Learning Data Transformations with Minimal User Effort

Minh Pham
Information Sciences Institute
University of Southern California
minhpham@usc.edu

Craig A. Knoblock
Information Sciences Institute
University of Southern California
knoblock@isi.edu

Jay Pujara
Information Sciences Institute
University of Southern California
jpujara@isi.edu

***Abstract*—Data collected from heterogeneous sources often have inconsistencies in data format and thus require transformation before the data can be used. A major issue of existing approaches is their dependency on parallel input-output data to learn the transformations. However, parallel data are not always available, and annotation requires excessive human interaction because of format diversity. Therefore, these approaches have limitations when applied to large-scale real-world problems. To address this issue, we introduce UDATA, a novel unsupervised system for non-parallel data transformation. Because the transforming data usually share common syntactic patterns, UDATA discovers common syntactic patterns from input/output examples and synthesizes the transformations between the patterns. Moreover, in UDATA, transformation results are verified by an active learning model and ambiguous results are reported to users for labeling. UDATA achieves accuracy close to other state-of-the-art supervised systems without the need for any labeled data.**

I. Introduction

Data normalization is always a challenge in applications that process and exploit data, especially when data are collected from multiple sources. Raw data are usually stored in many different formats. For instance, date/time data can be represented in multiple formats such as "dd/mm/yyyy," "mm/dd/yyyy," and "MM, dd, yy"; or people's names can be written in many ways such as "$\langle first\_name \rangle$ $\langle last\_name \rangle$" or "$\langle last\_name \rangle$, $\langle first\_name \rangle$". Therefore, raw data often need to be transformed to a standard format, as shown in Table I. If we can fully infer these transformations automatically, the problem of data integration and related tasks, such as data analysis, can be resolved more easily. As a result, applications on noisy data domains can be developed faster and produce more accurate results. Traditionally, users need to write programs to transform data, which are also hard to maintain because of unforeseen formats in new data sources.

Existing approaches — In recent years, much research has been undertaken to solve the data transformation problem with less human interaction. However, previous methods such as programming-by-example (PBE) or interactive data cleaning still depend mainly on human interaction to learn the correct transformations. For instance, interactive cleaning systems [16], [20] require users to specify transformation rules based on their suggestions. On the other hand, PBE [8], [28], [22], which is the state-of-the-art approach, achieves high transformation accuracy by relying on parallel input/output examples provided by users. Because of the dependency on human input, interactive data cleaning and PBE are rarely scalable when the volume of data and the number of data sources increase. These semi-supervised systems have two main difficulties: 1) handling format diversity; and 2) result verification. Source attributes can contain multiple data formats and thus require annotation for each format in order to learn the transformations. Moreover, the transformation result needs to be validated to guarantee the accuracy before being used in other applications. The process of providing input examples for each different format and verifying the transformed results eventually slows with thousands or millions of records.

TABLE I: Normalizing raw data from different formats

Raw data	Normalized data
Messi; Lionel	Lionel Messi
Paul Pogba	Paul Pogba
Sergio R. Garcia	Sergio Garcia

Proposed solution — In this paper we introduce UDATA, a novel unsupervised system to solve the data transformation problem with minimal human interaction. UDATA takes as input a set of string values to be transformed, along with output examples, in a target format. However, unlike prior work, UDATA does not require output examples to have any alignment with the input data. Examples of parallel and nonparallel data are shown in Figure 1. Since target output examples have no connection with the transforming data, they can be specified before data collection: thus, no human interaction is required to learn the transformation programs. In this paper, we call the problem of nonparallel data transformation *unsupervised* data transformation, in contrast to the *supervised* problem where alignments between input and output data are provided.

The main idea of UDATA is that data contained in attributes

Our research was sponsored by the Army Research Office and Defense Advance Research Projects Agency and was accomplished under Grant Number W911NF-18-1-0027. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office and Defense Advance Research Projects Agency or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

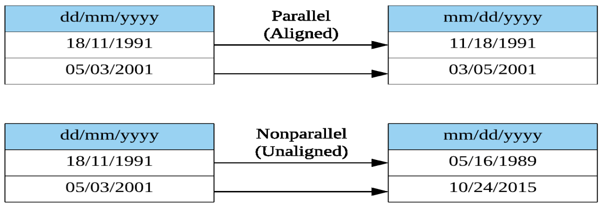

Fig. 1: Parallel and nonparallel input-output data

usually follow a set of common syntactic patterns. Therefore, with enough data, UDATA can infer the underlying patterns and thus leverage these patterns to learn the transformations. The problem then goes from the traditional value-to-value to the new pattern-to-pattern transformation. Since each pattern usually contains multiple string values, the similarity between the values in the pattern can provide the necessary information for UDATA to find the mappings between tokens. In the example of non-parallel data in Figure 1, if we have a sufficient amount of data, UDATA can conclude that the second set of digits in the input data should be mapped to the first set of digits in the output data, which share the same range of values.

