Botton-up Standardization for Data Preparation

Eugenie Lai, Yuze Lou, Brit Youngmann, Michael Cafarella

Background & Motivation

Data preparation has been seen as "janitor work" yet essential in data-to-insight pipelines. Now we have **more access to more data**, but **challenges** including:

- People have limited technical and domain expertise
- People who have the skills keep writing one-off programs

Core observation:

- Scripts **embed thought process** and knowledge
- There is **a latent space** of data preprocessing steps in script corpus

Recently, automating data preparation scripts improves many aspects of the pipeline, including **data quality**, **research reproducibility**, and user productivity. We propose an approach to **automatically standardize scripts**.

Problem Definition

Informally, our goal is to **standardize** user input scripts s^* **using a sequence of transformations** informed by the **script corpus** S.

Specifically, the output script $\hat{s}$ should (1) leverage the domain knowledge embedded in the script collection; (2) be easily consumed by the user

We use relative entropy to define a score for our standardization goal.

$$RE(P||Q) = \sum_{x \in X} P(x) log(\frac{P(x)}{Q(x)})$$

Our objective is to minimize relative entropy, where

- P represents the distribution of data preprocessing steps in the **input** s^*
- Q represents the the distribution of data preprocessing steps in the **collective knowledge** S

Intuition: (1) The latent space of dataset-specific data preprocessing steps in the script corpus manifests itself in the distribution of its data preprocessing steps; (2) with the optimal sequence of transformations, s^* becomes closer to S with control over the semantics.

We use table jaccard to measure how much script semantics have changed.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}, \text{ where } A, B \text{ are sets.}$$

Related Work

Roughly three topics:

- **Predictive data transformation**: step/pipeline recommendations (tldr: by-example approach, program synthesis)
- **Program synthesis** for data preparation: automatically generates code for given user intent (e.g., Copilot, AutoPandas)
- Exploratory data analysis (**EDA**) recommendation

Contributions

New things we do:

- We define a **new problem** of **data preparation program standardization**
- We propose a novel approach to transform programs while allowing **control over script semantics**
- We empirically show that our approach mostly outperforms learned methods (e.g., Copilot, Codex)

Framework & Data Model

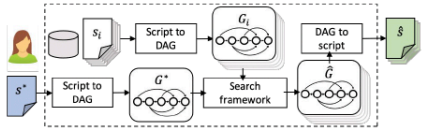
At high level, our approach

(1) uses collection $S = \{s_0, s_1, s_2, \dots, s_n\}$, transform each into its DAG representation, and obtain a latent space of data preprocessing steps $Q(x)$

(2) takes an input s^* and get its DAG representation G^*

(3) uses the search-based framework with $\boldsymbol{G} = \{G_0, G_1, G_2, \dots, G_n\}$ and G^* as input to find a sequence of transformations $\hat{F}$ that transforms G^* to $\hat{G}$

(4) returns $\hat{s}$ as the script representation of $\hat{G}$



Choice of script-to-AST and AST-to-DAG:

Improving script quality requires a way to take it apart and transform it. The challenge is that the improved script should be free from execution errors. AST already has many desired properties.

- We construct DAG $G = (V, E)$ from AST for implicit flows
- V: PL construct (e.g., function call); E: data flow

Algorithm 1: A meta-level greedy search framework

Input: $\mathcal{S}$ (Script collection), s^* (input script), K (budget), θ (data correctness threshold), α (early checking switch)
Output: $\hat{s}$ (A standardized version of s^*)

```
C = {s*};                                   /* Set of all candidates. */
while not StoppingCriteria (C) do
    C' = ∅;                                 /* Set of new candidates. */
    foreach s ∈ C do
        F = GetSteps (s, S) ; /* A ranked set of next steps
          based on RE score. */
        C' = GetTopKBeams (C', s', K, α, F);
    C = C'
C = VerifyConstraints (C, θ);
ŝ = GetTopCandidate (C);
return ŝ;
```

Methods & Optimization

Challenges:

- Huge **search space**, due to (1) possible sequences grow exponentially as the steps increase; (2) # of unique V is large
- Execution **errors** and data **semantics** are **difficult to estimate in the process**. Since we only care about them at the end, they are allowed to be off the given thresholds during the process, which poses challenges on pruning the candidate sequences

We propose two **GetTopKBeams()**.

- **Difference**: diversity measure
- Define heuristics to capture operator funcational semilarity

Algorithm 2: GetTopKBeams

Input: C (Current candidates), s (a script), K (budget), α (early checking switch), $\mathcal{F}$ (a ranked set of next steps)
Output: C' (Updated candidates)

```
valid = True;
C' = C;
foreach f ∈ F do
    s' = ApplyStep (s, f);
    if α then valid = CheckIfExecutes (s');
    if Score (s') < min_{s∈C} Score(s) or |C| ≤ K then
        if valid then C' = UpdateKBeams (C', s', K);
return C';
```

Algorithm 3: GetDiverseTopKBeams

Input: C (Current candidates), s (a script), K (budget), α (early checking switch), $\mathcal{F}$ (a ranked set of next steps)
Output: C' (Updated candidates)

```
C' = C;
F_M = ClusterSteps (F);
foreach F_{i∈M} do
    C' = GetTopKBeams (C', s', K/M, α, F);
return C';
```

Experiments & Discussion

Setup: Two datasets from Kaggle; for each of ~50 scripts, use one as s^*, the rest as S; repeat for transformation seq length in [2, 4, 8, 16]; beam in [1, 3]; data threshold in [1.0, .95, .8, .7].

% improvement = (start_score - end_score) / start_score