate new maximum-likelihood values of the parameters θ. Repeating these two steps in a loop gives a Monte Carlo version of an EM algorithm, where the E-step consists of cleaning the data table using the current parameters, and the M-step consists of re-estimating the parameters given a hypothesized clean table.

4 Preliminary evaluation

We have yet to do a full evaluation of PClean on real-world datasets. However, we report the results of two experiments here, designed to better explain connections to some existing work:

1. In Table 1, we show that on the Hospital benchmark dataset, PClean is expressive enough to encode the sorts of qualitative constraints that Holoclean uses, and thus to recover the exact same results (which are superior to those of other systems, reproduced here from [7]).
2. We also ran an experiment on the Iris dataset from the machine learning literature [4], to show that PClean can be used to repurpose existing machine learning tools for data cleaning. We removed 75 cells at random from the Iris dataset's "species" column, and trained a CrossCat model on the data using BayesDB [5]. We then encoded the learned model directly as a PClean script [10]. With this learned model, PClean imputes 71/75 (93%) of the missing values correctly, achieving much better performance than baseline methods like those used by ImputeDB [1], which imputes only 21/75 missing values correctly.

References

[1] Cambronero, J., Feser, J. K., Smith, M. J., and Madden, S. Query optimization for dynamic imputation. *Proc. VLDB Endow. 10*, 11 (Aug. 2017), 1310–1321.

[2] Chu, X., Morcos, J., Ilyas, I. F., Ouzzani, M., Papotti, P., Tang, N., and Ye, Y. Katara: Reliable data cleaning with knowledge bases and crowdsourcing. *Proc. VLDB Endow. 8*, 12 (Aug. 2015), 1952–1955.

[3] Cusumano-Towner, M. F., Saad, F. A., Lew, A. K., and Mansinghka, V. K. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2019), ACM, pp. 221–236.

[4] Fisher, R. A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics 7*, 7 (1936), 179–188.

[5] Mansinghka, V. K., Tibbetts, R., Baxter, J., Shafto, P., and Eaves, B. Bayesdb: A probabilistic programming system for querying the probable implications of data. *CoRR abs/1512.05006* (2015).

[6] Pasula, H., Marthi, B., Milch, B., Russell, S. J., and Shpitser, I. Identity uncertainty and citation matching. In *Advances in neural information processing systems* (2003), pp. 1425–1432.

[7] Rekatsinas, T., Chu, X., Ilyas, I. F., and Ré, C. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow. 10*, 11 (Aug. 2017), 1190–1201.

[8] Sa, C. D., Ilyas, I. F., Kimelfeld, B., Ré, C., and Rekatsinas, T. A formal framework for probabilistic unclean databases. *CoRR abs/1801.06750* (2018).

[9] Saad, F., and Mansinghka, V. A probabilistic programming approach to probabilistic data analysis. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)* (Barcelona, Spain, 2016), pp. 2011–2019.

[10] Saad, F. A., Cusumano-Towner, M. F., Schaechtle, U., Rinard, M. C., and Mansinghka, V. K. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM on Programming Languages 3*, POPL (2019), 37.