Contributions — To summarize, we make the following key contributions in this paper:

- We formulate the problem of unsupervised data transformation, which has not been considered in previous research.
- We present a learning-based system to solve the unsupervised data transformation.
- We propose a validation algorithm to validate our transformation result with high accuracy.
- We evaluate our system in five different data sets; and the system achieves an average accuracy close to the previous best PBE systems, and it does so without any labeled data.

II. Motivating Example

In this section we provide a real-world scenario where existing approaches have difficulty. We then explain how our approach can resolve these issues.

Scenario — The NYC Open Data portal[1] contains multiple data sources about places of interest. Tables II and III show two data sources concerning `Hotel` and `Restaurant` found in NYC Open Data. However, there are inconsistencies in the data that may become problematic when merging records into a homogeneous database. For example, we see that the phone numbers and the website URLs in both `Hotel` and `Restaurant` need to be normalized. In addition, the location column in `Hotel` data source needs to be split into longitude and latitude for consistency with the `Restaurant` data.

Challenges — Manual or programming approaches may be viable solutions if the number of sources is small. However, as the number of data sources grows, these approaches become intractable. PBE and interactive data cleaning systems allow semi-supervised operation to reduce user effort. However, a system that requires little or no human interaction is the ideal.

Solution — UDATA provides the ability to solve the data transformation problem with minimal human interaction. Users only need to provide a set of examples in the desired formats. The system will then cluster the new data into different formats, learn the transformation programs to convert data into the predefined formats, and validate the output results. Users can optionally curate a representative set of examples to verify the results.

III. Overview

In data sources, data values are usually stored in string format. Special values such as numeric (e.g, a phone number or SSN without punctuation) and special characters can also be interpreted as string values. In this section we define syntactic patterns, our main representation for string values. We then define the problem of unsupervised data transformation and give an overview of our approach.

A. Syntactic Patterns

A syntactic pattern is a representation of the syntactic structure in a string value. In data transformation, Jin et al. [12] find that groups of adjacent characters from the same character class (e.g., digit, alphabet, alpha-numeric) usually share the same role in the transformation program. These groups of characters also have their meanings. For example, in the telephone number "(213) 775-2123," the three-digit "213" should be grouped and transformed together since the digits represent an area code, which is a meaningful concept. In this paper we call these character groups *tokens*.

A *token* is a constrained regular expression, which contains a regex type and a quantifier to indicate the length of the token. Table IV shows the list of regex types supported in UDATA. In addition, we support different types of punctuation and symbols (e.g., ".", ";", "/"), and each forms a separate regex type of its own. For example, a question mark "?" has its own regex type "?". A quantifier can be either a natural number or "+", which indicates that the length of the token is greater than one.

Example 1: A token of 3 digits is represented as "$\langle D3 \rangle$." A token of 1 alphanum character is shown as "$\langle AD \rangle$."

A *syntactic pattern* is a sequence of n tokens $\mathcal{S} = \{T_1, T_2, ..., T_n\}$ that represents the syntactic structure of a string value. A token T is denoted as "$\langle rq \rangle$" where r is the regex type and q is its quantifier. If the token length is one, we can remove the quantifier to simplify the representation. A set of string values can be described by many syntactic patterns since some token types are supersets of other token types. For instance, `Alphabet` is the superset of `Uppercase` and `Lowercase`. Therefore, determining the most suitable pattern for a set of string values is also a challenge that affects our transformation.

Example 2: "12/11/2017" can fit into several patterns such as "$\langle D2 \rangle / \langle D2 \rangle / \langle D4 \rangle$" or "$\langle D+ \rangle / \langle D+ \rangle / \langle D+ \rangle$"

[1]https://data.cityofnewyork.us

TABLE II: Data source concerning NYC hotels

Name	Phone	Website	Location
Paramount Hotel	(212) 764-5500	http://www.nycparamount.com	(40.759132, -73.986348)
Doubletree Guest Suites	2127191600	www.nycdoubletreehotels.com	(40.759055, -73.98471)
The Westin New York at Times Square	(212) 868-1900 ext 245	www.westinny.com	(40.757482, -73.988309)

TABLE III: Data source concerning NYC restaurants

Name	Address	Phone	Website	Latitude	Longitude
Sosa Borella	832 Eighth Ave	(212) 262-8282	http://www.sosaborella.com/	40.762444	-73.985983
Starbucks	871-879 Eighth Ave,# 871	2122467699	www.starbucks.com	40.763644	-73.985134

TABLE IV: Supported regex types

Regex Types	Regex	Symbol
`Uppercase`	r_U = `[A-Z]+`	`U`
`Lowercase`	r_l = `[a-z]+`	`L`
`Alphabet`	r_a = `[A-Za-z]+`	`A`
`Digit`	r_0 = `[0-9]+`	`D`
`Whitespace`	r_{ws} = `\s+`	`(S)`
`Alphanum`	r_{a0} = `[A-Za-z0-9]+`	`AD`
`Alnumspace`	r_{a0} = `[A-Za-z0-9\s]+`	`ADS`

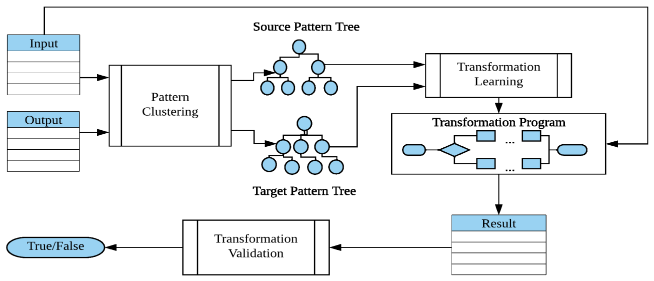

Fig. 2: Overall workflow of UDATA system

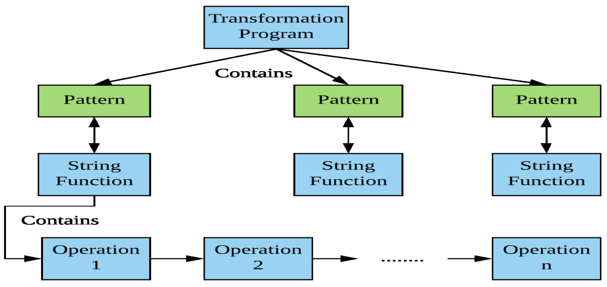

Fig. 3: Hierarchical structure of transformation programs

B. Data Transformation Problem

Based on aforementioned definitions, the non-parallel data transformation problem can be defined as follows:

Definition 3.1 (Data Transformation): Given a set of n strings to be transformed $\mathcal{S} = \{s_1, s_2, s_3, ..., s_n\}$ represented in m different patterns $\mathcal{P} = \{p_1, p_2, p_3, ..., p_m\}$ and a set of target example $\mathcal{T} = \{t_1, t_2, t_3, ..., t_n\}$ represented in k target patterns $\mathcal{P}' = \{p'_1, p'_2, p'_3, ..., p'_k\}$, generate a transformation program $\mathcal{L}$ to transform each string s_i in $\mathcal{S}$ to its corresponding string in any of the target pattern contained in $\mathcal{P}'$.

Our problem contains two different sets of inputs data: the set of data to be transformed $\mathcal{S}$ and the set of target patterns $\mathcal{P}'$. Both $\mathcal{S}$ and $\mathcal{P}'$ are meant to be extracted from a set of string values. The reason for supporting multiple target patterns is that it is difficult to represent semantically similar data in one pattern. A typical example is that of people's names. There are full names with two, three, or even more words, requiring different patterns to represent them.

C. Overall Approach

We propose a data transformation approach that operates in three phases: clustering, transformation, and validation. Figure 2 shows the overall model of our approach.

Pattern clustering — First, both input and output strings are processed by the pattern clustering module. In UDATA, we use the pattern clustering model from Jin et al.'s work [12].

Transformation learning — Using the syntactic patterns generated from the clustering model, the learning module synthesizes the transformation programs automatically. From the input and output patterns, tokens that contain similar string values are matched, and string manipulation functions can be applied when necessary. Since there are multiple source and target patterns, pattern matching also needs to be done after the token matching. We explain our transformation learning model in Section IV.

Transformation validation — The validation module verifies the transformation output and then identifies transformation failures for users' curation. Details of our validation algorithm can be found in Section V.

IV. TRANSFORMATION LEARNING

In this section we describe our overall transformation program structure and describe the algorithm that learns the program.

A. Transformation Program

In order to model the transformation program, we use a hierarchical structure as shown in Figure 3. The top level in the program $\mathcal{P}$ is a pattern-to-function mapping $\mathcal{M}$. $\mathcal{M}$ maps a specific original pattern p to a sting function F, which can transform p to the target format. A transformation function F is a sequence of string operations O, where each operation can apply to one token in the source pattern.

String operations — A string operation O has the form of $O = op(a_1, a_2, ..., a_n)$ where op is the operation name and a_1

to a_n are the required arguments. UDATA currently supports five string operations:

- `ConstStr`(s): return the constant s
- `Keep`(t): return string values of token t
- `ToLower`(t): return values of token t in lower case
- `ToUpper`(t'): return values of token t in upper case
- `Substr`(t, s, e): return substrings in range $(s..e)$ of values in token t. Negative values of s and e refer to backward direction.

Table V provides example outputs of four string operations (all except `Constant`). All the string operations except `Constant` require a source token in the main argument. In the remainder of the paper, we will refer to actual tokens by their absolute positions for convenience.

Example 3: `Keep`(2) means returning string values in the *second* source token. `Substr`(4, 1, 4) means generating the substrings $s[1:4]$ for all string values s in the *fourth* token.

B. Transformation Program Synthesis

We use a bottom-up approach to learn the transformation program. The input of our transformation algorithm is the source pattern tree T and the target pattern tree T'. First, the algorithm iterates through all the levels in the pattern trees. At each level, we find the best transformation function between each source pattern p and all target patterns p' (Section IV-C). The mapping in each level is built upon the source patterns and their corresponding transformation functions. Each level is then ranked by the average score of its transformation functions and the mapping from the best level is chosen as the final transformation program (Section IV-D).

C. String Function Generation

As we see in Figure 3, string functions are the key elements in a transformation program since the whole program structure is built on these functions. A valid string function should be able to transform values in a source pattern p to a target pattern p'. Therefore, a string function F is valid if and only if the number of operations in F is equal to the number of tokens in the target pattern p'. Since the transformed strings must follow the target pattern p', they will have the same number of tokens n and n operations are necessary to create n tokens.

Example 4: Figure 4 shows an example string function that contains four operations: `Substr`(1, 0, 1), `ConstStr`("."), `ConstStr`("(space)"), `Keep`(3).

To generate valid string functions for each source pattern, our string function generation method has three steps: candidate generation, operation scoring, and function synthesis.

Candidate generation — In the candidate generation step, we

TABLE V: Output of atomic string operations

Operation	Output
`ToUpper`("Lionel Messi")	LIONEL MESSI
`ToLower`("Lionel Messi")	lionel messi
`Substr`("Lionel Messi", 0, 3)	Lio
`Substr`("Lionel Messi", -5, -2)	Mes
`Keep`("Lionel Messi")	Lionel Messi

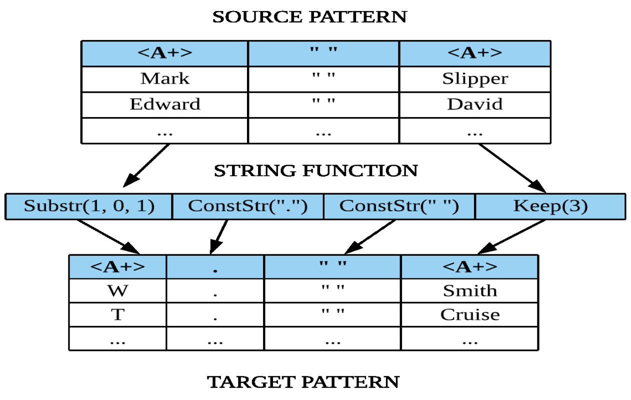

Fig. 4: String function with source and target patterns

generate a set of candidate operations for each pair of source and target tokens. An operation is a *candidate operation* if the source and target tokens satisfy the predefined conditions of that operation. Conditions of atomic string operations are:

- All string operations require that the quantifiers from two tokens be the same.
- `ConstStr`(s) is a valid candidate if all the values in the target token are s.
- `ToUpper`(t)/`ToLower`(t) is a valid candidate if the t's regex type is `Lowercase`/`Uppercase` and the target regex type is `Uppercase`/`Lowercase`.
- `Substr`(t, s, e) operations are valid candidates if all strings in the target token have the same length of $|e - s|$ and all string lengths of t are longer than e.

Example 5: Some candidate string functions between the two first tokens in Figure 4 are: `Keep`(1), `ToUpper`(1), `SubStr`(1, 0, 1), `SubStr`(1, 1, 2), `SubStr`(1, -2, -1).

Based on the conditions above, for each pair of tokens, UDATA iterates through the list of available operations. If the source and target tokens satisfy an operation condition, the operation will be added into the candidate set.

String operation scoring — The generated candidate operations need to be scored and ranked to find the best operation for every source token. In our ranking model, the score of a string function F to transform token t to token t' is the estimated similarity between the transformed values $O(v_t)$ and the target values $v_{t'}$. The similarity score is calculated using a logistic regression model following the idea from our previous work [19]. The score of a string function O transforming values in token t to token t' is calculated as follows:

$$\begin{aligned} score(O_{t,t'}) &= sim\left(O(v_t), v_{t'}\right) \\ &= sigmoid\left(w^T \mathfrak{f}\left(O(v_t), v_{t'}\right)\right) \end{aligned} \quad (1)$$

The score of an operation `ConstStr`(s) is 1 because the transformed values, which are all s, are the same as the values in the target token by operation condition. The feature vector $\mathfrak{f}\left(O(v_t), v_{t'}\right)$ consists of two different feature types:

- Semantic metrics: illustrate the overlap between values in two different subsets. Higher values of similarity indicate

TABLE VI: Score and operation matrix from Figure 4

	$\langle A+\rangle(1^{st}$ token)	$\langle A+\rangle(4^{th}$ token)
$\langle A+\rangle(1^{st}$ token)	0.11 `Substr(1, 0, 1)`	0.07 `Substr(3, 0, 1)`
$\langle A+\rangle(3^{rd}$ token)	0.07 `Keep(1)`	0.12 `Keep(2)`

that they are more likely to contain the same specific parts of information such as months, years, first names, or last names. We use Jaccard and tokenized Jaccard similarity as our features.
- Syntactic metrics: denote the degree of similarity in syntactic structures between the transformed source data and the target data. We use Jaccard similarity over the set of regex types between two patterns as our syntactic metrics.

The training data for our logistic regression model is generated exclusively from the source and target data given as the problem input. For each token in the source and target patterns, we split their values into three equal sets, which can be considered as training sets. Training samples are created by randomly sampling a pair of string sets. Using the assumption that there is no pair of tokens in the same pattern that contain the same piece of information, if the pair of string sets comes from the same token, the training label is True and vice versa. After training, the learning model can be used for string function scoring as shown in Equation 1.

String function synthesis — After scoring all the candidate operations between source and target tokens, we create a score matrix where rows indicate tokens in the source pattern, columns denote tokens in the target pattern and values are tuples of the best operations and their scores. `ConstStr` operations are excluded for simplification since they do not affect the problem of finding the optimal mapping. Table VI shows the score matrix generated from Figure 4.

The problem of finding the best alignment between tokens of the source and target patterns is a maximum assignment problem and can be solved using the Hungarian algorithm [17]. After determining the best mappings between source and target tokens together with the corresponding string functions, the string function F is then formed as a sequence of operations ordered by target token positions. For example, the final string function in Table VI after including `ConstStr` operations is the string function shown in Figure 4.

D. Pattern Mapping

Same-level pattern mapping — In every pattern level, UDATA generates the best string functions between two patterns based on results from previous steps. The input of UDATA's pattern mapping module is also a mapping between a pair of source and target patterns to the corresponding string functions. An example of pattern mapping is shown in Table VII.

TABLE VII: Score and string function matrix

	p'_1	p'_2
p_1	0.17 {`Substr(1,0,1)`},`Keep(2)`	0.03 {`Substr(2,0,1)`}
p_2	0.22 {`Keep(1)`, `Keep(2)`}	0.08 {`Keep(2)`}

As we see from the Table VII, each source pattern can be mapped to different target patterns using different string functions. UDATA selects the best string function for each source pattern to build the final pattern-to-function mappings. For instance, in Table VII, both p_1 and p_2 should be matched to p'_1 using the corresponding string functions.

Pattern-level ranking — Since the pattern mappings are restricted to apply to only one pattern level, it is necessary to find, score, and rank the transformation program among the pattern levels to select the final transformation program. In this step, we apply the same scoring idea from our string function scoring to score the pattern-level. In our pattern-level scoring model, the mapping score between two pattern-levels is the predicted similarity score between the transformed data and the target data. The transformed data can be obtained by applying the string function F of every pattern p to the set of source strings that fits in p. The pattern-level with the highest score is chosen to build the transformation program.

E. Scalable Transformation

As the volume of data increases, the computational cost of our system increases because of the complexity in our pattern clustering and transformation learning modules. Therefore, we develop an adaptive transformation approach to improve the scalability of our system.

Our adaptive transformation works as follows:
- First, sample a subset A from the set of input strings S.
- Run UDATA on A to learn the program P.
- Apply P to the rest of the data and create a subset S' of S for the values that P cannot transform.
- Repeat the process with S' as the input set of strings.
- Iterate until all of the strings in S are transformed.

The adaptive transformation method introduces a sample-build-test cycle that prevents building pattern trees and mapping the patterns repeatedly. Therefore, it prevents the computation cost of the UDATA from increasing exponentially.

V. TRANSFORMATION VALIDATION

Using unsupervised techniques to transform data comes with the usual caveat that mistakes will be overlooked. To verify the accuracy of the transformation system for real-world applications, we implement a validation algorithm that can alert a user as to whether the transformation program is likely to produce correct results. In an application where a handful of mistransformed data elements could lead to catastrophic loss, UDATA can still be used as the first module of the pipeline to reduce the workload for users in later stages. There are two main reasons for transformation failures: syntactic mismatches and semantic mismatches, and we propose an approach that can identify these types of failures separately.

Syntactic mismatches — The syntactic validation algorithm builds the pattern tree from our transformed data and compares it with the target pattern tree. If the transformed pattern tree is a subtree of the target pattern tree, which means they have similar syntactic structures, we can conclude that our transformation program is syntactically correct.

This method captures mistakes in syntactic structure and identifies structural mismatches between transformed strings and desired string values. However, it cannot provide perfect accuracy since there are also errors caused by the semantic ambiguities. In real-world problems, there are cases where finding the similarity between different sets of string values is ambiguous, causing our transformation learning to fail.

Semantic mismatches — To solve the issue above, we developed a semantic validation method that can report potential failures due to semantic mappings. The main reason for semantic mismatches in our system stems from the scoring model. Although we provide similarity features with different aspects to cover the semantic meanings contained in the string values, in many cases the amount of data is not sufficient to distinguish between different tokens.

Example 6: In Table VIII, the source data can be represented as "$\langle D3\rangle$-$\langle D3\rangle$-$\langle D3\rangle$" and the target data can be represented as "$\langle D3\rangle$". The transformation task is to extract the last three digits of from a nine-digit phone number.

TABLE VIII: Semantic ambiguity in the transformation

Source	Target
308-916-545	504
623-599-749	843
118-980-214	749

As we see from Table VIII, there is no overlap between the target data with any three-digit chunk in the source data. Moreover, the target data is syntactically similar to all the three-digit chunks in the source data. Therefore, there is not enough discrepancy between the source tokens for the system to infer the correct token mapping. Thus, the semantic validation module reports to users if top transformation programs have comparable scores. On the other hand, we also report scenarios where the highest similarity scores are zero. In these cases, our classification finds no overlap in semantic and syntactic features between the target and source data. As a result, the transformation programs cannot be learned.

VI. Evaluation

In this section we evaluate the transformation and validation methods in UData. Our system, datasets, and results are available online.[2]

[2]https://github.com/minhptx/ieee-bigdata2019-transformation

TABLE IX: Performance of transformation systems

	AAC	IJCAI	Sygus	Prog	NYC	Mean
IPBE	0.99	0.83	0.93	0.99	0.97	0.94
FlashFill	0.91	0.62	0.88	0.99	0.90	0.86
UData	0.93	0.71	0.56	0.93	0.88	0.80

A. Experimental Setup

Baseline systems — In the transformation evaluation, we compare UData performance with two state-of-the-art PBE systems: IPBE [28] and FlashFill [9]. Since IPBE and FlashFill can leverage parallel input-output examples, they serve as the upper-bound systems in our evaluation.

Datasets — We use five different datasets in the evaluation. Details of the five datasets are shown below:

- **AAC**: contains 173 transformation problems collected from 14 American art museums [15].
- **Sygus**: contains 27 transformation problems from the Syntax-guided Synthesis Competition over the years [12].
- **PregProg**: contains six transformation problems from previous papers on program synthesis [24].
- **IJCAI**: contains 36 transformation problems synthesized by Wu et al. [27].
- **NYC**: contains five transformations scenarios that we collected and labeled from NYC Open Data.[3]

Data organization — In our transformation accuracy experiments, we only use the first 1000 examples for evaluation since PBE systems are not designed to handle extensive datasets. Since all the data in our datasets are parallel data, we need to organize the data such that the three systems can use them properly. For each scenario, the parallel examples are divided into three partitions: $P1$, $P2$, and $P3$. For UData, we use $P1$ and $P2$ as our data to be transformed and $P3$ as example data in the target format. The output examples in $P1$ and $P2$ and the input examples in $P3$ are removed in our system. For FlashFill, the input-output examples of $P2$ are provided as parallel data while input examples of $P1$ serve as the transforming data. In IPBE, we limit the set of active learning examples to be in the same size of $P3$.

Computer Specifications — All the experiments are run on an i7-7700k machine with 32GB RAM and 256GB SSD.

B. Transformation Result

In the transformation experiment, we evaluate the transformation accuracy of UData, IPBE, and FlashFill. The transformation accuracy of each scenario is reported as the ratio between the number of correctly transformed values and the size of the input data.

As we see in Table IX, UData's accuracy is about 10% lower than IPBE and comparable with FlashFill in AAC, NYC and PregProg datasets. These three datasets are the real-world datasets in our evaluation data. Real-world scenarios

[3]https://opendata.cityofnewyork.us/

TABLE X: Validation peformance of UDATA

Dataset	AAC	IJCAI	Sygus	Prog	NYC	Mean
Recall	1.0	1.0	0.95	1.0	1.0	0.99

usually contain little semantic ambiguity in their tasks since data are stored in a way that is easy for humans to understand. There are cases where real-world data contains multiple syntactic patterns due to data entry errors or conflicts in data specifications. However, the number of syntactic patterns is usually low. These two reasons make UDATA perform better on real-world datasets.

In the Sygus and IJCAI datasets, there is a bigger difference between the performance of other systems and UDATA. The main reason for our low accuracy is that both the Sygus and IJCAI datasets are designed by humans for supervised program synthesis evaluation. Both the IJCAI and Sygus datasets contain very complicated transformation scenarios with many syntactic patterns in the input data. For example, there are ten scenarios in Sygus related to extracting or reordering phone numbers (Example 6). Not only are the digits in phone numbers distributed randomly, but there is almost no overlap between sets of three digits in different phone numbers. Therefore, our string function scoring models cannot determine the mapping between source and target tokens. On the other hand, supervised systems such as IPBE and FLASHFILL have full information about the alignments and can handle these scenarios better.

To summarize, we see that UDATA has comparable performance on real-world datasets with state-of-the-art systems, but has difficulty with performance on synthesized scenarios.

C. Validation Result

We evaluate our validation method and report the failure recall on the same datasets. We consider that a transformation is successful if all the string values are transformed correctly. Otherwise, the scenario is a failure. The reason for this strict condition is that even if only one string value is transformed incorrectly, user interaction is required to make corrections; thus, such failures should be reported to the users. As shown in Table X, the UDATA system achieves a high failure recall of 0.99 across different dataset, which guarantees that UDATA can identify almost every mistranformed output values for later curation.

It is clear from the transformation evaluation that UDATA still has a gap in performance compared with other supervised systems. However, by allowing fully unsupervised transformations UDATA provides a tradeoff between the need for training data and higher accuracy. With a transformation accuracy of 0.80 and the validation recall of 0.99, in normal transformation applications, UDATA can be used as the first stage in a pipeline to transform a large portion of the data and reduce the workload for users in later phases.

D. Running Time

We report the running time of the UDATA, IPBE and FLASHFILL in Table XI. As we see from the table, UDATA learns the transformations within two seconds in simple datasets. In more complicated ones such as IJCAI and Sygus, the source input data can have a high number of source patterns in some scenarios, which increases the size of the generated pattern tree and affects the performance of the system. IPBE uses an adaptive active learning method to iteratively learn and suggest ideal examples for annotation, which is time-consuming and results in its highest running time in the evaluation. FLASHFILL does not have a built-in pattern clustering module and thus the system attempts to learn the transformation program from all input/output examples at once. As a result, FLASHFILL achieves lower performance in datasets with various format but has the lowest running time across all the systems.

TABLE XI: Average running time of UDATA

	AAC	IJCAI	Sygus	Prog	NYC	Mean
IPBE	3.9s	36.8s	1.2s	4.2s	36s	16.4s
FLASHFILL	2.9s	1.6s	1.5s	1.4s	1.3s	1.7s
UDATA	7.6s	17.4s	0.6s	0.4s	1.8s	5.6s

VII. RELATED WORK

Data transformation — PBE systems are often used because they can automatically perform transformations given a small number of output samples. FLASHFILL [9] designed a string expression language and a set of transformation programs to learn the transformations based on input-output pair examples. Wu et al. [27], [28] extended FLASHFILL by providing a partitioning method to partition string values as well as introducing an incremental process to eliminate incorrect transformation programs while users are inputting the examples. Recently, Singh [22] and Jin [12] proposed a semi-supervised method to build syntactic patterns of input data to support transformation learning. Another approach for data transformation that has also received attention in recent years is machine learning. Shu [21] proposed a method of using deep neural networks to learn the transformation programs. Wang et al. [26] used a probabilistic approach to model and learn the string transformations while Devlin et al. [6] take advantage of deep learning models. Since both PBE and machine learning systems take advantage of a small set of labeled data combined with a large amount of unlabeled data, these systems can be considered semi-supervised systems, compared to our unsupervised system.

Another direction of data transformation focuses on transforming strings based on their semantic meaning. Singh et al. [23] learn the semantic transformations by using the information that is available within a given data source. DATAXFORMER [2] learns the semantic transformations from the set of web tables and forms available on the web. Singh et al. [25] learned semantic transformations of different data types based on a set of operations predefined by developers. To date, our system cannot learn semantic transformations, but it is one of the goals in future work.

Data cleaning — Interactive data cleaning is another common approach since it can analyze the data and provide useful tools to create transform rules faster. Raman et al. [20] proposed an interactive data cleaning system that analyzes data and suggests transformation rules for users to select. Wrangler [13] provides a user-friendly interface for creating scripts that can be used to clean and transform data. ActiveClean [16] is another interactive data cleaning system that combines adaptive and rule-based models to suggest dirty data for users' curation based on previous dirty and cleaned data. Although interactive cleaning systems reduce users' time and effort to create the transformations, they do not address the problem of human interaction, as the UDATA system does.

Data profiling — There is also research in the field of data profiling. Saswat et al. [18] implemented FLASHPROFILER, which is a syntactic profiling system that discovers syntactic patterns from a set of string values. Andrew et al. [11] proposed a method that extracts syntactic patterns from databases in three different layers: branch, token, and symbol layers. LEARNPADS [7] proposed a learning algorithm using statistics over symbols and tokenized data chunks to discover pattern structure. However, these systems assume that attribute values follow the same pattern structure, which is hard to ensure for noisy datasets collected from multiple sources. In addition, there is other work focusing on analyzing and profiling data attributes (i.e., columns) from relational databases [10], [5], RDF triple stores [1], [3], and XML files [4].

VIII. CONCLUSION AND FUTURE WORK

In this paper we presented a novel unsupervised approach to infer expressive syntactic structures from data, learn the transformation between source and target data using these syntactic structures, and validate the transformed results. Without human interaction, our system can transform attribute data from different formats into one desired format. Therefore, our system can be applied to problems that require fully automatic processing. For example, we have used UDATA for automatically normalizing data in the problem of automatic spatial and temporal indexing for many data sources [14].

In the future we plan to support semantic transformation in UDATA. For example, we plan to extract transformations from web tables to provide human knowledge in our transformation programs. Moreover, we plan to extend the set of supported transformation functions so that the UData system can perform a wider set of transformations.

REFERENCES

[1] Z. Abedjan, T. Grütze, A. Jentzsch, and F. Naumann. Mining and profiling rdf data with prolod++. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1198–1201, 2014.

[2] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. Dataxformer: A robust transformation discovery system. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 1134–1145. IEEE, 2016.

[3] S. Auer, J. Demter, M. Martin, and J. Lehmann. Lodstats–an extensible framework for high-performance dataset analytics. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 353–362. Springer, 2012.

[4] G. J. Bex, F. Neven, and S. Vansummeren. Inferring xml schema definitions from xml data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 998–1009. VLDB Endowment, 2007.

[5] X. Chu, I. F. Ilyas, P. Papotti, and Y. Ye. Ruleminer: Data quality rules discovery. In *2014 IEEE 30th International Conference on Data Engineering*, pages 1222–1225. IEEE, 2014.

[6] J. Devlin, J. Uesato, S. Bhupatiraju, R. Singh, A.-r. Mohamed, and P. Kohli. Robustfill: Neural program learning under noisy i/o. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 990–998. JMLR. org, 2017.

[7] K. Fisher, D. Walker, and K. Q. Zhu. Learnpads: automatic tool generation from ad hoc data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1299–1302. ACM, 2008.

[8] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, volume 46, pages 317–330. ACM, 2011.

[9] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. *Communications of the ACM*, 55(8):97–105, 2012.

[10] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, et al. The madlib analytics library: or mad skills, the sql. *Proceedings of the VLDB Endowment*, 5(12):1700–1711, 2012.

[11] A. Ilyas, J. M. da Trindade, R. C. Fernandez, and S. Madden. Extracting syntactical patterns from databases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 41–52. IEEE, 2018.

[12] Z. Jin, M. J. Cafarella, H. V. Jagadish, S. Kandel, and M. Minar. Unifacta: Profiling-driven string pattern standardization. *CoRR*, abs/1803.00701, 2018.

[13] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

[14] C. A. Knoblock, A. R. Joshi, A. Megotia, M. Pham, and C. Ursaner. Automatic spatio-temporal indexing to integrate and analyze the data of an organization. In *Proceedings of the 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, page 7. ACM, 2017.

[15] C. A. Knoblock, P. Szekely, E. Fink, D. Degler, D. Newbury, R. Sanderson, K. Blanch, S. Snyder, N. Chheda, N. Jain, et al. Lessons learned in building linked data for the american art collaborative. In *International Semantic Web Conference*, pages 263–279. Springer, 2017.

[16] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12):948–959, 2016.

[17] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

[18] S. Padhi, P. Jain, D. Perelman, O. Polozov, S. Gulwani, and T. Millstein. Flashprofile: Interactive synthesis of syntactic profiles. *arXiv preprint arXiv:1709.05725*, 2017.

[19] M. Pham, S. Alse, C. A. Knoblock, and P. Szekely. Semantic labeling: a domain-independent approach. In *International Semantic Web Conference*, pages 446–462. Springer, 2016.

[20] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.

[21] C. Shu and H. Zhang. Neural programming by example. In *AAAI*, pages 1539–1545, 2017.

[22] R. Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment*, 9(10):816–827, 2016.

[23] R. Singh and S. Gulwani. Learning semantic string transformations from examples. *Proceedings of the VLDB Endowment*, 5(8):740–751, 2012.

[24] R. Singh and S. Gulwani. Predicting a correct program in programming by example. In *International Conference on Computer Aided Verification*, pages 398–414. Springer, 2015.

[25] R. Singh and S. Gulwani. Transforming spreadsheet data types using examples. In *ACM SIGPLAN Notices*, volume 51, pages 343–356. ACM, 2016.

[26] Z. Wang, G. Xu, H. Li, and M. Zhang. A probabilistic approach to string transformation. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1063–1075, 2014.

[27] B. Wu and C. A. Knoblock. An iterative approach to synthesize data transformation programs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[28] B. Wu and C. A. Knoblock. Maximizing correctness with minimal user effort to learn data transformations. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 375–384. ACM, 2016.